

# **Conflict-Driven Learning in AI Planning State-Space Search**

A dissertation submitted towards the degree  
*Doctor of Natural Sciences (Dr. rer. nat.)*  
of the Faculty of Mathematics and Computer Science  
of Saarland University

by

Marcel Steinmetz

Saarbrücken, 2022

<b>Date of the Colloquium</b>	December 5, 2022
<b>Dean of the Faculty</b>	Prof. Dr. Jürgen Steimle
<b>Chair of the Committee</b>	Prof. Dr. Verena Wolf
<b>Reviewers</b>	Prof. Dr. Jörg Hoffmann
	Prof. Dr. Holger Hermanns
	Prof. Dr. Malte Helmert
<b>Academic Assistant</b>	Dr. Michaela Klauck

# Abstract

Many combinatorial computation problems in computer science can be cast as a reachability problem in an implicitly described, potentially huge, graph – the state space. State-space search is a versatile and widespread method to solve such reachability problems, but it requires some form of guidance to prevent exploring that combinatorial space exhaustively. Conflict-driven learning is an indispensable search ingredient for solving constraint satisfaction problems (most prominently, Boolean satisfiability). It guides search towards solutions by identifying conflicts during the search, i.e., search branches not leading to any solution, learning from them knowledge to avoid similar conflicts in the remainder of the search.

This thesis adapts the conflict-driven learning methodology to more general classes of reachability problems. Specifically, our work is placed in AI planning. We consider goal-reachability objectives in classical planning and in planning under uncertainty. The canonical form of “conflicts” in this context are dead-end states, i.e., states from which the desired goal property cannot be reached. We pioneer methods for learning sound and generalizable dead-end knowledge from conflicts encountered during forward state-space search. This embraces the following core contributions:

- (i) When acting under uncertainty, the presence of dead-end states may make it impossible to satisfy the goal property with absolute certainty. The natural planning objective then is MaxProb, maximizing the probability of reaching the goal. However, algorithms for MaxProb probabilistic planning are severely underexplored. We close this gap by developing a large design space of probabilistic state-space search methods, contributing new (a) search algorithms, (b) admissible state-space reduction techniques, and (c) goal-probability bounds suitable for heuristic state-space search. We systematically explore this design space through an extensive empirical evaluation.
- (ii) The key to our conflict-driven learning algorithm adaptation are unsolvability detectors, i.e., goal-reachability overapproximations. We design three complementary families of such unsolvability detectors, building upon known techniques: (a) critical-path heuristics, (b) linear-programming-based heuristics, and (c) dead-end traps. We develop search methods to identify conflicts in deterministic and probabilistic state spaces, and we develop suitable refinement methods for the different unsolvability detectors so to recognize these states. Arranged in a depth-first search, our techniques approach the elegance of conflict-driven learning in constraint satisfaction, featuring the ability to learn to refute search subtrees, and intelligent backjumping to the root cause of a conflict.
- (iii) We provide a comprehensive experimental evaluation, demonstrating that the proposed techniques yield state-of-the-art performance for (a) finding plans for solvable classical planning tasks, (b) proving classical planning tasks unsolvable, and (c) solving MaxProb in probabilistic planning, on benchmarks where dead-end states abound.



# Zusammenfassung

Viele kombinatorisch komplexe Berechnungsprobleme in der Informatik lassen sich als Erreichbarkeitsprobleme in einem implizit dargestellten, potenziell riesigen, Graphen – dem Zustandsraum – verstehen. Die Zustandsraumsuche ist eine weit verbreitete Methode, um solche Erreichbarkeitsprobleme zu lösen. Die Effizienz dieser Methode hängt aber maßgeblich von der Verwendung strikter Suchkontrollmechanismen ab. Das konfliktgesteuerte Lernen ist eine essenzielle Suchkomponente für das Lösen von Constraint-Satisfaction-Problemen (wie dem Erfüllbarkeitsproblem der Aussagenlogik), welches von Konflikten, also Fehlern in der Suche, neue Kontrollregeln lernt, die ähnliche Konflikte zukünftig vermeiden.

In dieser Arbeit erweitern wir die zugrundeliegende Methodik auf Zielerreichbarkeitsfragen, wie sie im klassischen und probabilistischen Planen, einem Teilbereich der Künstlichen Intelligenz, auftauchen. Die kanonische Form von “Konflikten” in diesem Kontext sind sog. Sackgassen, Zustände, von denen aus die Zielbedingung nicht erreicht werden kann. Wir präsentieren Methoden, die es ermöglichen, während der Zustandsraumsuche von solchen Konflikten korrektes und verallgemeinerbares Wissen über Sackgassen zu erlernen. Unsere Arbeit umfasst folgende Beiträge:

- (i) Wenn der Effekt des Handelns mit Unsicherheiten behaftet ist, dann kann die Existenz von Sackgassen dazu führen, dass die Zielbedingung nicht unter allen Umständen erfüllt werden kann. Die naheliegendste Planungsbedingung in diesem Fall ist MaxProb, das Maximieren der Wahrscheinlichkeit, dass die Zielbedingung erreicht wird. Planungsalgorithmen für MaxProb sind jedoch wenig erforscht. Um diese Lücke zu schließen, erstellen wir einen umfangreichen Bausatz für Suchmethoden in probabilistischen Zustandsräumen, und entwickeln dabei neue (a) Suchalgorithmen, (b) Zustandsraumreduktionsmethoden, (c) und Abschätzungen der Zielerreichbarkeitswahrscheinlichkeit, wie sie für heuristische Suchalgorithmen gebraucht werden. Wir explorieren den resultierenden Gestaltungsraum systematisch in einer breit angelegten empirischen Studie.
- (ii) Die Grundlage unserer Adaption des konfliktgesteuerten Lernens bilden Unerreichbarkeitsdetektoren. Wir konzipieren drei Familien solcher Detektoren basierend auf bereits bekannten Techniken: (a) Kritische-Pfad Heuristiken, (b) Heuristiken basierend auf linearer Optimierung, und (c) Sackgassen-Fallen. Wir entwickeln Suchmethoden, um Konflikte in deterministischen und probabilistischen Zustandsräumen zu erkennen, sowie Methoden, um die verschiedenen Unerreichbarkeitsdetektoren basierend auf den erkannten Konflikten zu verfeinern. Instanziiert als Tiefensuche weisen unsere Techniken ähnliche Eigenschaften auf wie das konfliktgesteuerte Lernen für Constraint-Satisfaction-Problemen.
- (iii) Wir evaluieren die entwickelten Methoden empirisch, und zeigen dabei, dass das konfliktgesteuerte Lernen unter gewissen Voraussetzungen zu signifikanten Suchreduktionen beim Finden von Plänen in lösbaren klassischen Planungsproblemen, Beweisen der Unlösbarkeit von klassischen Planungsproblemen, und Lösen von MaxProb im probabilistischen Planen, führen kann.



## Acknowledgments

First and foremost, I would like to thank Jörg Hoffmann for offering me the opportunity of pursuing my PhD studies under his supervision, for his incredible patience for me to gather all our results and bring them onto paper (aka writing this thesis), for his seemingly never ending supply of cool new research ideas (which occasionally can cross one's work plans though), and of course for his great guidance, his advice, and all the things I could learn from the work with him during all the time since my bachelor's studies.

I want to thank Malte Helmert and Holger Hermanns for agreeing to cram the review of this thesis into their already tightly-packed schedules.

I want to thank all current and former FAI group members for an interesting time and a great working atmosphere. Thanks to Álvaro, Daniel, Daniel, Dan, Julia, Marcel, Max, Rebecca, and Thorsten! I also want to thank everyone with whom I had the pleasure to collaborate. I want to thank especially Patrick and Robert, with whom I frequently shared an office at CISP, and Holger and Michaela for the many fruitful collaborations over the past years.

Last but not least, I would like to thank my family, my grand-parents in particular, my parents, and my sister for their continuous support and encouragement ever since.



# Contents

<b>1. Introduction</b>	<b>1</b>
1.1. Planning, State-Space Search, and Conflict-Driven Learning . . . . .	1
1.2. Example Walk Through . . . . .	3
1.3. Contributions . . . . .	5
1.4. Outline . . . . .	8
1.5. Publications . . . . .	9
1.5.1. Core Publications and Relation to Prior Work . . . . .	9
1.5.2. Further Publications . . . . .	11
 <b>I. Preliminaries</b>	 <b>21</b>
<b>2. Classical Planning</b>	<b>23</b>
2.1. Planning Task Formalisms . . . . .	23
2.1.1. STRIPS Planning . . . . .	23
2.1.2. FDR Planning . . . . .	25
2.1.3. Planning Objectives & Complexity . . . . .	26
2.2. Transition Systems & State Spaces . . . . .	26
2.3. Heuristic Search . . . . .	27
<b>3. Strongly Connected Components</b>	<b>29</b>
<b>4. Linear Programming</b>	<b>31</b>
 <b>II. Conflict-Driven Learning in Classical-Planning State-Space Search</b>	 <b>33</b>
<b>5. Conflict Identification in Forward State-Space Search</b>	<b>35</b>
5.1. Generic Search Algorithm . . . . .	35
5.2. Depth-First Search . . . . .	40
<b>6. Critical-Path Heuristics: Conflict Refinement &amp; NoGood Learning</b>	<b>43</b>
6.1. Preliminaries . . . . .	44
6.2. Conflict Analysis & Refinement . . . . .	46
6.2.1. Path-Cut Refinement . . . . .	46
6.2.2. Neighbors Refinement . . . . .	49
6.2.3. Learning Effectiveness: Worst-Case Analysis . . . . .	54
6.3. Critical-Path NoGoods . . . . .	54
6.3.1. Equivalent $\mathcal{U}^C$ NoGood Characterization . . . . .	55
6.3.2. NoGood Learning . . . . .	57
6.3.3. Experimental Evaluation . . . . .	59

6.4. Experimental Evaluation . . . . .	62
6.4.1. Algorithm Configurations, Competing Approaches, and Benchmarks . . . . .	62
6.4.2. Dead-End Detection in Solvable Planning Tasks . . . . .	65
6.4.3. Proving Unsolvability . . . . .	74
6.4.4. Generating Unsolvability Certificates . . . . .	80
<b>7. LP Heuristics Over Conjunctions: Compilation, Convergence &amp; Conflict Refinement</b>	<b>83</b>
7.1. Preliminaries . . . . .	84
7.2. The State-Equation Heuristic . . . . .	86
7.2.1. Definition . . . . .	86
7.2.2. The State Equation over Conjunctions . . . . .	87
7.3. Potential Heuristics . . . . .	90
7.3.1. Admissible Fact Potential Heuristics . . . . .	91
7.3.2. Potential Heuristics Over Arbitrary Conjunctions . . . . .	92
7.3.3. Convergence . . . . .	95
7.3.4. Relation to the State Equation . . . . .	96
7.4. Conflict Analysis: Refining the State Equation . . . . .	97
7.5. Learning Effectiveness: State Equation vs. Critical-Path Heuristics . . . . .	99
7.6. Experimental Evaluation . . . . .	100
7.6.1. Experiment Setup . . . . .	100
7.6.2. Conjunction Set Refinement: On the Order In Which To Remove Facts . . . . .	101
7.6.3. Discussion . . . . .	101
<b>8. On Unsolvability Detectors, the Traps They Set, and Trap Learning</b>	<b>105</b>
8.1. Preliminaries . . . . .	106
8.2. Unsolvability Detectors and the Traps they Set . . . . .	107
8.3. Offline Construction . . . . .	109
8.4. Trap Learning: Conflict Analysis & Refinement . . . . .	110
8.4.1. Overall Refinement Procedure . . . . .	110
8.4.2. Generalizing $\mathcal{U}$ -Traps . . . . .	111
8.5. Learning Effectiveness: Comparison to the State-Equation and Critical-Path Heuristics . . . . .	112
8.6. Experimental Evaluation . . . . .	114
8.6.1. Experiment Setup . . . . .	114
8.6.2. Unsolvability Detectors . . . . .	115
8.6.3. Discussion . . . . .	115
<b>9. Discussion</b>	<b>119</b>
9.1. Summary . . . . .	119
9.2. Related Work . . . . .	120
9.2.1. Learning Search Control Knowledge . . . . .	120
9.2.2. Dead-End Detection and Proving Unsolvability . . . . .	121
9.3. Future Work . . . . .	122
<b>III. State-Space Search Methods for Goal-Probability Analysis in Probabilistic Planning</b>	<b>125</b>
<b>10. Probabilistic Planning</b>	<b>127</b>
10.1. Probabilistic FDR Planning Tasks . . . . .	127

10.2. Markov Decision Processes . . . . .	129
10.3. Probabilistic State Space . . . . .	131
10.4. Policies . . . . .	132
10.5. Stochastic Shortest Path Problems . . . . .	132
10.6. Goal-Probability Value Function . . . . .	132
10.7. Goal-Probability Objectives & Complexity . . . . .	134
<b>11. Fundamental Algorithms</b>	<b>135</b>
11.1. Linear Program Representation . . . . .	135
11.2. Value Iteration . . . . .	136
11.2.1. Principles . . . . .	137
11.2.2. Termination . . . . .	139
11.2.3. Topological Value Iteration . . . . .	141
11.2.4. Policy Extraction . . . . .	142
<b>12. MDP Heuristic Search</b>	<b>147</b>
12.1. Introductory Example . . . . .	148
12.2. Acyclic MDPs . . . . .	150
12.2.1. AO* . . . . .	150
12.2.2. Exhaustive AO* . . . . .	153
12.3. General MDPs via FRET . . . . .	154
12.3.1. The Find-and-Revise Schema . . . . .	154
12.3.2. LRTDP . . . . .	155
12.3.3. Depth-First Heuristic Search . . . . .	158
12.3.4. FRET Framework . . . . .	162
12.4. General MDPs without FRET . . . . .	169
12.4.1. Trap-Aware LRTDP . . . . .	169
12.4.2. Trap-Aware Depth-First Heuristic Search . . . . .	171
12.4.3. Anytime Exhaustive Depth-First Search . . . . .	172
12.4.4. I-Dual . . . . .	173
<b>13. Heuristic Search Tie-Breaking Strategies</b>	<b>175</b>
<b>14. Goal-Probability Occupation-Measure Heuristics</b>	<b>179</b>
14.1. Goal-Probability Occupation-Measure Heuristic . . . . .	180
14.2. Probabilistic Operator-Counting Heuristic . . . . .	185
<b>15. State-Space Reduction Techniques</b>	<b>193</b>
15.1. Probabilistic Bisimulation . . . . .	193
15.2. Dead-End Pruning . . . . .	195
15.3. Budget Pruning via Landmarks: Heuristic vs. Budget Reduction . . . . .	196
<b>16. Experimental Evaluation</b>	<b>199</b>
16.1. Implementation . . . . .	199
16.2. Benchmark Suite . . . . .	200
16.3. Evaluating the State of the Art in Goal-Probability Planning . . . . .	201
16.3.1. Acyclic Planning . . . . .	201
16.3.2. Cyclic Planning . . . . .	212

16.4. Occupation-Measure and Operator-Counting Heuristics for Goal Probability . . . . .	222
16.4.1. Experiment Setup . . . . .	222
16.4.2. Discussion . . . . .	225
<b>17. Discussion</b>	<b>229</b>
17.1. Summary . . . . .	229
17.2. Related Work . . . . .	230
17.3. Future Work . . . . .	232
<b>IV. Conflict-Driven Learning in Probabilistic-Planning State-Space Search</b>	<b>235</b>
<b>18. Conflict-Driven Learning in MDP State-Space Search for Goal-Probability Analysis</b>	<b>237</b>
18.1. Preliminaries . . . . .	237
18.2. Conflict Characterization . . . . .	238
18.3. Conflict-Driven Learning in MDP Heuristic Search . . . . .	239
18.3.1. Utilizing the Learned Knowledge . . . . .	239
18.3.2. Conflict Identification . . . . .	241
18.4. Conflict-Driven Learning in Topological VI & Anytime DFS . . . . .	242
18.5. Experimental Evaluation . . . . .	243
18.5.1. Experiment Setup . . . . .	243
18.5.2. Coverage Analysis . . . . .	244
18.5.3. Search Reduction . . . . .	246
18.5.4. Performance Analysis . . . . .	248
18.5.5. Search Algorithm Comparison . . . . .	251
18.6. Discussion . . . . .	252
18.6.1. Summary and Related Work . . . . .	252
18.6.2. Future Work . . . . .	253
<b>V. Conclusion</b>	<b>255</b>
<b>19. Conclusion</b>	<b>257</b>
<b>Appendix</b>	<b>261</b>
<b>A. Supplemental Results</b>	<b>261</b>
A.1. Offline $\mathcal{U}$ -Trap Coverage . . . . .	261
A.2. MaxProb Coverage for DFHS Variants . . . . .	263
<b>B. Proofs of Part II</b>	<b>265</b>
B.1. Conflict Identification in Forward State-Space Search . . . . .	265
B.1.1. Correctness of the Known-Dead-End Labeling Procedure (Theorem 5.1) . . . . .	265
B.2. Critical-Path Heuristics: Conflict Refinement & NoGood Learning . . . . .	265
B.2.1. Correctness of Path-Cut Refinement Algorithm (Theorem 6.1) . . . . .	265
B.2.2. Proof that Size-Minimal $\mathcal{U}^C$ Neighbor-Conflict Extraction is NP-HARD . . . . .	266
B.2.3. Worst-Case $\mathcal{U}^C$ Refinement Example (Proposition 6.2) . . . . .	267

B.3. LP Heuristics Over Conjunctions: Compilation, Convergence & Conflict Refinement . . . . .	268
B.3.1. $\Pi^C$ Dominates Partial Variable Merges (Theorem 7.1) . . . . .	268
B.3.2. Partial Variable Merges May Need Exponentially More Conjunctions Than $\Pi^C$ (Theorem 7.2) . . . . .	270
B.3.3. Potential Heuristic Convergence (Theorem 7.5) . . . . .	272
B.4. On Unsolvability Detectors, the Traps They Set, and Trap Learning . . . . .	277
B.4.1. Learning Effectiveness: Comparison to the State-Equation and Critical-Path Heuristics (Propositions 8.1 and 8.2 and 8.3) . . . . .	277
<b>C. Proofs of Part III</b> . . . . .	<b>279</b>
C.1. Fundamental Algorithms . . . . .	279
C.1.1. Proof that Bellman Residuals are Monotonically Decreasing . . . . .	279
C.1.2. Correctness Proof of VI Policy Extraction (Theorem 11.6 and Lemma 11.1) . . . . .	279
C.2. MDP Heuristic Search . . . . .	281
C.2.1. Relation Between SSPs and the Existence of Traps (Theorem 12.6) . . . . .	281
C.2.2. Collapsing Traps Preserves Goal Probabilities (Theorem 12.7) . . . . .	282
C.2.3. Correctness of FRET- $V$ (Theorem 12.8) . . . . .	283
C.2.4. Comparison Between FRET- $V$ and FRET- $\pi$ (Theorem 12.12) . . . . .	284
C.3. Goal-Probability Occupation-Measure Heuristics . . . . .	286
C.3.1. Equations for the $H^{\text{gpom}}$ Monotonicity Proof (Theorem 14.1) . . . . .	286
C.3.2. Dominance Relation Between $H^{\text{gpom}}$ and $H_{\text{seq}}^{\text{g poc}}$ (Theorem 14.4) . . . . .	288
C.4. Goal-Probability Fact Potential Heuristics . . . . .	290
C.5. Goal-Probability Occupation-Measure Heuristics vs. Multiplicative Goal-Probability PDBs . . . . .	291
<b>Bibliography</b> . . . . .	<b>293</b>



# 1. Introduction

Planning is the process of us humans to think ahead of acting by reasoning over knowledge gained in the past. Fundamental to this process is our ability to learn, i.e., to constantly improve our decision-making by acquiring new knowledge. As per these definitions, planning and learning constitute cornerstones in our common understanding of human-alike intelligence. Hence not surprisingly, planning and learning have ever since been two central strands of research in the field of Artificial Intelligence (AI). The aim of this thesis is the development of AI planning methods that continuously improve their performance along the solution process by learning from experience gained through that process.

## 1.1. Planning, State-Space Search, and Conflict-Driven Learning

The endeavor to obtain machines with human-alike planning capabilities has its origin in the early works on *general problem solvers* (Newell et al., 1959; Ernst and Newell, 1971). The core idea – the development of computer programs with the ability to reason in a general, application unspecific, manner – has prevailed until today in the quest for domain-independent planning systems. From an algorithmic point of view, automated planning (henceforth simply planning) (Ghallab et al., 2004) formulates the problem of, given some formulation of the goal that is to be achieved and the actions at disposal to accomplish this task, synthesizing a strategy that decides when to execute which action in order to satisfy the stated goal. Solving general classes of planning tasks is made possible by having access to a formal model of the world in which one is acting, i.e., a mathematical characterization of the world’s states and its dynamics in terms of how the world states change as one is acting. The model serves as basis to predict and to anticipate the possible outcomes of the actions, which when put in relation to the goal specification, allows reasoning about the course of actions achieving the objective without any prior knowledge of the particular application. By making different assumptions and restrictions on the world models, one arrives at different variants of the planning problem.

Amongst the simplest and well-studied variants is classical planning (Fikes and Nilsson, 1971), which assumes finite and discrete dynamics, perfect sensing (the world state is unambiguously known at all time), that the outcome of every action in any world state is deterministic, and that the world is not influenced by exogenous sources (the world states only change as an effect of our actions). Despite this simplicity, classical planning has become a popular approach for many applications as diverse as single-player puzzle games like Sokoban and the sliding tile puzzle, the dynamic configuration of complex printer systems (Ruml et al., 2011), greenhouse logistics (Helmert and Lasinger, 2010), robotics (Beetz, 2002; Hofmann et al., 2016), or even space mission planning (Pell et al., 1998; Backes et al., 1999).

Classical planning is well suited if the environment is controllable and predictable. However, in many real-world scenarios this is not the case: sensing may not be perfect, actions may fail (e.g., a planetary rover may get stuck), exogenous sources may affect the world’s state (e.g., surrounding traffic), and the result of some actions may simply not be deterministic (e.g., rolling a dice). In all cases where such stochastic behavior could have fatal consequences (e.g., leaving the planetary rover in a cave with no energy left), one

should take into account the uncertainty during planning already. Probabilistic planning deals with models that exhibit such random phenomena. One of the most widely-used modeling frameworks to this end are Markov Decision Processes (MDPs) (Howard, 1960; Puterman, 1994). MDPs extend the classical planning view on the world by the support of action-outcome uncertainty. MDPs have been utilized extensively in applications like unmanned aerial vehicle mission planning (Jeong et al., 2014; Feng et al., 2015), search-and-rescue missions (Teichteil-Königsbuch and Fabiani, 2006; Pineda et al., 2015), military operation planning (Aberdeen et al., 2004), controlling satellites (Pováda et al., 2019), and for modeling particular tasks in autonomous driving (Abbeel et al., 2008; Brechtel et al., 2011; Wei et al., 2011).

The curse of generality is complexity. Domain-independent planning is an exemplary demonstration of this principle. Despite the restrictive assumptions, common classical-planning modeling formalisms still give rise to PSPACE-COMPLETE decision problems (Bylander, 1994; Bäckström and Nebel, 1995). Extending these formalisms just by the ability to specify action-outcome uncertainty pushes the complexity of planning even further to the class of EXPTIME-COMPLETE problems (Littman, 1997). Dealing with such challenging problems effectively requires to design algorithms capable of automatically identifying and utilizing as much of a task’s structure as possible.

A generic algorithmic paradigm that has proved particularly successful is *forward state-space search* guided by a *heuristic function* (Bonet and Geffner, 2001; Hoffmann and Nebel, 2001; Bonet and Geffner, 2005; Richter and Westphal, 2010; Kolobov et al., 2012b; Trevizan et al., 2016). The state space explicates the behavior implicitly encoded by a planning task’s model, a step that however comes with an exponential blow-up commonly known as the state explosion problem. Searching the state space exhaustively is not possible for any but the simplest planning tasks. Heuristic functions help to narrow down the search to states that appear particularly likely to be part of the desired solution. The automatic construction of heuristic functions from the planning task description has been subject of decades of research, during which many sophisticated approaches have been introduced (e.g., Haslum and Geffner, 2000; Hoffmann and Nebel, 2001; Edelkamp, 2002; Bonet and Geffner, 2005; van den Briel et al., 2007; Helmert and Domshlak, 2009; Teichteil-Königsbuch et al., 2011; Helmert et al., 2014; Trevizan et al., 2017b).

Heuristic search approaches typically follow a *static* procedure. The heuristic function is constructed offline, before search is started, and remains fixed until the end of the search. This has the risk that if the chosen heuristic function happens to not be informative for the particular planning task one is facing, heuristic search will not be effective, in the worst case doing no better than an exhaustive state-space exploration.

An alternative technique to steering search known from the area of constraint satisfaction problems (particularly Boolean satisfiability) is *conflict-driven learning* (Stallman and Sussman, 1977; Davis, 1984; Gensereth, 1984; Bruynooghe and Pereira, 1984; Dechter, 1986; Bayardo and Schrag, 1996; Silva and Sakallah, 1996). In contrast to heuristic search, conflict-driven learning aims at acquiring control knowledge online, during the search, by identifying and analyzing conflicts of that very search, and thereby tailors search dynamically to the particular task at hand. In a nutshell, this works as follows. A search conflict is a partial variable assignment (e.g., a truth assignment to a subset of the Boolean variables) that has been considered by search, but all whose completion attempts resulted in violating some constraint. By generating an explanation of why it is not possible to complete that assignment to a solution, one learns knowledge (e.g., clauses) refuting that conflict. This explanation has the potential to generalize, refuting also yet unseen parts, and therewith reduces the work in the remainder of the search. Additionally, the explanation allows for non-chronological backtracking by spotting the specific decision in the search responsible for the conflict; the part between the conflict and the decision point need not be further explored.

Conflict-driven learning stands for *the* performance breakthrough in constraint-satisfaction-problem solv-

ing, and builds the core of most solvers in that area today. But despite this success, conflict-driven learning approaches for more general state-space search problems have received only little attention in research so far, and existing efforts pertain almost exclusively to constraint-satisfaction-problem equivalent search problems. Here, we start to close this gap.

We consider goal-reachability objectives in classical and probabilistic planning. The canonical form of “conflicts” in this setting are dead-end states, i.e., states starting from which the goal condition cannot be satisfied. Over the course of this thesis, we will develop methods to learn sound and generalizable dead-end knowledge from conflicts encountered during forward state-space search. Part II establishes the foundation, focusing on classical planning. We introduce suitable concepts capturing the currently *known* conflicts in search, and present different methods to represent and to use this information for learning to prune dead-end states during search. When acting under uncertainty, the presence of dead-end states may make it impossible to reach the goal with absolute certainty. The natural objective for probabilistic planning then is MaxProb, determining the maximal probability with which the goal condition can be satisfied. Unfortunately, algorithms for MaxProb probabilistic planning are severely underexplored, to the extent that there is scant evidence of what the empirical state of the art actually is. Part III addresses this deficiency before we then extend our conflict-driven learning techniques to probabilistic planning in Part IV. We show empirically that conflict-driven learning can lead to substantial performance improvements for (a) finding plans for solvable classical planning, (b) proving a classical planning task unsolvable, and (c) for goal-probability analysis in probabilistic planning, in situations where reasoning over dead ends is key.

## 1.2. Example Walk Through

We insert a simple example to illustrate to the reader the principle ideas behind the proposed methods. Adapted variants of this example will reappear throughout the thesis. We assume classical planning for simplicity. Consider the primitive space-mission-planning task depicted in Figure 1.1. The task in this example consists in collecting rock samples from two designated locations and carrying the results to the base station. There is a single rover at disposal to accomplish this mission, which can *move* freely between the locations, and can *collect* and *drop* samples at its current location. The movements consume energy. We want to find a sequence of control actions, a *plan*, so to complete all tasks before the rover runs out of energy.

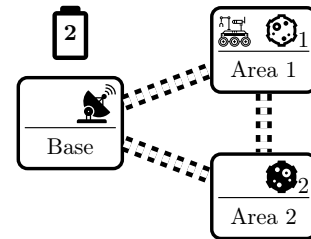


Figure 1.1.: Illustrative example.

The world states are represented via four state variables: the rover’s current position and remaining battery level, and variables keeping track of the status of the samples. For our demonstration purposes, it suffices to distinguish only whether or not a sample has been collected (i.e., abstracting away their precise positions). The possible battery level values are discretized up to a sufficient granularity. For this simple example, it suffices to distinguish between three different battery levels: 2 (“full”); 1 (“half-full”); and 0 (“empty”). We assume that every *move* depletes the battery by one unit, and that the samples can only be collected at the locations as shown in the figure. Figure 1.1 shows the initial situation: the rover is located at “area 1” (abbreviated as  $A_1$ ), has full battery, and no rock sample has been collected yet. The goal characterizes the states of the model at which we want to arrive: the rover should be at the “base” (abbreviated as  $B$ ) while having collected both samples. Further details regarding how the task is modeled exactly are not needed to understand the example.

Obviously, a solution to the task is given by the action sequence: *collect* the sample at  $A_1$ ; *move* to  $A_2$ ; *collect* the sample at  $A_2$ ; *move* to  $B$ . With merely  $3 \cdot 3 \cdot 2^2 = 36$  different states (3 battery values, 3 possible positions of the rover, and a Boolean flag for each sample), this sequence can of course be found by even the most simplest techniques. The situation however changes quickly as the number of different locations increases. Consider the extension of the example task to a chain of  $n$  areas, each of which contains a rock sample that needs to be collected, and, accordingly, a maximal battery capacity limit of  $n$  units. This variant has  $(n+1) \cdot (n+1) \cdot 2^n$  different states – the number of states of the model scales exponentially in the size of the description: the state explosion problem. Naively approaching the task is clearly infeasible as  $n$  is increasing.

Computer science’s Swiss knife when it comes to solving all sorts of computational problems is *search*, and indeed (explicit) *state-space search* has become one of predominant approaches to solve planning tasks. The state space relates states of the model by explicating the effects of actions. It essentially forms a directed graph over the states, with edges  $s \xrightarrow{a} t$  where, according to the model, the action application of  $a$  in state  $s$  results in the state  $t$ . The solution to the planning task is given by any path from the state representing the initial situation to a state satisfying the desired goal conditions, and can be found via any standard graph-search algorithm. Importantly, the state space does not need to be fully constructed for this process; the parts actually visited by search can be computed on-the-fly from the model description.

Now, notice that of all the exponentially many states in the example, there are only  $2n + 1$  states that do actually matter. Namely, exactly those states that are traversed by the solution. Every diversion of this path introduces the need of additional rover movements in order to accomplish the goal, which is not possible with the battery level initially provided. In other words, every diversion leads to a *dead-end state*, one from which the goal can no longer be reached. If we managed to identify the dead-end states, we would be able to constrain search to the solution path, thereby alleviating the state explosion problem. Unfortunately, deciding whether a state is a dead end is in general no easier than the complexity of planning itself. Hence, obtaining a perfect dead-end detector, recognizing all dead-end states, is in general intractable, and we have to expect to wastefully explore in search at least some dead-end parts. We next illustrate how to use these wasteful explorations for learning knowledge that (i) refutes completed parts of the search, (ii) leads to backjumping as in constraint satisfaction, and (iii) generalizes to other similar search branches.

Figure 1.2 depicts the search space of a depth-first search ran on the example task, assuming no prior knowledge about the dead-end states. The search starts at the initial state, marked ①. We *expand* this state, determining which actions of the model are applicable, and generating the corresponding edges and successor states. There are three possibilities to follow up on next: *move* to  $B$ , *move* to  $A_2$ , or *collect* the sample at  $A_1$ . Suppose we decided on the first option, proceeding to state ②. ② does not satisfy the goal, so we continue with its expansion. There are two symmetric options: *move* back to  $A_1$ , or *move* to  $A_2$ . Suppose we chose the former option, proceeding to ③. Back at  $A_1$ , the rover now has no energy left for further movements, but we can still *collect* the sample at  $A_1$ . In the resulting state, ④, the battery remains empty. The sample has been collected, so cannot be collected again. The only applicable action is dropping the collected sample, which however leads to a state that we have already seen. By inspecting the search space, we see that in fact all states reachable from ④ were already processed, while none of them satisfied the goal condition. At this moment, ④ has become a *known dead end*. In other words, search has encountered a *conflict*.

We initiate the learning procedure, *explaining* the conflict to obtain generalizable criteria recognizing similar dead-end states in the remainder of the search. The details are technically involved, so we omit them here. For this example, we use the well-known delete relaxation  $h^+$  to generate a *clause*  $\psi$ , a disjunction

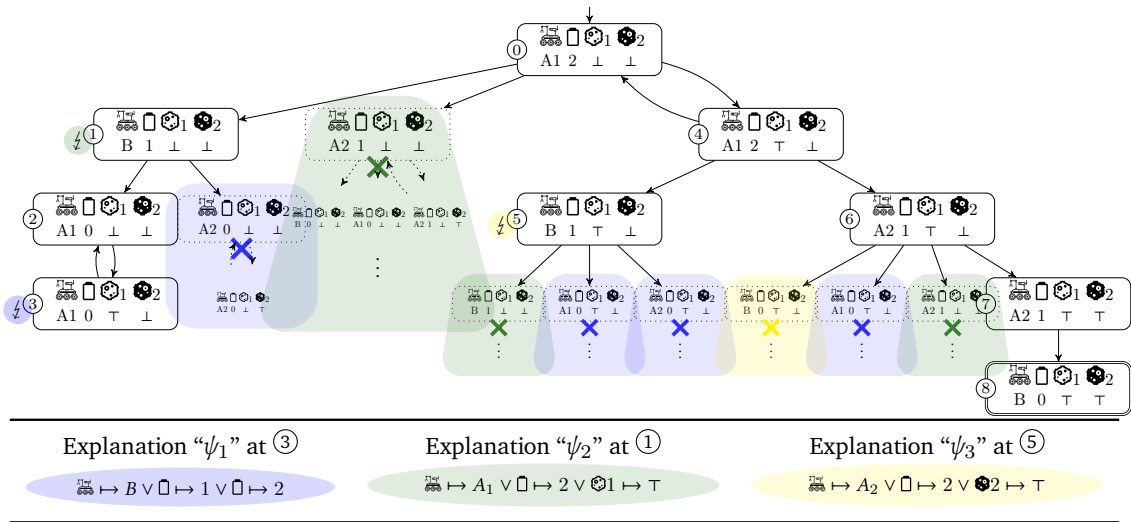


Figure 1.2.: Search space illustration of a depth-first search on the example task from Figure 1.1. Location names have been abbreviated. Each node is associated with a state in the model, as depicted. The edges correspond to action applications. The numbers attached to the nodes indicate the order in which the nodes are considered by search. The  $\text{⚡}$  labels indicate conflicts encountered during search. The colored regions represent dead-end states refuted by the generated explanations. The explanations in this example constitute clauses, disjunctions of variable-value pairs, that necessarily need to be satisfied in every non-dead-end state.

over variable-value pairs, such that, for all states  $s$ ,  $s \not\models \psi$  implies that  $s$  is a dead end. Intuitively, the delete relaxation overapproximates reachability by ignoring negative effects of actions. In our case, it basically ignores energy consumption. As the state ③ however has no energy left, it is recognized as dead end even in this relaxation: “ $h^+(\text{③}) = \infty$ ”. We initialize  $\psi_1 := \mathcal{F} \setminus \text{③}$  to be the disjunction over all variable-value pairs not true ③, and then iteratively remove elements from  $\psi_1$  until we arrived at a minimal *reason* for  $h^+(s) = \infty$ . In the specific case, we obtain  $\psi_1 = \text{Battery} \mapsto B \vee \square \mapsto 1 \vee \square \mapsto 2$ . To understand that clause, observe that every state in which the rover is not at the base requires at least one move action, which is possible only if the battery is not empty, even under delete-relaxed semantics.

Notice that the generated explanation (i) refutes ③ and ②, i.e., the part of the search space rooted at the identified conflict, and that (ii) search can backjump directly to the shallowest non-refuted ancestor state, ①. (The latter would happen here anyway, because ② has no more open successor states to explore. For an example with non-trivial backjumping, suppose there were  $n$  samples that have been collected, so can be discarded, but which can only be recollected at some location different from  $A_1$ . Exploring from ② a single trace of *drop* actions suffices to learn the same clause  $\psi_1$ , backjumping all the way back to ①.)

Back at ①, there is still the option to instead move to  $A_2$ . The resulting state does however violate  $\psi_1$ . In other words, (iii) the knowledge learned on the previous search branch generalized to the present one. We *prune* the expansion of the state, and continue with the remaining options in search. Until finding the desired plan, search learns two more clauses. These 3 explanations are enough to cover the 15 dead-end states that would have otherwise been considered by the search.

### 1.3. Contributions

Our contributions pertain to three subjects.

### Conflict-Driven Learning in Classical-Planning State-Space Search

We develop methods learning sound and generalizable knowledge from dead-end states encountered during state-space search in classical planning. This relies on two components: (1) *conflict identification*, i.e., the identification of dead-end states visited by search that are not yet refuted by the learned knowledge; and (2) *conflict explanation & learning*, i.e., refining the knowledge so to refute identified conflicts, in hopes that this generalizes to dead-end states not yet seen in search. The basic principles were already introduced in prior work (Steinmetz, 2015), upon which we build (see next).

Regarding (1), we introduce the concept of *known dead ends*, capturing the explicit knowledge that search creates through its state-space exploration. We use a method presented in prior work (Steinmetz, 2015) to identify the known dead ends in a generic open- & closed-list based search algorithm (which can be instantiated to well-known search algorithms like A\* or greedy best-first search). We show that this method ensures completeness, i.e., that before every state expansion all dead ends known at that moment were identified, and hence used for learning. Moreover, we design a depth-first search variant that is particularly effective in making dead ends quickly become known.

For learning (2), we consider three families of *unsolvability detectors*, i.e., sound yet incomplete dead-end characterizations: based on critical-path heuristics  $h^C$  (Haslum and Geffner, 2000; Haslum, 2012; Hoffmann and Fickert, 2015); based on the state-equation (Bonet, 2013) and potential heuristics (Pommerening et al., 2015); and based on dead-end traps (Lipovetzky et al., 2016).

Critical-path heuristics  $h^C$  approximate the reachability of a fact (variable-value pair) conjunction, e.g., the goal, by the reachability of the induced *atomic conjunctions* as defined by the parameter  $C$ . The choice of  $C$  allows to trade between computational cost and the accuracy of the approximations. We use this for conflict-driven learning by refining the conjunction set  $C$  to recognize previously unrecognized dead-end states, as identified by search. The refinement methods were already introduced in prior work (Steinmetz, 2015). To alleviate the  $h^C$  computation overhead as  $C$  keeps growing, we consider *critical-path NoGood learning*, i.e., learning formulas  $\varphi$  such that  $s \models \varphi$  implies that  $s$  is a dead end even under  $h^C$ 's reachability approximation. We show that, in theory, the dead ends recognized by  $h^C$  can be represented exactly through a (worst-case exponentially large) NoGood formula  $\Phi^{C*}$ . We devise two practical variants that dynamically generate such NoGoods during search, via a form of clause learning, and one that leverages the  $\Phi^{C*}$  construction. We provide a comprehensive evaluation of  $h^C$  dead-end learning with respect to the state of the art, for finding plans in solvable benchmarks with dead ends, for proving unsolvability in unsolvable benchmarks, and we evaluate the usefulness of the learned conjunction sets  $C$  as unsolvability certificates.

The state-equation heuristic approximates goal reachability by formulating a linear program (LP) that encodes certain relations between action-execution counts satisfied in every plan. Prior work has studied the use of partial variable merges to enhance the LP via additional constraints, improving the approximations. We show that, under reasonable assumptions, partial variable merges are strictly dominated by applying the state equation to the compilation  $\Pi^C$ , explicating a given fact conjunction set  $C$ . Potential heuristics approximate the cost-to-goal via a linear combination of real-valued fact conjunction weights. We show that admissible potential heuristics over arbitrary sets of fact conjunctions can be constructed via a variant of  $\Pi^C$ . Moreover, we provide *convergence results*, showing that the approximations of both types of heuristics can be made perfectly accurate via suitable sets  $C$ . We use this property for conflict-driven learning, designing a  $C$  refinement method to recognize previously unrecognized dead-end states. Our experimental evaluation on unsolvable benchmarks shows considerable performance improvements on several domains.

The learning variants mentioned so far suffer from the *transitivity property* provided by the respective un-

solvability detectors: if a state  $s$  is recognized as dead end, then all descendant states of  $s$  must necessarily be recognized as dead end as well. Transitivity can become a bottleneck when combining conflict-driven learning with other sources of dead-end information; to learn to recognize a dead end  $s$ , transitivity forces one to also learn to recognize the descendant states of  $s$ , possibly redundantly to the other technique. We introduce  $\mathcal{U}$ -traps as a remedy, generalizing the previous dead-end trap idea to allow for synergy with a complementary unsolvability detector  $\mathcal{U}$ . We furthermore provide a  $\mathcal{U}$ -trap refinement method to learn to recognize a dead-end state that is neither covered by  $\mathcal{U}$  nor by the current trap, and we provide a static offline analysis method, constructing the trap before search starts. We empirically evaluate the approach for proving unsolvability, showing benefits on several domains.

### State-Space Search Methods for Goal-Probability Analysis in Probabilistic Planning

Goal-probability analysis has been neglected in probabilistic-planning literature. We close this gap by (1) designing and exploring a large space of probabilistic state-space search methods, systematizing known algorithms and contributing several new algorithm variants; (2) introducing relevant special cases and weaker goal-probability objectives, alongside suitable algorithm adaptations exploiting these structures; and (3) providing an extensive empirical analysis that clarifies the state of the art, characterizes the behavior of a wide range of MDP algorithms, and demonstrates significant benefits of our new algorithm variants.

To detail on (1), we provide a comprehensive survey of the foundation of state-space-based algorithms addressing MaxProb. We revisit MDP heuristic state-space search, which has so far been used almost exclusively for expected-cost minimization objectives. We make contributions along the following algorithm dimensions:

- (a) *Search algorithm.* We design variants of AO\* (Nilsson, 1971) and LRTDP (Bonet and Geffner, 2003b), and introduce the family of depth-first MDP heuristic search algorithms systematizing known algorithms like LILAO\* (Hansen and Zilberstein, 2001) and HDP (Bonet and Geffner, 2003a). We develop early-termination criteria addressing the weaker objectives, and prove the resulting algorithms to be correct. To solve goal-probability objectives, MDP heuristic search must in general be iterated multiple times, interleaved with additional processing steps, a procedure known as FRET (Kolobov et al., 2012b). We introduce a new FRET variant, show its correctness, and demonstrate its benefits and downsides with respect to the original FRET design. We observe that this new FRET variant tightly integrates into MDP heuristic search. Based on this observation, we develop novel heuristic search algorithms natively supporting goal-probability analysis, without the need of any FRET outer loop. Finally, we design a comprehensive arsenal of simple strategies, biasing tie-breaking in action and state selection in manners targeted at fostering early termination.
- (b) *State-space reduction.* We design sound probabilistic-state-space reduction methods, via bisimulation relative to the all-outcomes determinization (Bonet and Geffner, 2005; Jimenez et al., 2006), and via dead-end pruning. Regarding the latter, we employ classical-planning heuristic functions for dead-end detection in probabilistic planning, again utilizing the all-outcomes determinization, as previously done by Teichteil-Königsbuch et al. (2011). This is especially promising in *limited-budget planning*, where we can prune a state  $s$  if an admissible classical-planning estimate exceeds the remaining budget in  $s$ . On the side, we discover that the landmarks compilation as per Domshlak and Mirkis (2015), employed for dead-end pruning in their oversubscription planning setting, is actually, on its own, equivalent to pruning against the remaining budget with a standard admissible landmark heuristic. This is relevant to our work because, otherwise, that compilation would be a canonical candidate also for dead-end pruning in our setting.

- (c) *Goal-probability heuristic function*. We develop two families of goal-probability estimators, starting from Trevizan et al.’s (2017b) linear programming (LP) based expected-cost estimators.

First, we introduce *goal-probability projection occupation-measure heuristics* that, besides the adaption to the goal-probability objective, generalize Trevizan et al.’s (2017b) expected-cost variant by the support of syntactic projections of the planning task onto arbitrary state-variable sets. We show that the resulting heuristic functions constitute monotone upper bounds on the maximal goal-probability function, as needed for the correctness of the MDP heuristic search algorithms.

Second, we design the *probabilistic operator-counting heuristic* framework, generalizing its classical-planning counterpart (Pommerening et al., 2014). We provide a suitable definition of *probabilistic operator-counting constraints*, a characterization of the constraints of the heuristic’s LP sufficient to obtain monotone upper-bounding goal-probability estimates. We instantiate the general framework by showing how to generate such constraints based on: goal-probability projection occupation-measure heuristics; a generalization of the state equation (Bonet, 2013) to the probabilistic setting; and via action landmarks (e.g., Karpas and Domshlak, 2009).

We implemented all these techniques within FAST DOWNWARD (FD) (Helmert, 2006), thus contributing, as a side effect of our work, an ideal implementation basis for exploiting classical-planning heuristic search techniques in MDP heuristic search. To explore the behavior of our algorithm design space, we created large benchmark suite comprising domains from the planning competitions, resource-constrained planning, and network penetration testing.

### Conflict-Driven Learning in Probabilistic-Planning State-Space Search

Lastly, we lift the classical-planning techniques to the probabilistic setting – learning to recognize dead-end states during MDP state-space search. Thanks to the all-outcomes determinization (Bonet and Geffner, 2005; Jimenez et al., 2006), the conflict explanation & learning methods from Part II can be plugged in directly. We provide a suitable adaption of the *known dead end* concept. We show how to identify the known dead ends in the different algorithms inspected in Part III, and prove a completeness property similar to the classical case. Our empirical study on MaxProb analysis demonstrates that the conflict-driven learning approach can be an effective means for goal-probability analysis.

## 1.4. Outline

The structure of the thesis roughly follows the outline of our contributions.

Part I starts with introducing some general background. Chapter 2 concerns classical planning. It sets up the basic notions used throughout the thesis, including the definition of classical planning tasks, the definition of state space in this context, and it briefly discusses heuristic state-space search. Chapter 3 provides a definition of the state space’s strongly connected components (SCCs), and revisits Tarjan’s (1972) algorithm for computing all maximal SCCs. Chapter 4 provides an overview of linear programming. Tarjan’s algorithm and linear programming are central to many parts of this thesis.

In Part II, we introduce our conflict-driven learning techniques for classical planning. Chapter 5 provides the basics, explaining how to exactly identify dead-end states during search. Chapters 6, 7, and 8 then provide the means for learning from the identified dead ends, via critical-path heuristics  $h^C$ , the LP-based

state-equation and potential heuristics, respectively via dead-end traps, in this order. Chapter 9 briefly summarizes and discusses the material presented in this part.

Part III is concerned with heuristic search techniques for goal-probability analysis in MDP probabilistic planning. Chapter 10 introduces formally probabilistic planning tasks, MDPs, and the different goal-probability objectives. Chapter 11 revisits two fundamental algorithms for solving MDPs optimally: via linear programming, and value iteration. Chapter 12 adapts MDP heuristic search algorithms towards solving the goal-probability objectives. Chapter 13 discusses our strategies to break ties left by the design of those algorithms. Goal-probability heuristic functions are the subject of Chapter 14, the MDP state-space reduction methods are discussed in Chapter 15. Chapter 16 shows our empirical study, and we conclude this part by a brief summary and discussion in Chapter 17.

The last content part, Part IV, applies the conflict-driven learning schema to MDP state-space search.

We conclude the thesis in Part V.

## 1.5. Publications

We provide a brief overview and summary of the publications that appeared during the course of work on this thesis, and point out the thesis author’s contributions. In general, the thesis author helped in the writing of all listed publications (if not stated otherwise). We hence focus on contributions pertaining to the development of concepts, algorithms, implementation, or experiments. We group the publications by subject, treating separately the core publications which underlay this thesis. We also discuss in more depth the relation of this thesis to our preceding work.

### 1.5.1. Core Publications and Relation to Prior Work

Most parts of this thesis have been published in proceedings of AI and planning conferences and journals. The principal work underlying all these publications, i.e., development of the theory, proofs, algorithms, implementation, and experiments, is due to the author of this thesis.

#### Conflict-Driven Learning in Classical-Planning State-Space Search

The following two publications establish general framework of conflict-driven learning in classical-planning state-space search (presented in Chapter 5), and instantiate the framework via critical-path heuristic  $h^C$  refinements (Chapter 6). The journal version expands on the conference version by introducing a depth-first search variant designed for conflict-driven learning, a second  $h^C$  refinement algorithm, a clause learning method to generate  $h^C$ -NoGoods, and a comprehensive experimental evaluation.

- Marcel Steinmetz and Jörg Hoffmann. “Towards Clause-Learning State Space Search: Learning to Recognize Dead-Ends.” In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI 2016*. AAAI Press, pp. 760–768
- Marcel Steinmetz and Jörg Hoffmann. “State space search nogood learning: Online refinement of critical-path dead-end detectors in planning.” In: *Artificial Intelligence* 245, pp. 1–37

**Relation to our prior work** These two publications were partly based on our prior work (Steinmetz, 2015), preceding this thesis, so we insert at this point the discussion of the relation of this thesis with respect to that prior work. We extended that work by formalizing the idea of what it means for a dead-end state to become *known* during search; by a thorough discussion of properties of different search algorithms with respect to the conflict-learning context; by a depth-first search variant geared at conflict identification; by minor improvements to the  $h^C$  neighbors refinement algorithm, specifically, a minimization procedure to find set-inclusion-minimal conjunctions to include into  $C$ ; by the clause learning method; and by an extensive experimental evaluation. We reused from that work the conflict identification method for the general open- & closed-list based search algorithm; some of the arguments proving its correctness; and the two  $h^C$  refinement algorithms, alongside their correctness proofs. The presentation of the material, the implementation, and the experiments were redone completely.

In the next publication, we further explored the idea of critical-path NoGood learning (Section 6.3). We showed that one can construct a NoGood formula  $\Phi^{C*}$  that captures the  $h^C$ -recognized dead ends exactly, and we leveraged that construction for our second critical-path NoGood learning variant.

- Marcel Steinmetz and Jörg Hoffmann. “Critical-Path Dead-End Detection versus NoGoods: Offline Equivalence and Online Learning.” In: *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017*. AAAI Press, pp. 283–287

The following publication analyzes the convergence behavior of LP-based heuristics, when combined with the  $\Pi^C$  compilation (Chapter 7). We adapted the  $\Pi^C$  compilation to the FDR planning-task framework underlying those heuristics, proved that there always exists a suitable conjunction set  $C$  so to render those heuristics perfect, and exploited this property through a conflict-based conjunction-set refinement method.

- Marcel Steinmetz and Jörg Hoffmann. “LP Heuristics over Conjunctions: Compilation, Convergence, Nogood Learning.” In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018*, pp. 4837–4843

Our exposition of the potential heuristics (Section 7.3) follows the technical report that accompanied this publication.

The following publication introduces the concept of  $\mathcal{U}$ -traps (Chapter 8), presents a conflict-based refinement method for dead-end learning, as well as an alternative static offline construction method.

- Marcel Steinmetz and Jörg Hoffmann. “Search and Learn: On Dead-End Detectors, the Traps they Set, and Trap Learning.” In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, pp. 4398–4404

As typical in planning, there is no single technique that works equally well in all situations. This is no different for the presented learning approaches. Going beyond the mentioned publications, we provide examples (in Sections 6.2.3, 7.5, and 8.5) demonstrating the benefits and downsides of each of the considered dead-end representation methods.

### State-Space Search Methods for Goal-Probability Analysis in Probabilistic Planning

The following two publications build the basis of Part III of this thesis. Specifically, we adapted several known MDP (heuristic) state-space search algorithms for goal-probability analysis and developed our FRET variant (jointly discussed in Chapter 12), designed tie-breaking strategies fostering early termination (Chapter 13), and introduced bisimulation respectively dead-end-pruning-based MDP state-space reduc-

tion methods via the all-outcomes determinization (Chapter 15). We systematically explored this large algorithm design space through an extensive experimental evaluation (Chapter 16). The journal version additionally introduced the depth-first heuristic search algorithm family, and extended the empirical study accordingly.

- Marcel Steinmetz, Jörg Hoffmann, and Olivier Buffet. “Revisiting Goal Probability Analysis in Probabilistic Planning.” In: *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016*. AAAI Press, pp. 299–307
- Marcel Steinmetz, Jörg Hoffmann, and Olivier Buffet. “Goal Probability Analysis in Probabilistic Planning: Exploring and Enhancing the State of the Art.” In: *Journal of Artificial Intelligence Research* 57, pp. 229–271

This thesis extends these works by providing additional details on the theory and correctness of the algorithms (in Chapters 11 and 12), by significantly expanding the description and correctness proofs of both FRET variants (Section 12.3.4), by introducing MDP heuristic search variants that support goal-probability analysis in the general (cyclic) case without FRET outer loop (Section 12.4), and by introducing goal-probability occupation-measure heuristics (Chapter 14 and Section 16.4).

### 1.5.2. Further Publications

The following publications are not covered by this thesis.

Beyond the listed core publications, we worked on diverse other aspects of planning, spanning over different formalisms, algorithms, and applications. While some of the works directly build on or make use of the presented results, others more tangentially touch the topics of this thesis.

#### Classical & Probabilistic Planning Algorithms

**Partial Delete Relaxation** The delete relaxation is a popular technique to derive classical-planning heuristic functions, but in some cases can harshly oversimplify a planning task’s semantics. Partial delete relaxation approaches allow to interpolate between the delete-relaxed and the actual semantics at the expense of an increased computational overhead.

- Maximilian Fickert, Jörg Hoffmann, and Marcel Steinmetz. “Combining the Delete Relaxation with Critical-Path Heuristics: A Direct Characterization.” In: *Journal of Artificial Intelligence Research* 56, pp. 269–327

One approach to make the delete relaxation more informative is by explicitly taking into account selected fact (variable-value pair) conjunctions. Previous works have done so via *compilations*, augmenting the planning task description by explicating the conjunctions. In this work, we married the delete relaxation with the critical-path heuristic  $h^C$ , getting rid of that intermediate compilation step.

The contributions of the thesis author pertain to the evaluation and presentation of the experimental results. The theoretical analysis and the implementation are due to the co-authors of this publication, and were partly already published in prior work.

- Patrick Speicher, Marcel Steinmetz, Daniel Gnad, Jörg Hoffmann, and Alfonso Gerevini. “Beyond Red-Black Planning: Limited-Memory State Variables.” In: *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017*, pp. 269–273

Red-black planning partitions the planning task’s state-variable set into *red* and *black* variables. The red variables are interpreted in delete-relaxed semantics (memorizing all variable values ever achieved), while the black variables follow the standard semantics (are assigned a single value at all time). It is well known that for certain variable-set partitionings planning becomes tractable, i.e., can be solved in polynomial time, a property that has been exploited for heuristic construction. In this work, we introduced a third category “gray variables” with a limited memory. These variables allow enlarging the tractable fragment, giving rise to more informative heuristic functions.

This publication was the result of Patrick Speicher’s Bachelor’s thesis, co-supervised by Daniel Gnad and the thesis author. The general approach and ideas on the proofs and implementation were developed in joint discussions.

- Daniel Gnad, Marcel Steinmetz, Mathäus Jany, Jörg Hoffmann, Ivan Serina, and Alfonso Gerevini. “Partial Delete Relaxation, Unchained: On Intractable Red-Black Planning and Its Applications.” In: *Proceedings of the Ninth Annual Symposium on Combinatorial Search, SOCS 2016*. AAAI Press, pp. 45–53

In this work, we considered applications of red-black planning beyond the computation of heuristic functions. We introduced *red-black search*, a method solving arbitrary red-black planning tasks. We used this procedure as basis for computing plans for the original, non-relaxed, planning task by using the computed relaxed plans as starting point for an off-the-shelf plan-repair algorithm; and for proving the original planning task unsolvable.

The original idea and implementation of red-black search is due to Daniel Gnad. The thesis author contributed to the design, presentation, and correctness proof of red-black search, as well as to the experimental evaluation. The results on relaxed-plan repair are partly due to Mathäus Jany’s Bachelor’s thesis, co-supervised by Daniel Gnad and the thesis author.

**Partial-Order Reduction in Planning with Resources** Stubborn sets are a well-known technique for pruning the state space during forward state-space search by exploiting the permutability of actions. In the following publication, we studied stubborn sets in the context of planning with resources.

- Anna Wilhelm, Marcel Steinmetz, and Jörg Hoffmann. “On Stubborn Sets and Planning with Resources.” In: *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018*, pp. 288–297

We introduced stubborn set variants, exploiting that, thanks to the commutativity of addition and subtraction, the actions’ effects on the resources do not affect permutability on a concrete state-space path. Moreover, we developed a method for automatically classifying state variables in a planning task description as resource variables, which allows us to apply our pruning techniques even in absence of explicit knowledge about the resources.

This publication was based on Anna Wilhelm’s Bachelor’s thesis, supervised by the author of this thesis. Ideas on the stubborn set definitions and proofs were developed in joint discussions. The

resource-variable detection method and implementation is due to the author of this thesis. The experimental evaluation was a collaborative result.

**Hyperabstraction Heuristics** Abstraction and critical-path heuristics belong to the most important families of admissible heuristics in classical planning. In the following publication, we married their underlying ideas, forming the class of hyperabstraction heuristics.

- Marcel Steinmetz and Álvaro Torralba. “Bridging the Gap between Abstractions and Critical-Path Heuristics via Hypergraphs.” In: *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019*. AAAI Press, pp. 473–481

We analyzed various aspects of this new formalism. We showed that abstraction and critical-path heuristics can naturally be expressed as members of this family. On the side, we answered a long-standing question, showing that optimal cost partitioning (the best possible admissible additive combination) for critical-path heuristics, hence hyperabstraction heuristics, is computationally hard.

The publication was a joint effort of both co-authors. The formalisms, theoretical results, and implementation were principally developed by the thesis author, supported by Álvaro Torralba with his expertise on abstraction heuristics.

**Pattern Database Heuristics for Probabilistic Planning** Pattern databases (PDBs), i.e., collections of projections, are a well-known technique in classical planning to derive admissible (lower-bounding) cost-to-goal bounds. The following two publications lifted them to probabilistic planning.

- Thorsten Klößner, Jörg Hoffmann, Marcel Steinmetz, and Álvaro Torralba. “Pattern Databases for Goal-Probability Maximization in Probabilistic Planning.” In: *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021*. AAAI Press, pp. 201–209
- Thorsten Klößner, Marcel Steinmetz, Álvaro Torralba, and Jörg Hoffmann. “Pattern Selection Strategies for Pattern Databases in Probabilistic Planning.” In: *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022*. AAAI Press, pp. 184–192

The first publication introduced probabilistic PDB heuristics admissibly estimating (upper bounding) the maximal probability of reaching the goal. We showed that the goal-probability estimates of multiple projections can be combined admissibly by taking their minimum, and identified criteria under which this is guaranteed even by taking their product. The second publication investigated pattern generation algorithms, i.e., deciding what projections to actually use.

The first publication was the result of Thorsten Klößner’s Master’s thesis, co-supervised by Álvaro Torralba and the author of this thesis. The implementation of goal-probability projections, and of the goal-probability occupation-measure heuristics used for comparison, is due to the author of this thesis. Ideas on the criteria when taking the product remains admissible, on the pattern-generation methods, and the experimental evaluation were developed in joint discussions between the co-authors. All proofs and the implementation of the advanced features, being multiplicative goal-probability PDBs and the pattern-generation methods, are due to Thorsten Klößner, who also took care of conducting the experiments.

### Classical Planning with Description Logics

Classical planning typically makes *closed-world* and *closed-domain* assumptions: facts not present in a state are false, and the universe of objects is fixed and known a priori. Description logics (DL) formulate knowledge over facts true in the world over an infinite domain. The following two publications are placed in the context of *eKABs*, a known formalism for combining classical planning with DL ontologies, where states are interpreted under open-world and open-domain semantics.

- Stefan Borgwardt, Jörg Hoffmann, Alisa Kovtunova, and Marcel Steinmetz. “Making DL-Lite Planning Practical.” In: *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021*, pp. 641–645
- Stefan Borgwardt, Jörg Hoffmann, Alisa Kovtunova, Markus Krötzsch, Bernhard Nebel, and Marcel Steinmetz. “Expressivity of Planning with Horn Description Logic Ontologies.” In: *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022*. AAAI Press, pp. 5503–5511

Prior work has studied *eKABs* in the context of the description logic DL-lite, and showed that these can be solved via a translation into standard classical planning formalisms. In the first publication, we identified bottlenecks in this translation, and designed transformations to the translated *eKABs* tasks that make the tasks easier to handle for an off-the-shelf classical planner. The second publication dealt with the translation of *eKABs* using more powerful description logics into classical planning, by leveraging enhanced features of the planning task description language. This translation was found to be superior to the previous one, even equipped with our transformations, on the previous DL-lite *eKABs* benchmark set.

Both publications were the result of many joint discussions between the co-authors. The thesis author contributed in the first publication to the development of the transformation steps, to the benchmarks, and to the evaluation and presentation of the experimental results. The implementation is due to Alisa Kovtunova, who also conducted the experiments. In the second publication, the thesis author was responsible for the implementation, and contributed to the design of the translation and to the experimental evaluation. The theoretical exposition is primarily due to Stefan Borgwardt. Alisa Kovtunova conducted the experiments.

### Explainable Planning

Why has the automated planner chosen that particular plan  $\pi$ ? In the following publications, we attempted to provide answers to this question by generating explanations with respect to the space of all plans.

- Rebecca Eifler, Michael Cashmore, Jörg Hoffmann, Daniele Magazzeni, and Marcel Steinmetz. “A New Approach to Plan-Space Explanation: Analyzing Plan-Property Dependencies in Oversubscription Planning.” In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020*. AAAI Press, pp. 9818–9826
- Rebecca Eifler, Marcel Steinmetz, Álvaro Torralba, and Jörg Hoffmann. “Plan-Space Explanation via Plan-Property Dependencies: Faster Algorithms & More Powerful Properties.” In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*. ijcai.org, pp. 4091–4097

The basis are *plan-property dependencies*, i.e., relations between formulae  $\phi$  and  $\psi$  where all plans satisfying

$\phi$  necessarily violate  $\psi$ . Such plan-property dependencies allow to explain why the found plan does not provide a desired property  $\psi$ , using the relationships to the properties  $\phi$  it satisfies: “to satisfy  $\psi$  would have to forgo  $\phi$ ”. On an algorithmic level, finding plan-property dependencies boils down to computing all minimal unsolvable goal subsets (MUGS). In the first publication, we considered propositional and action-set (simple temporal) plan properties. We showed how to leverage conflict-driven learning for computing all MUGS, via multiple searches and transferring learned knowledge from one to the other search. In the second publication, we lifted the framework to the full range of (finite-path) temporal plan properties, and introduced additional algorithms for computing all MUGS based on BDD symbolic search and a conflict-driven learning approach requiring just a single search.

The contributions of the thesis author to both publications pertain to adaptations of the conflict-driven learning techniques and their implementation.

### Planning in Safety-Critical Applications

- Rasha Faqeh, Christof Fetzner, Holger Hermanns, Jörg Hoffmann, Michaela Klauck, Maximilian A. Köhl, Marcel Steinmetz, and Christoph Weidenbach. “Towards Dynamic Dependable Systems Through Evidence-Based Continuous Certification.” In: *Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Proceedings, Part II*, pp. 416–439

This publication has laid down a methodology for the continuous certification of dynamically evolving cyber-physical systems. System updates are first run in *shadow mode* in parallel to the current system, monitoring their behavior, and putting them into operation only when sufficient evidence of their correct behavior has been collected. Planning is used for generating test instructions to obtain this evidence.

The author of this thesis contributed to the countless discussions on the design of the general certification framework, and to the formalization of the planning component.

- Frederik Wiehr, Anke Hirsch, Lukas Schmitz, Nina Knieriemen, Antonio Krüger, Alisa Kovtunova, Stefan Borgwardt, Ernie Chang, Vera Demberg, Marcel Steinmetz, and Jörg Hoffmann. “Why Do I Have to Take Over Control? Evaluating Safe Handovers with Advance Notice and Explanations in HAD.” In: *ICMI '21: International Conference on Multimodal Interaction*. ACM, pp. 308–317

In highly automated driving, safety-critical situations may necessitate to handover control from the machine back to a human. Making this handover successful is however a challenging task, involving decisions on, e.g., when to initiate the handover, what information to provide to make the human aware of the situation, and how to present that information. In this publication, we approached this task from a model-based perspective. We predict safety-critical situations by simulating the autonomous system in a suitably precise world model. The result serves as basis for the explanation. A user study has explored various options how to use this information for designing the handover.

The author of this thesis contributed to the development of the autonomous car driving simulator used for the user study, has constructed the world model, and implemented the model-based simulation procedure.

- Marcel Steinmetz, Jörg Hoffmann, Alisa Kovtunova, and Stefan Borgwardt. “Classical Planning with Avoid Conditions.” In: *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-*

*Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022*, pp. 9944–9952

Given a formal characterization of safety, i.e., a formula  $\phi$  that must remain satisfied at all time. In this publication, we considered the question of how to effectively find a plan that avoids unsafe states, i.e., states that violate  $\phi$ . We designed approaches handling  $\phi$  indirectly, by compiling it into a new planning task. We furthermore developed state-space search approaches that handle  $\phi$  directly, based on an adaptation of Cartesian abstraction heuristics, and by a conflict-driven learning variant that learns to predict when  $\neg\phi$  becomes unavoidable on the way to the goal.

The principal work underlying this publication is due to the thesis author.

### Stackelberg Planning and Applications to Network- and Web-Security Analysis

Stackelberg planning extends classical planning to a two-player setting, where one player – the *leader* – aims at finding a minimal-cost action sequence that maximizes the plan cost of the other player – the *follower*. Such planning questions naturally appear in security-based applications, the follower taking the role of an attacker (e.g., a hacker attempting to gain access to a computer network), and the role of the leader (e.g., a network administrator) is to take precautions preventing successful attacks. The solutions to such Stackelberg planning tasks provide information on how to cost-effectively mitigate security threats.

The first of the following publications introduced the general Stackelberg planning framework, as well as a *leader-follower search* algorithm, with various enhancements, that computes all Pareto optimal solutions of such a task. Leader-follower search solves one classical planning task, “the follower sub-task”, for each move of the leader. The second publication develops further optimizations to this algorithm, by identifying and reusing common structures in these sub-tasks. The third publication investigates the use of Stackelberg planning for the network-security scenario as just sketched. The last publication in this series considers a related application, e-mail infrastructure analysis, analyzing the security (privacy, authenticity, and confidentiality) of e-mail traffic as it is routed through the internet.

- Patrick Speicher, Marcel Steinmetz, Michael Backes, Jörg Hoffmann, and Robert Künnemann. “Stackelberg Planning: Towards Effective Leader-Follower State Space Search.” In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18)*. AAAI Press, pp. 6286–6293
- Álvaro Torralba, Patrick Speicher, Robert Künnemann, Marcel Steinmetz, and Jörg Hoffmann. “Faster Stackelberg Planning via Symbolic Search and Information Sharing.” In: *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021*. AAAI Press, pp. 11998–12006
- Patrick Speicher, Marcel Steinmetz, Jörg Hoffmann, Michael Backes, and Robert Künnemann. “Towards automated network mitigation analysis.” In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019*, pp. 1971–1978
- Patrick Speicher, Marcel Steinmetz, Robert Künnemann, Milivoj Simeonovski, Giancarlo Pellegrino, Jörg Hoffmann, and Michael Backes. “Formally Reasoning about the Cost and Efficacy of Securing the Email Infrastructure.” In: *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018*, pp. 77–91

Regarding the thesis author's contributions, the Stackelberg planning formalism and leader-follower search with all its enhancements were jointly developed by Patrick Speicher and the author of this thesis. The implementation used in the first publication is due to Patrick Speicher, who also took care of conducting the experiments. The thesis author further contributed to the design of the benchmarks, and to the experimental evaluation. The thesis author's contributions to the second publication primarily pertain to the writing and to assisting with the benchmark generation. In the last two publications, the thesis author developed the implementations, tailored to the applications, and was responsible for the experiments. The Stackelberg planning task models in both publications were the collaborative result of all co-authors involved.

The following publication considers planning for network-security analysis in a non-Stackelberg-planning context. Planning has been used successfully for automating network penetration testing (pentesting), i.e., to generate possible attacks to a network. Yet, prior approaches considered either classical planning (highly abstract model, good scalability) or partially-observable MDPs (exact model, but impossible to solve for realistic networks). Here, we explored a middle ground – contingent planning – which considers partial observability in a qualitative form. We developed suitable pentesting contingent planning models, adapted solvers, and demonstrated the feasibility of our techniques based on experiments on real networks.

- Dorin Shmaryahu, Guy Shani, Jörg Hoffmann, and Marcel Steinmetz. “Simulated Penetration Testing as Contingent Planning.” In: *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018*, pp. 241–249

The thesis author only contributed to the benchmark generator used for the systematic evaluation of the proposed methods. The thesis author was neither involved in the writing, nor in the development of the models and methods.

### On the Connection of Planning & Model Checking

The relation between classical planning and model-checking safety properties in deterministic models has been explored extensively in the past. Works on the intersection between probabilistic planning and probabilistic model checking are however scarce. In the following series of publications, we started to close this gap. We developed compilations between two commonly used modeling languages, PPDDL (probabilistic planning) and Jani (probabilistic model checking). We used the compilations to generate an overarching benchmark collection encompassing standard benchmarks from both communities. Based on this collection, we ran an extensive experimental evaluation comparing goal-probability planning techniques, as will be presented in this thesis, and techniques from probabilistic model checking.

- Michaela Klauck, Marcel Steinmetz, Jörg Hoffmann, and Holger Hermanns. “Compiling Probabilistic Model Checking into Probabilistic Planning.” In: *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018*. AAAI Press, pp. 150–154
- Michaela Klauck, Marcel Steinmetz, Jörg Hoffmann, and Holger Hermanns. “Bridging the Gap Between Probabilistic Model Checking and Probabilistic Planning: Survey, Compilations, and Empirical Comparison.” In: *Journal of Artificial Intelligence Research* 68, pp. 247–310
- Ernst Moritz Hahn, Arnd Hartmanns, Christian Hensel, Michaela Klauck, Joachim Klein, Jan Kretínský, David Parker, Tim Quatmann, Enno Ruijters, and Marcel Steinmetz. “The 2019 Comparison of Tools for the Analysis of Quantitative Formal Models - (QComp 2019 Competition Report).” In: *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Proceedings, Part III*. Vol. 11429. Lecture Notes in Computer Science.

Springer, pp. 69–92

The first publication introduced the compilation from Jani to PPDDL, and presented a preliminary experimental comparison. The compilation and its implementation were developed by Michaela Klauck. The thesis author contributed to the experimental evaluation. The second publication introduced the reverse compilation, from PPDDL to Jani, and significantly extended the experiments. The thesis author developed and implemented that compilation, and took care of collecting all tools, and conducting the experiments. The benchmark collection, and the experimental evaluation were jointly developed by Michaela Klauck and the author of this thesis. The last publication in this sequence is the quantitative model-checking competition report, for which our benchmark collection served as basis, and in which we participated with our probabilistic FAST DOWNWARD variant.

In the following publication, we advocated the use of model-checking modeling languages – instead of planning modeling formalisms – to avoid modeling difficulties arising from certain model characteristics.

- Jörg Hoffmann, Holger Hermanns, Michaela Klauck, Marcel Steinmetz, Erez Karpas, and Daniele Magazzeni. “Let’s Learn Their Language? A Case for Planning with Automata-Network Languages from Model Checking.” In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020*. AAAI Press, pp. 13569–13575

The paper was inspired by modeling hassles that we ourselves ran into during attempts to apply planning tools on actual applications. Besides the writing, the contributions of the thesis author mostly pertain to discussions and the motivating examples.

### **Gaining Trust in Neural-Network Action Policies**

Motivated by the increased popularity of using machine learning for decision-making in safety-critical applications, like autonomous driving, in the following publications, we embarked on the quest for techniques that help to gain trust in these decisions. We adapted three approaches known from other contexts towards analyzing the behavior of machine-learned policies, i.e., functions that take the action decisions, with respect to a formal model and a given property.

In the first publication, we considered quantitative properties in stochastic environments, asking, e.g., for the probability that executing the policy results in an unsafe state. We leveraged tools from statistical model checking (SMC) to compute high precision approximations of the property value, and demonstrated the feasibility of the approach through a series of case studies on Racetrack (as known from AI literature). The second publication developed an interactive tool that allows to visually explore the SMC results for Racetrack at various granularity levels, ranging from a high-level overview down to an in-depth examination of the generated policy execution runs. The third publication developed a predicate-abstraction-based machinery for verifying qualitative safety properties on learned policies. Predicate abstraction is a well-established model-checking method. Yet, the integration of a policy raises non-trivial problems, like deciding whether an abstract transition is chosen by the considered policy, which we tackled via a whole toolbox of techniques. The last publication took up a method commonly used in software engineering – testing – to find “bugs”, i.e., states on which executing the policy does not comply with the desired property. We introduced a generic testing framework, formally analyzed the use of bounds on the property values for bug confirmation, and provided a implementation framework and experimental evaluation in classical planning.

- Timo P. Gros, Holger Hermanns, Jörg Hoffmann, Michaela Klauck, and Marcel Steinmetz. “Deep Statistical Model Checking.” In: *Formal Techniques for Distributed Objects, Components, and Systems - 40th IFIP WG 6.1 International Conference, FORTE 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020*. Vol. 12136. Lecture Notes in Computer Science. Springer, pp. 96–114
- Timo P. Gros, David Groß, Stefan Gumhold, Jörg Hoffmann, Michaela Klauck, and Marcel Steinmetz. “TraceVis: Towards Visualization for Deep Statistical Model Checking.” In: *Leveraging Applications of Formal Methods, Verification and Validation: Tools and Trends - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Proceedings, Part IV*. Vol. 12479. Lecture Notes in Computer Science. Springer, pp. 27–46
- Marcel Vinzent, Marcel Steinmetz, and Jörg Hoffmann. “Neural Network Action Policy Verification via Predicate Abstraction.” In: *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022*. AAAI Press, pp. 371–379
- Marcel Steinmetz, Daniel Fiser, Hasan Ferit Eniser, Patrick Ferber, Timo P. Gros, Philippe Heim, Daniel Höller, Xandra Schuler, Valentin Wüstholtz, Maria Christakis, and Jörg Hoffmann. “Debugging a Policy: Automatic Action-Policy Testing in AI Planning.” In: *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022*. AAAI Press, pp. 353–361

The first publication was the result of many paper refinements and many discussions involving all co-authors. The bulk of the work was carried out by Timo P. Gros and Michaela Klauck, who took care of learning the policies, integrating the policies in the statistical model checker Modes, and conducting the actual experiments. The author of this thesis contributed the formal Racetrack model, encoded in Jani, and a Racetrack instance generator. The thesis author’s contributions to the second publication pertain primarily to the discussions during which the visualization tool was developed. The contributions to the third publication pertain to the writing (of an earlier version of this publication), to the design of the benchmarks, and to regular discussions. In the last publication, the author of this thesis developed the theoretical analysis, the implementation (starting from the code developed by Phillippe Heim), and contributed to the experimental evaluation.



**Part I.**

## **Preliminaries**



## 2. Classical Planning

We lay down the mathematical foundation for the upcoming chapters. Classical planning will be a constant companion throughout the thesis. This chapter starts with introducing the two definitions of classical planning tasks most commonly used in literature. We introduce labeled transition systems, and explain how the behavior induced by a classical planning task is represented in terms of such a system. Finally, we briefly discuss heuristic search as an algorithmic approach to solve classical planning tasks.

### 2.1. Planning Task Formalisms

Classical planning deals with the question of finding a sequence of actions that, when applied to the initial world state, results in some goal world state. States and actions are specified symbolically by means of an abstract world model, assuming closed-world semantics, finite, discrete and deterministic dynamics, and full observability. In the following, we adapt two well-known frameworks to describe such models formally. Both formalisms are equally expressive in theory, yet certain classical planning techniques are more naturally expressed in one than the other. We briefly discuss common algorithmic problems arising from these formalisms at the end of the section.

#### 2.1.1. STRIPS Planning

The STRIPS formalism (Fikes and Nilsson, 1971) describes the world through the lenses of *propositions*, also called *facts*. A world state in this formalism is represented by the list of facts known to be true in that state. Every fact that is not listed is assumed to be false. Actions modify states by making true “*adding*” new facts, respectively making false “*deleting*” facts, which were true previously. The goal world states are characterized through facts whose simultaneous satisfaction is sought. The complete definition is:

**Definition 2.1** (STRIPS Planning Task). *A STRIPS planning task is a tuple*

$$\Pi = \langle \mathcal{F}, \mathcal{A}, I, \mathcal{G} \rangle$$

*with components*

- $\mathcal{F}$  is a finite set of *facts*.
- $\mathcal{A}$  is a finite set of *actions*. Each action  $a \in \mathcal{A}$  has a **precondition**  $\text{pre}(a) \subseteq \mathcal{F}$ , **add effect**  $\text{add}(a) \subseteq \mathcal{F}$ , **delete effect**  $\text{del}(a) \subseteq \mathcal{F}$ , and non-negative **cost**  $\text{c}(a) \in \mathbb{R}_0^+$ .
- $I \subseteq \mathcal{F}$  is the *initial state*.
- $\mathcal{G} \subseteq \mathcal{F}$  is the *goal*.

The **states**  $\mathcal{S}^\Pi = 2^\mathcal{F}$  of  $\Pi$  are all sets of facts ( $2^X$  denoting the powerset of a set  $X$ ). An action  $a$  is called **applicable** in a state  $s$  if  $\text{pre}(a) \subseteq s$ . The result of this application is the state  $s \llbracket a \rrbracket = (s \setminus \text{del}(a)) \cup \text{add}(a)$ .

We refer to the individual components of  $\Pi$  by  $\mathcal{F}^\Pi$ ,  $\mathcal{A}^\Pi$ ,  $\mathcal{I}^\Pi$ , and  $\mathcal{G}^\Pi$ . We refer by  $\mathcal{A}(s) \subseteq \mathcal{A}$  to the set of all actions applicable in state  $s$ . The application of actions is extended to sequences of actions in an iterative manner. The empty action sequence  $\varepsilon$  is applicable in all states  $s$ , and  $s \llbracket \varepsilon \rrbracket = s$ . A non-empty action sequence  $\langle a_1, a_2, \dots, a_n \rangle$  is applicable in  $s$  if  $a_1 \in \mathcal{A}(s)$  and  $\langle a_2, \dots, a_n \rangle$  is applicable in  $s \llbracket a_1 \rrbracket$ . The resulting state is denoted  $s \llbracket \langle a_1, a_2, \dots, a_n \rangle \rrbracket$ . The cost of an action sequence is the sum of the cost of the individual actions:

$$c(\langle a_1, a_2, \dots, a_n \rangle) = \sum_{i=1}^n c(a_i)$$

Solutions of  $\Pi$  are action sequences of the following kind:

**Definition 2.2** (Plan). *Let  $s$  be a state of  $\Pi$ . An action sequence  $\pi$  is a **plan** for  $s$  if  $\pi$  is applicable in  $s$  and  $\mathcal{G} \subseteq s \llbracket \pi \rrbracket$ .  $\pi$  is an **optimal plan** for  $s$  if its cost is minimal among all plans for  $s$ . A (optimal) plan for  $\mathcal{I}$  is a (optimal) plan for  $\Pi$ .*

**Example 2.1.** *The following STRIPS planning task  $\Pi = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  models the rover example sketched in Section 1.2:*

- $\mathcal{F} = \{ \text{rov}(B), \text{rov}(A_1), \text{rov}(A_2) \}$  (the rover's position)  
 $\cup \{ \text{bat}(0), \text{bat}(1), \text{bat}(2) \}$  (the battery level)  
 $\cup \{ \text{samp}_i(x) \mid i \in \{1, 2\}, x \in \{B, A_1, A_2, R\} \}$  (location of the samples)
- $\mathcal{A} = \{ \text{move}(x, y, k) \mid x, y \in \{B, A_1, A_2\}, k \in \{1, 2\}, x \neq y \}$   
 $\cup \{ \text{collect}(\text{samp}_i, x) \mid i \in \{1, 2\}, x \in \{B, A_1, A_2\} \}$   
 $\cup \{ \text{drop}(\text{samp}_i, x) \mid i \in \{1, 2\}, x \in \{B, A_1, A_2\} \}$

*with preconditions, add and delete effects as shown in the table below; all actions have cost 1,*

	pre	add	del
$\text{move}(x, y, k)$	$\{ \text{rov}(x), \text{bat}(k) \}$	$\{ \text{rov}(y), \text{bat}(k-1) \}$	$\{ \text{rov}(x), \text{bat}(k) \}$
$\text{collect}(\text{samp}_i, x)$	$\{ \text{rov}(x), \text{samp}_i(x) \}$	$\{ \text{samp}_i(R) \}$	$\{ \text{samp}_i(x) \}$
$\text{drop}(\text{samp}_i, x)$	$\{ \text{rov}(x), \text{samp}_i(R) \}$	$\{ \text{samp}_i(x) \}$	$\{ \text{samp}_i(R) \}$

- Initial state  $\mathcal{I} = \{ \text{rov}(A_1), \text{bat}(2), \text{samp}_1(A_1), \text{samp}_2(A_2) \}$
- Goal  $\mathcal{G} = \{ \text{samp}_1(B), \text{samp}_2(B) \}$

*A plan for  $\Pi$  is given by the action sequence  $\langle \text{collect}(\text{samp}_1, A_1), \text{move}(A_1, A_2, 2), \text{collect}(\text{samp}_2, A_2), \text{move}(A_2, B, 1), \text{drop}(\text{samp}_1, B), \text{drop}(\text{samp}_2, B) \rangle$ . This plan is an optimal plan.*

All actions in Example 2.1 have cost 1. In this case, plan cost and plan length become the same. In general, we will use the term *unit cost* to refer to planning tasks that exhibit this property.

A plan may not always exist. For instance, consider the state  $s = \{ \text{rov}(B), \text{bat}(0), \text{samp}_1(A_1), \text{samp}_2(A_2) \}$  in the task from Example 2.1. This state does not satisfy the goal, and there is no action applicable in  $s$ . But this means that there is no possibility to ever accomplish the goal.

**Definition 2.3** (Dead End). *A state  $s$  is called a **dead end** if there is no plan for  $s$ . If  $\mathcal{I}^\Pi$  is a dead end, we say that  $\Pi$  is **unsolvable**. Otherwise  $\Pi$  is called **solvable**.*

### 2.1.2. FDR Planning

The finite domain representation (FDR) formalism (Bäckström and Nebel, 1995; Helmert, 2009), also known under the term SAS<sup>+</sup>, lifts the propositional STRIPS syntax to multi-valued state variables:

**Definition 2.4** (FDR Planning Task). *An FDR planning task is a tuple*

$$\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$$

with components

- $\mathcal{V}$  is a finite set of **variables**. Each  $v \in \mathcal{V}$  is associated with a finite domain  $\mathcal{D}_v$ . A **variable assignment** is a function  $P$  that maps a subset of variables  $\text{vars}(P) \subseteq \mathcal{V}$  to values of the respective domain. We denote by  $P[v] \in \mathcal{D}_v$  the value assigned to variable  $v \in \text{vars}(P)$  by  $P$ .  $P$  is called **complete** if  $\text{vars}(P) = \mathcal{V}$ .
- $\mathcal{A}$  is a finite set of **actions**. Each action  $a \in \mathcal{A}$  has a **precondition**  $\text{pre}(a)$  and an **effect**  $\text{eff}(a)$ , both are variable assignments, and a non-negative **cost**  $\mathbf{c}(a) \in \mathbb{R}_0^+$ .
- $\mathcal{I}$  is the **initial state**, a complete variable assignment.
- $\mathcal{G}$  is the **goal**, a variable assignment.

The **states**  $\mathcal{S}^\Pi$  of  $\Pi$  are the complete variable assignments. An action  $a$  is **applicable** in a state  $s$  if  $s[v] = \text{pre}(a)[v]$  holds for all  $v \in \text{vars}(\text{pre}(a))$ . The application results in the state  $s[a]$  with values

$$s[a][v] = \begin{cases} \text{eff}(a)[v] & \text{if } v \in \text{vars}(\text{eff}(a)) \\ s[v] & \text{otherwise} \end{cases}$$

The **facts**  $\mathcal{F}^\Pi$  of an FDR planning task  $\Pi$  are the variable-value pairs  $\langle v, d \rangle$ , also written  $v \mapsto d$ , for  $v \in \mathcal{V}$  and  $d \in \mathcal{D}_v$ . We treat variable assignments and sets of facts interchangeably. For two variable assignments  $P_1$  and  $P_2$ , we denote by  $P_1 \circ P_2$  the **update** of  $P_1$  by  $P_2$ , i.e., the variable assignment with  $\text{vars}(P_1 \circ P_2) = \text{vars}(P_1) \cup \text{vars}(P_2)$ ,  $(P_1 \circ P_2)[v] = P_2[v]$  for all  $v \in \text{vars}(P_2)$  and  $(P_1 \circ P_2)[v] = P_1[v]$  for  $v \in \text{vars}(P_1) \setminus \text{vars}(P_2)$ . We say that  $P_1$  and  $P_2$  are **consistent**, written  $P_1 \parallel P_2$ , if it holds for all variables  $v \in \text{vars}(P_1) \cap \text{vars}(P_2)$  that  $P_1[v] = P_2[v]$ . Otherwise,  $P_1$  and  $P_2$  are **inconsistent**, and we write  $P_1 \nparallel P_2$ .

We share symbols and notations across STRIPS and FDR planning tasks. We frequently use *planning task*, or simply *task*, to refer to either STRIPS or FDR planning tasks. If important for the discussion, it will be clear from the context to which formalism we refer to exactly. The application of action sequences, plans, and dead ends for FDR planning tasks can be defined just as for STRIPS (cf. Definitions 2.2 and 2.3). We omit restating the definitions here.

**Example 2.2.** Example 2.1 can be translated into the FDR planning task  $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  where:

- Variables  $\mathcal{V} = \{ \text{rov}, \text{bat}, \text{samp}_1, \text{samp}_2 \}$  with domains

$$\mathcal{D}_{\text{rov}} = \{ B, A_1, A_2 \} \quad \mathcal{D}_{\text{bat}} = \{ 0, 1, 2 \} \quad \mathcal{D}_{\text{samp}_1} = \mathcal{D}_{\text{samp}_2} = \{ B, A_1, A_2, R \}$$

- The actions  $\mathcal{A}$  are defined as follows:

	<i>pre</i>	<i>eff</i>
$\text{move}(x, y, k)$	$\{ \text{rov} \mapsto x, \text{bat} \mapsto k \}$	$\{ \text{rov} \mapsto y, \text{bat} \mapsto k - 1 \}$
$\text{collect}(\text{samp}_i, x)$	$\{ \text{rov} \mapsto x, \text{samp}_i \mapsto x \}$	$\{ \text{samp}_i \mapsto R \}$
$\text{drop}(\text{samp}_i, x)$	$\{ \text{rov} \mapsto x, \text{samp}_i \mapsto R \}$	$\{ \text{samp}_i \mapsto x \}$

- Initial state  $\mathcal{I} = \{rov \mapsto A_1, bat \mapsto 2, samp_1 \mapsto A_1, samp_2 \mapsto A_2\}$
- Goal  $\mathcal{G} = \{samp_1 \mapsto B, samp_2 \mapsto B\}$

The plans of  $\Pi$  are identical to the plans of the STRIPS task from Example 2.1.

### 2.1.3. Planning Objectives & Complexity

The two algorithmic problems most frequently considered in the classical planning literature are those of *optimal* and *satisficing* planning. Satisficing planning deals with the problem of finding some plan, ideally of good quality, while optimal planning imposes the additional requirement that the found plans must be optimal. With the first international planning competition on proving *unsolvability* in 2016, a third category has recently started to gain momentum: proving that no plan exists. The three planning variants give rise to the following decision theoretic questions:

**Plan Existence** Given a planning task  $\Pi$ , does there exist any plan for  $\Pi$ ?

**Cost-Bounded Plan Existence** Given a planning task  $\Pi$  and a cost bound  $b \in \mathbb{R}_0^+$ , does there exist a plan for  $\Pi$  whose cost does not exceed  $b$ ?

As has been shown by Bylander (1994) and by Bäckström and Nebel (1995), both questions fall into the class of PSPACE-complete decision problems. This result holds, regardless of whether the planning task is described in STRIPS or in FDR notation.

## 2.2. Transition Systems & State Spaces

Planning tasks specify *implicitly* how states change as function of the chosen action. Labeled transition systems make these dynamics *explicit*:

**Definition 2.5** (Labeled Transition System). A *labeled transition system (LTS)* is a tuple

$$\Theta = \langle \mathcal{S}, \mathcal{L}, \mathcal{T}, s_I, \mathcal{S}_*, \mathfrak{c} \rangle$$

with components

- $\mathcal{S}$  is a finite set of *states*.
- $\mathcal{L}$  is a finite set of *transition labels*.
- $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$  are the *transitions*.
- $s_I$  is the *initial state*.
- $\mathcal{S}_* \subseteq \mathcal{S}$  are the *goal states*.
- $\mathfrak{c}: \mathcal{L} \rightarrow \mathbb{R}_0^+$  is the *label cost function*.

We additionally use the following notation. A state  $s' \in \mathcal{S}$  is called a **successor** of a state  $s \in \mathcal{S}$  in  $\Theta$  if there is a transition  $\langle s, l, s' \rangle \in \mathcal{T}$ . In this case,  $s$  is also called a **predecessor** or **parent** of  $s'$ . We denote the set of all successors of a state  $s$  in  $\Theta$  by  $\text{Succ}[\Theta](s)$ . The set of all predecessors is denoted  $\text{Pred}[\Theta](s)$ . By  $\text{Succ}^+[\Theta]$  and  $\text{Pred}^+[\Theta]$  we denote the transitive closure of  $\text{Succ}[\Theta]$  and  $\text{Pred}[\Theta]$ , i.e.,  $\text{Succ}^+[\Theta](s) \subseteq \mathcal{S}$  is the smallest set that satisfies  $\text{Succ}[\Theta](s) \subseteq \text{Succ}^+[\Theta](s)$  and  $s'' \in \text{Succ}^+[\Theta](s)$  if

there exists  $s' \in \text{Succ}^+[\Theta](s)$  and  $\langle s', l, s'' \rangle \in \mathcal{T}$ .  $\text{Pred}^+[\Theta]$  is defined similarly. A state  $s'$  is an **ancestor** of  $s$  in  $\Theta$  if  $s' \in \text{Pred}^+[\Theta](s)$ .  $s'$  is a **descendant** of  $s$  in  $\Theta$  if  $s' \in \text{Succ}^+[\Theta](s)$ .  $s'$  is **reachable** from  $s$  in  $\Theta$  if  $s' = s$  or  $s' \in \text{Succ}^+[\Theta](s)$ . The set of all states reachable from  $s$  is denoted as  $\mathcal{R}[\Theta](s) (= \text{Succ}^+[\Theta](s) \cup \{s\})$ . A state  $s$  is a dead end in  $\Theta$  if  $\mathcal{R}[\Theta](s) \cap \mathcal{S}_* = \emptyset$ . For sets of states  $S \subseteq \mathcal{S}$ , we define  $\text{Succ}[\Theta](S)$ ,  $\text{Pred}[\Theta](S)$ ,  $\text{Succ}^+[\Theta](S)$ ,  $\text{Pred}^+[\Theta](S)$ , and  $\mathcal{R}[\Theta](S)$  as the union over the respective sets of the individual states. We omit  $[\Theta]$  if  $\Theta$  is clear from the context.

An alternating sequence of states and labels  $\sigma = \langle s_1, l_1, s_2, l_2, \dots, s_n \rangle$  is called a **path** in  $\Theta$  if  $\langle s_i, l_i, s_{i+1} \rangle \in \mathcal{T}$  for all  $1 \leq i < n$ .  $\sigma$  is called a **cycle** if  $n > 1$  and  $s_n = s_1$ . The cost of  $\sigma$  is given by  $\mathfrak{c}(\sigma) = \sum_{i=1}^{n-1} \mathfrak{c}(l_i)$ . By  $\text{labels}(\sigma) = \langle l_1, l_2, \dots, l_{n-1} \rangle$  we denote the sequence of labels that appear along  $\sigma$ .

An LTS  $\Theta' = \langle \mathcal{S}', \mathcal{L}', \mathcal{T}', s_I', \mathcal{S}_*', \mathfrak{c}' \rangle$  is called **subgraph** of  $\Theta$  if (i)  $\mathcal{S}' \subseteq \mathcal{S}$ , and (ii)  $\mathcal{S}_*' \subseteq \mathcal{S}_*$ , and (iii)  $\mathcal{T}' \subseteq \mathcal{T}$ . Let  $\emptyset \subset S \subseteq \mathcal{S}$  be a non-empty subset of states. The subgraph of  $\Theta$  induced by  $S$  is the LTS  $\Theta|_S = \langle S, \mathcal{L}, \mathcal{T}|_S, s_I', \mathcal{S}_*|_S, \mathfrak{c} \rangle$ , where  $\mathcal{T}|_S = \{ \langle s, a, s' \rangle \in \mathcal{T} \mid s, s' \in S \}$ , and  $\mathcal{S}_*|_S = \mathcal{S}_* \cap S$ . Initial state may be defined arbitrarily. Let  $s \in \mathcal{S}$  be a state. The subgraph of  $\Theta$  reachable from  $s$  is the subgraph of  $\Theta$  induced by the states reachable from  $s$ , i.e.,  $\Theta|_{\mathcal{R}(s)}$ .

The LTS induced by a planning task  $\Pi$  explicates the result of action applications. The initial state of the LTS is that of  $\Pi$ , the goal states are all states of  $\Pi$  that satisfy the goal condition. The plans of  $\Pi$  correspond exactly to the paths from the initial state to the goal states in the LTS.

**Definition 2.6** (State Space). *Let  $\Pi$  be a planning task. The **state space** induced by  $\Pi$  is the LTS*

$$\Theta^\Pi = \langle \mathcal{S}^\Pi, \mathcal{L}^\Pi, \mathcal{T}^\Pi, s_I^\Pi, \mathcal{S}_*^\Pi, \mathfrak{c}^\Pi \rangle$$

where

- $\mathcal{S}^\Pi$  are the states of  $\Pi$ .
- The labels are the actions of  $\Pi$ ,  $\mathcal{L}^\Pi = \mathcal{A}^\Pi$ .
- The set of transitions is given by  $\mathcal{T}^\Pi = \{ \langle s, a, s \llbracket a \rrbracket \rangle \mid s \in \mathcal{S}^\Pi, a \in \mathcal{A}^\Pi(s) \}$ .
- Same initial state  $s_I^\Pi = \mathcal{I}^\Pi$ .
- The goal states are  $\mathcal{S}_*^\Pi = \{ s \in \mathcal{S}^\Pi \mid \mathcal{G}^\Pi \subseteq s \}$ .
- $\mathfrak{c}^\Pi$  is the action cost function of  $\Pi$ .

The number of states of a planning task  $\Pi$  is exponential in the size of  $\Pi$ , and hence the size of  $\Theta^\Pi$  is exponential in the size of  $\Pi$ . The discrepancy between compact *syntactic* ( $\Pi$ ) versus *semantic* ( $\Theta^\Pi$ ) representation is commonly known as the *state explosion problem*.

## 2.3. Heuristic Search

Given that the semantics of planning tasks can be conceived as a weighted directed graph, the state space, graph search hence becomes an obvious approach to compute plans. However, *blindly* exploring this graph via algorithms such as breadth-first search (Moore, 1959) or Dijkstra's search (Dijkstra, 1959) is generally not feasible due to the state explosion problem. Heuristic search exploits additional task-specific information to stringently guide the exploration towards goal states: *heuristic functions* (Pearl, 1984).

**Definition 2.7** (Heuristic). *Let  $\Pi$  be a planning task. A **heuristic function**, in short **heuristic**, for  $\Pi$  is a function  $h: \mathcal{S}^\Pi \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$ . The **perfect heuristic**  $h^*$  for  $\Pi$  assigns every state  $s \in \mathcal{S}^\Pi$  the cost of an optimal plan for  $s$ , or  $\infty$  if  $s$  is a dead end. A heuristic  $h$  is **admissible** if  $h(s) \leq h^*(s)$  for all  $s \in \mathcal{S}^\Pi$ .  $h$  is **consistent** if  $h(s) \leq h(s \llbracket a \rrbracket) + c^\Pi(a)$  for all  $s \in \mathcal{S}^\Pi$  and actions  $a \in \mathcal{A}^\Pi(s)$ .  $h$  is **goal-aware** if  $h(s) = 0$  for all  $s \in \mathcal{S}_*^\Pi$ .*

Heuristic functions provide estimations of the cost-to-go to reach the goal for a given state. Heuristic search algorithms use this information for guidance by prioritizing states with smaller heuristic value. The exact degree to which the heuristic values influence the exploration order differs between heuristic search algorithms. For example, A\* search (Hart et al., 1968) prioritizes states not just by that estimate, but also takes into account the cost of reaching states from the initial state. The additional bias makes the search more exhaustive, yet guarantees to find an optimal solution, if provided with an admissible heuristic. Greedy best-first search (GBFS) (Doran and Michie, 1966) orders states purely based on their heuristic values, which usually makes the search more effective, but loses the optimality property.

Note that some of the listed heuristic properties are correlated. In particular, a heuristic is admissible if it is consistent and goal-aware (Russell and Norvig, 2010). A heuristic is goal-aware if it is admissible. The heuristic value  $\infty$  serves the purpose of identifying dead ends. In principle, one can ignore dead ends during search completely, as it is not possible to reach the goal from them by definition. But as we have seen in Section 2.1.3, identifying all dead ends is in general not tractable. So, one falls back to the detection capabilities of the heuristic, pruning states  $s$  during search if  $h(s) = \infty$ .

### 3. Strongly Connected Components

In the previous chapter, we have seen that graphs yield a very natural interpretation of (classical) planning tasks. Throughout the thesis, we will frequently run into the need of identifying *strongly connected components*. In this chapter, we provide the formal definition, and present a variant of Tarjan's (1972) famous algorithm to compute the strongly connected components of a graph. This algorithm builds the foundation of many algorithms that will follow.

**Definition 3.1** (Strongly Connected Component). *Let  $\text{Succ}$  be the successor relation underlying some graph structure over states  $\mathcal{S}$ , and let  $\text{Succ}^+$  be its transitive closure. A set  $S \subseteq \mathcal{S}$  is a **strongly connected component (SCC)** under  $\text{Succ}$  if for each distinct pair of states  $s, s' \in S$ , it holds that  $s \in \text{Succ}^+(s')$ . An SCC  $S$  is **maximal** if there is no SCC  $S' \subseteq \mathcal{S}$  under  $\text{Succ}$  such that  $S \subset S'$ .*

The maximal SCCs can be computed along a single depth-first exploration (Tarjan, 1972). Algorithm 3.1 sketches the main function. To compute all maximal SCCs, `DepthFirstTraversal` has to be called iteratively on every non-visited state until all states have been visited. The order of the calls is not important. SCCs are identified by tracking the visited, but not yet fully explored states in a stack like data structure. Due to the depth-first traversal, the order of the elements on the stack is consistent with the reachability relation. More precisely, every state  $s'$  above  $s$  in the stack was visited during the traversal of  $s$ , and hence each such  $s'$  is reachable from  $s$ . Finding the SCCs then boils down to maintaining reachability information also in the other direction, i.e., remembering which states below  $s$  on the stack are reachable from  $s$ . Say that some  $s'$  beneath  $s$  is reachable from  $s$ . Due to the stack order property, every state between  $s'$  and  $s$  is reachable from  $s'$ , and hence also reachable from  $s$  via transitivity of reachability. Since every state below  $s$  can reach  $s$ , the sequence of states on the stack from  $s'$  up to  $s$  therefore forms an SCC. A maximal SCC is found, when the traversal backtracks out of a state that cannot reach any state beneath itself on the stack.

**Theorem 3.1** (Tarjan, 1972). *Algorithm 3.1 computes the set of all maximal SCCs under  $\text{Succ}$ .*

---

**Algorithm 3.1:** Depth-first traversal, storing the maximal SCCs.

---

**Input:** Some graph structure over states  $\mathcal{S}$  with associated successor relation  $\text{Succ}$

**Output:**  $\text{sccs}$ : maximal SCCs under  $\text{Succ}$

---

```

1 globals
2    $\text{sccs} \leftarrow \emptyset$ ;                                /* computed SCCs */
3    $\text{visited} \leftarrow \emptyset$ ;                        /* states that were processed */
4    $\text{stack} \leftarrow \text{empty stack}$ ;                      /* current exploration stack */
5   /* lowest index of state in stack reachable from  $s$ : */
6    $\text{minReach}[s] \leftarrow \infty$  for all  $s \in \mathcal{S}$ ;
7
6 procedure DepthFirstTraversal( $s$ )
7    $\text{visited} \leftarrow \text{visited} \cup \{s\}$ ;
8    $\text{stackIdx} \leftarrow |\text{stack}|$ ;
9    $\text{minReach}[s] \leftarrow \text{stackIdx}$ ;                  /* initialize to own index */
10  push  $s$  onto  $\text{stack}$ ;
11  foreach  $s' \in \text{Succ}(s)$  do
12    if  $s' \notin \text{visited}$  then                          /* need to explore  $s'$  */
13      DepthFirstTraversal( $s'$ );
14      /* update index of reachable stack states */
15       $\text{minReach}[s] \leftarrow \min(\text{minReach}[s], \text{minReach}[s'])$ ;
16
15  if  $\text{minReach}[s] = \text{stackIdx}$  then
16    /* no state on stack below  $s$  is reachable */
17     $S \leftarrow \emptyset$ ;
18    while  $s \notin S$  do                                /* collect states down to  $s$  */
19       $s' \leftarrow \text{pop from stack}$ ;
20       $\text{minReach}[s'] \leftarrow \infty$ ;
21       $S \leftarrow S \cup \{s'\}$ ;
22   $\text{sccs} \leftarrow \text{sccs} \cup \{S\}$ ;

```

---

## 4. Linear Programming

The linear programming problem is a well-studied problem in AI and operations research literature. Due to its flexibility paired with very efficient solution approaches, linear programs have become a popular method for solving a wide range of combinatorial problems. And so, various techniques in the context of planning rely on linear programming as a sub-procedure. Linear programs will also appear in different parts of the thesis. This chapter introduces the relevant background and notions. For an in-depth elaboration of this topic, we refer the reader to classical textbooks (Chvátal, 1983; Dantzig and Thapa, 1997; Korte and Vygen, 2000).

The linear programming problem is the maximization or minimization of a linear objective function over a finite set of real-valued variables subject to a finite set of linear constraints. A linear programming problem instance is called *linear program*:

**Definition 4.1** (Linear Program). *A **linear program (LP)** with  $n$  variables and  $m$  constraints has the following components*

- **Constraint coefficients:**  $a_{ij} \in \mathbb{R}$  for  $1 \leq i \leq m$  and  $1 \leq j \leq n$ .
- **Bounds:**  $b_i \in \mathbb{R}$  for  $1 \leq i \leq m$ .
- **Objective coefficients:**  $c_j \in \mathbb{R}$  for  $1 \leq j \leq n$ .

The reals  $x_1, x_2, \dots, x_n \in \mathbb{R}$  are called a **feasible solution** to the LP if they satisfy the **constraint**

$$\sum_{j=1}^n a_{ij}x_j \leq b_i$$

for each  $1 \leq i \leq m$ . An **optimal solution** is a feasible solution with maximal **objective value**  $\sum_{j=1}^n c_j x_j$ . The LP is called **infeasible** if there is no feasible solution. The LP is called **unbounded** if it has some feasible solution, but no optimal one. The **objective value** of the LP is the objective value of its optimal solutions, or  $\infty$  if no such solution exists.

Linear programs are often written compactly using matrix notation. The constraint coefficients are formulated as an  $m$ -by- $n$  matrix  $A \in \mathbb{R}^{m \times n}$ , the bounds and objective coefficients as column vectors  $b \in \mathbb{R}^m$  and  $c \in \mathbb{R}^n$ . A feasible solution is then a column vector  $x \in \mathbb{R}^n$  such that  $Ax \leq b$ , where the inequality is applied per vector-component. An optimal solution maximizes  $c^T x$ , where  $c^T$  is the transpose of  $c$ . The product  $c^T x$  is commonly denoted as the *objective function*. The whole LP is also written  $\operatorname{argmax}_{x \in \mathbb{R}^n} \{ c^T x \mid Ax \leq b \}$ .

LPs as per Definition 4.1 are in *standard form*. Minimization objectives, and equality and  $\geq$  constraints are supported, in principle, as all these can be represented directly in standard form. The minimization of  $c^T x$  is equivalent to maximizing  $-c^T x$ . Equality constraints can be split into two inequality constraints.  $A \geq$  constraint with bound  $b_i$  and coefficients  $a_{ij}$  is equivalent to the  $\leq$  constraint with bound  $-b_i$  and coefficients  $-a_{ij}$ .

The *dual LP* offers an alternative view on LPs in standard form by exchanging the role of variables and constraints:

**Definition 4.2** (Dual LP). *Let*

$$\operatorname{argmax}_{x \in \mathbb{R}^n} \{ c^T x \mid Ax \leq b \}$$

*be an LP in standard form, called the **primal**. The **dual LP** is*

$$\operatorname{argmin}_{y \in \mathbb{R}^m} \{ b^T y \mid A^T y = c, y \geq 0 \}$$

In words, the dual LP contains one variable for every constraint in the primal, and one constraint for every variable in the primal. The  $j$ -th constraint in the dual aggregates the constraint coefficients referring to  $j$ -th variable in the primal:  $\sum_{i=0}^m a_{ij} y_i$ . The objective coefficients of the primal become bounds in the dual, and vice versa the objective coefficients in the dual are the bounds of the primal. The constraints in the dual enforce equality because the variables in the primal are unbounded. Conversely, the variables in the dual must be non-negative because of the upper-bounding constraints in the primal. The dual transformation is symmetric (Korte and Vygen, 2000), i.e., the dual of the dual gives again the primal (assuming that the LPs are brought back into standard form after each transformation).

The dual view offers many important properties that can be exploited to compute optimal solutions for the primal, or to prove that no such solution exists. In particular, one of the most fundamental results in the theory of linear programming is the *Duality Theorem*:

**Theorem 4.1** (The Duality Theorem (Gale et al., 1951)). *If the primal has an optimal solution, then the dual has an optimal solution with the same objective value.*

This theorem has several important consequences. As the notion of primal and dual is symmetric, the primal hence has an optimal solution if and only if the dual has one. And if this is the case, then both LPs have the same objective value. Moreover, it can be shown that the objective value of the primal is bounded by the objective values of the feasible solutions of the dual. In other words, if the primal is unbounded, then the dual cannot have any feasible solution. Symmetrically, if the dual is unbounded, then the primal must be infeasible.

Linear programs can be solved optimally in time polynomial in the size of the LP (Khachiyan, 1979). These algorithms however perform poorly in practice. Current LP solvers instead implement variants of the *Simplex Method* (Dantzig, 1951), which in pathological cases can actually require exponentially many steps in the size of the LP to find an optimal solution (Klee and Minty, 1972).

## Conflict-Driven Learning in Classical-Planning State-Space Search

In the realm of state-space search problems, research of conflict-driven learning methods is limited almost exclusively to *length-bounded reachability*, where reachability analysis degenerates to a constraint satisfaction problem, and via appropriate encodings (e.g., into SAT), standard learning techniques in principle apply unmodified. From the perspective of solving general reachability problems, length-bounded reachability is a limitation, as one needs to iterate over different length bounds until some termination criterion applies. In this part of the thesis, we show how to apply the principles of conflict-driven learning to state-space search without length bound, considering goal-reachability objectives in classical planning.

The canonical form of “conflicts” in this setting are dead-end states. We adapt common search algorithms to identify conflicts as search is unveiling new parts of the state space. We show how to *explain* the identified conflicts so to *learn* recognizing similar dead-end states that search may encounter in the future. The core component are *unsolvability detectors*, parameterized goal-reachability approximations  $\mathcal{U}[\rho]$  that (i) are sound, i.e., where  $\mathcal{U}[\rho]$  does not misclassify any state as being a dead end, and (ii) are complete in the limit, i.e., where, for some  $\rho^*$ ,  $\mathcal{U}[\rho^*]$  recognizes all dead-end states. We use  $\mathcal{U}[\rho]$  as a dead-end detection mechanism, disregarding recognized dead-end states during search. Property (i) ensures that this is safe, i.e., that we do not prune any solution. When search encounters a dead-end state  $\hat{s}$  that is not recognized by the current  $\mathcal{U}[\rho]$ , we explain the situation at  $\hat{s}$ , finding a refinement  $\hat{\rho}$  so that (1)  $\mathcal{U}[\hat{\rho}]$  recognizes  $\hat{s}$ , while (2)  $\mathcal{U}[\hat{\rho}]$  preserves the knowledge learned so far, i.e., still covers all dead-end states recognized by  $\mathcal{U}[\rho]$ . Property (ii) ensures that this is always possible. The refinement has the potential to generalize to unseen dead-end states, and if so, may allow to prune future search branches.

We instantiate this general framework by three different families of unsolvability detectors, adapting well-known techniques from classical planning that offer (i) out of the box: critical-path heuristics  $h^C$ ; LP-based heuristics centered around the state equation; and dead-end traps. We will be showing that they also satisfy (ii), designing alongside suitable conflict refinement methods.

We conduct comprehensive empirical evaluations demonstrating that the proposed methods can, under certain conditions, yield substantial search reductions for finding plans for solvable classical planning tasks, as well as for proving classical planning tasks unsolvable.



## 5. Conflict Identification in Forward State-Space Search

The foundation to our techniques is formed by *unsolvability detectors* (Hoffmann et al., 2014). Suppose  $\Pi$  is a classical planning task, and  $\Theta^\Pi = \langle \mathcal{S}, \mathcal{L}, \mathcal{T}, s_I, \mathcal{S}_*, \mathfrak{c} \rangle$  is its state space. Then formally

**Definition 5.1** (Unsolvability Detector). *A function  $\mathcal{U}: \mathcal{S} \rightarrow \{0, \infty\}$  is an **unsolvability detector** (also called **dead-end detector**) if  $\mathcal{U}(s) = \infty$  implies that  $s$  is a dead end. We say that a dead end  $s$  is **recognized** by  $\mathcal{U}$  if  $\mathcal{U}(s) = \infty$ , and it is **unrecognized** otherwise. We say that  $\mathcal{U}$  is **transitive** if, for every transition  $\langle s, a, t \rangle \in \mathcal{T}$ ,  $\mathcal{U}(s) = \infty$  implies  $\mathcal{U}(t) = \infty$ .  $\mathcal{U}$  is called **perfect** if it recognizes all dead ends. The perfect unsolvability detector is also denoted by  $\mathcal{U}^*$ .*

Assume we use such a  $\mathcal{U}$  to recognize and subsequently skip the exploration of dead-end states in search. Notice that, trivially, this kind of pruning can neither impede the completeness nor the optimality property of the search algorithm. It may however substantially reduce the number of states needed to be touched by search – in the extreme case, where  $\Pi$  is unsolvable, it may even remove the need for search altogether. Ideally, we would want to use the perfect unsolvability detector  $\mathcal{U}^*$ , as this would allow to ignore all dead-end states in search. However, the computation of  $\mathcal{U}^*$  is in general intractable, as it subsumes solving the input planning task in the first place. So, we need to expect that at least some dead-end states are, wastefully, considered. We will henceforth refer to such states by **conflicts**, and we attempt to learn from them, refining  $\mathcal{U}$  during search, so to recognize similar dead-end states in the future. But how to know whether a state considered by search is a conflict? And how to identify these states efficiently? We spell out these details in what follows.

In Section 5.1, we design a generic extension to search algorithms using open & closed lists, like  $A^*$ , greedy best-first search, etc., preserving their optimality and completeness guarantees. In Section 5.2, we design a dedicated depth-first search variant, which has turned out to be most useful in our experiments as it identifies conflicts much more quickly, facilitating the learning process. This chapter is based on (Steinmetz and Hoffmann, 2017c). The algorithm modifications and the arguments underlying our soundness and completeness results in Section 5.1 are due to prior work (Steinmetz, 2015). Section 5.1 extends that work by the formalization of *known dead ends*, and the discussion of additional properties beyond the fundamental correctness guarantees.

Throughout this chapter, we consider an arbitrary unsolvability detector  $\mathcal{U}$ , the only assumption being that there exists a refinement method which, given a conflict state  $s$ , refines  $\mathcal{U}$  to recognize  $s$ .

### 5.1. Generic Search Algorithm

Algorithm 5.1 shows the pseudo-code for our procedure. Consider first only the main loop, a generic search that can be instantiated into standard search algorithms in the obvious manner by suitable handling of the open and closed lists. The only difference to the standard algorithms then lies in (a) dead-end pruning via

---

**Algorithm 5.1:** Generic open & closed list based forward search algorithm, with conflict identification and learning.

---

**Input:** Classical planning task  $\Pi$  with initial state  $I$ , actions  $\mathcal{A}$ , and goal  $\mathcal{G}$ ,  
 Unsolvability detector  $\mathcal{U}$

**Output:** Plan  $\pi$  for  $\Pi$ , or “unsolvable”.

```

1 initialize empty search graph  $\hat{\Theta}$ ;
2 insert  $I$  into  $\hat{\Theta}$ , i.e., insert  $I$  into  $\hat{S}$ , and if  $\mathcal{G} \subseteq I$ , insert  $I$  into  $\hat{S}_*$ ;
3  $open \leftarrow \{I\}$ ;  $closed \leftarrow \emptyset$ ;  $conflicts \leftarrow \emptyset$ ;
4 while  $open \neq \emptyset$  do
5   select  $s \in open$ ;
6   if  $s \in \hat{S}_*$  then
7     return path in  $\hat{\Theta}$  from  $I$  to  $s$ ;
8    $closed \leftarrow closed \cup \{s\}$ ;
9   if  $\mathcal{U}(s) \neq \infty$  then                                /* expand if not recognized as dead end */
10    forall applicable actions  $a \in \mathcal{A}(s)$  do
11       $s' \leftarrow s[a]$ ; insert  $s'$  into  $\hat{\Theta}$  if not present;
12       $\hat{\mathcal{T}} \leftarrow \hat{\mathcal{T}} \cup \{ \langle s, a, s' \rangle \}$ ;
13      if  $s' \in closed \cup open$  then
14        continue;
15      if  $\mathcal{U}(s') = \infty$  then                                /* immediately close recognized dead ends */
16         $closed \leftarrow closed \cup \{s'\}$ ;
17      else
18         $open \leftarrow open \cup \{s'\}$ ;
19    CheckAndLearn( $s$ );
20 return unsolvable;
21 procedure CheckAndLearn( $s$ )
22   /* loop detection */
23   if  $s \in conflicts$  then
24     return;
25    $S \leftarrow \mathcal{R}[\hat{\Theta}](s)$ ;                                /* lookahead in search graph  $\hat{\Theta}$  */
26   if  $S \subseteq closed$  then
27     /* refinement (conflict analysis) */
28     refine  $\mathcal{U}$  s.t.  $\mathcal{U}(s') = \infty$  for every  $s' \in S \setminus conflicts$ ;
29     /* backward propagation */
30      $conflicts \leftarrow conflicts \cup S$ ;
31     forall predecessors  $t \in Pred[\hat{\Theta}](S)$  do
32       CheckAndLearn( $t$ );

```

---

$\mathcal{U}(s)$  at node expansion time (line 9), (b) pruning via  $\mathcal{U}(s')$  at node generation time (line 15), and (c) via a call to the CheckAndLearn procedure after state expansion. Of these, (a) and (b) are straightforward. A state is pruned, and considered closed, if it is detected to be a dead end. Note that (a) makes sense

despite the fact that  $s$  was already tested by (b) when it was first generated. This is because  $\mathcal{U}$  may have been refined in the meantime, and may now recognize  $s$  to be a dead end.

The conflict identification and learning process is organized by (c), the **CheckAndLearn** procedure. Consider any time point during search using Algorithm 5.1. The states in the *open*- & *closed* lists are exactly those states for which the search has found a path from the initial state. The *closed*-list contains those states that have been expanded, i.e., whose successors have already been added to the *open*- & *closed*-list, and those whose expansion has been pruned due  $\mathcal{U}$ . Vice versa, the *open*-list contains those states that have not been expanded. The “knowledge” gathered by the search is captured in terms of state-space subgraph  $\hat{\Theta}$ , the *search graph*. We call a state  $s$  a *known dead end* if, given  $\hat{\Theta}$ ,  $s$  must be a dead end, no matter of the result of future state expansions. To capture this formally, let  $\Theta^1 = \langle \mathcal{S}^1, \mathcal{L}^1, \mathcal{T}^1, s_I^1, \mathcal{S}_*^1 \rangle$  and  $\Theta^2 = \langle \mathcal{S}^2, \mathcal{L}^2, \mathcal{T}^2, s_I^2, \mathcal{S}_*^2 \rangle$  be two transitions systems that agree on their goal states, i.e.,  $\mathcal{S}_*^1 \cap \mathcal{S}_*^2 = \mathcal{S}_*^2 \cap \mathcal{S}_*^1$ , and let  $S \subseteq \mathcal{S}^1 \cap \mathcal{S}^2$  be a set of states joint among them. We say that  $\Theta^2$  **coincides with**  $\Theta^1$  **on**  $S$  if for every  $s \in S$ ,  $\langle s, a, s' \rangle \in \mathcal{T}^1$  if and only if  $\langle s, a, s' \rangle \in \mathcal{T}^2$ . Then

**Definition 5.2** (Known Dead End). *Let  $s \in \hat{S}$  be any state visited by search so far.  $s$  is a **known dead end** if  $s$  is a dead end in every LTS  $\Theta$  that coincides with  $\hat{\Theta}$  on *closed*.*

In other words, search knows  $s$  to be a dead end if that is so in all state spaces indistinguishable from the present one given the state-space structure unveiled by search so far.

It is easy to see that the known dead ends are exactly the states all of whose descendants in the search graph are already closed:

**Proposition 5.1.** *At any time point during the execution of Algorithm 5.1, the known dead ends are exactly those states  $s$  where  $\mathcal{R}[\hat{\Theta}](s) \subseteq \text{closed}$ .*

*Proof.* First, say that  $\mathcal{R}[\hat{\Theta}](s) \subseteq \text{closed}$ . Consider any descendant state  $s'$  of  $s$  in the current search graph. Then,  $s'$  is closed, either because it has been expanded, or because it has been detected as a dead end. In the former case, all outgoing transitions of  $s'$  lead to states in *closed*; in the latter case,  $s'$  does not have any outgoing transitions. Let now  $\Theta$  be a transition system that coincides with  $\hat{\Theta}$  on *closed*. Then all states reachable from  $s$  in  $\Theta$  are necessarily contained in *closed*. As  $\text{closed} \cap \mathcal{S}_* = \emptyset$ ,  $s$  must be a dead end in  $\Theta$ , which is what we needed to prove.

Vice versa, if  $\mathcal{R}[\hat{\Theta}](s) \not\subseteq \text{closed}$ , then some descendant  $s'$  of  $s$  in the current search graph is still open. We can construct a counter-example  $\Theta$  simply by extending  $\hat{\Theta}$  with a direct transition from  $s'$  to some goal state.  $\square$

Given this, a naive means to identify all known conflicts is to evaluate, after every state expansion and for every  $s \in \text{closed}$ , whether  $\mathcal{R}[\hat{\Theta}](s) \subseteq \text{closed}$ . But one can do much better than this, by a dead-end labeling procedure.

One might, at first sight, expect such a labeling procedure to be trivial, doing a simple bottom-up labeling following the reasoning that, if all direct successors of  $s$  are already known dead ends, then  $s$  is a known dead end as well. Such a simple procedure would, however, be incomplete, i.e., would in general not label all known dead ends, due to cycles. If states  $s_1$  and  $s_2$  are dead ends but have outgoing transitions to each other, then neither of the two will ever be labeled. Our labeling method, conducted as part of **CheckAndLearn**, thus involves complete lookaheads on the current search graph, but on only those states that might actually have become a known dead end given the last state expansion. Namely, a state  $t$  can only become a known dead end after the expansion of a descendant  $s$  of  $t$  where  $\mathcal{R}[\hat{\Theta}](s) \subseteq \text{closed}$  after

the expansion: otherwise, either the descendants of  $t$  have not changed at all, or  $t$  still has at least one descendant in *open*.

Consider now the bottom part of Algorithm 5.1. In the top-level call of `CheckAndLearn`( $s$ ),  $s$  cannot yet be labeled; the label check at the start of `CheckAndLearn` is needed only for loop detection in recursive invocations, cf. below. The test on  $S \subseteq \text{closed}$  corresponds to Proposition 5.1. Consider any  $t \in S$ . As  $t$  is reachable from  $s$ , we have  $\mathcal{R}[\hat{\Theta}](t) \subseteq \mathcal{R}[\hat{\Theta}](s) = S$ , and thus  $\mathcal{R}[\hat{\Theta}](t) \subseteq \text{closed}$ . So,  $t$  is a known dead end as well. Some  $t$  may however be recognized already,  $\mathcal{U}(t) = \infty$ , due to previous  $\mathcal{U}$  refinements, and thus may be dead ends, but not conflicts. If that is so for all  $t \in \mathcal{R}[\hat{\Theta}](s)$ , then the refinement step is void and can be skipped.

Backward propagation on the parents of  $S$  is needed to identify all dead ends known at this time. Observe that the recursion will eventually reach all ancestors  $t$  of  $s$ , and thus all states  $t$  that might have become a known dead end. Given the label check at the start of `CheckAndLearn`, every state is labeled at most once, and hence  $|\text{closed}|$  is an obvious upper bound on the number of recursive invocations, even if the state space contains cycles. Note that, in each recursive call, we need to backpropagate from the predecessors of *all* states from  $S = \mathcal{R}[\hat{\Theta}](s)$ , because  $\mathcal{R}[\hat{\Theta}](s)$  could itself contain ancestors  $t$  of  $s$ , while some other ancestor  $t'$  of  $s$  may be connected to  $s$  only via such a  $t$ . Due to the labeling in line 27,  $t'$  would otherwise not be reached by the recursion.

In short, we label known dead-end states bottom-up along forward search transition paths, conducting a full lookahead of the current search graph in each. With the arguments outlined above, this is sound and complete relative to the search knowledge:

**Theorem 5.1.** *At the start of the **while** loop in Algorithm 5.1, the labeled states are exactly the known dead ends.*

*Proof sketch.* Soundness, i.e.,  $t$  is labeled  $\Rightarrow t$  is a known dead end, holds because  $\mathcal{R}[\hat{\Theta}](t) \subseteq \text{closed}$  at the time of labeling. Completeness, i.e.,  $t$  is a known dead end  $\Rightarrow t$  is labeled, holds because the recursive invocations of `CheckAndLearn`( $t$ ) will reach all relevant states.  $\square$

The full proof is available in Appendix B.1.1.

**Example 5.1.** *To illustrate the conflict identification and propagation process, consider the variant of the running example depicted in Figure 5.1a. The goal is collecting all samples and disposing them at the base. Figure 5.1b draws a possible search space. Suppose that the unsolvability detector  $\mathcal{U}$  recognizes initially  $s_5$  and  $s_6$  as dead ends, and that the states are expanded by search in the order  $s_0, s_1, s_4, s_2, s_3$ . During the expansion of  $s_4$ , since  $\mathcal{U}(s_6) = \infty$ ,  $s_6$  is immediately inserted into *closed*. After that expansion, the call to `CheckAndLearn`( $s_4$ ) constructs  $\mathcal{R}[\hat{\Theta}](s_4) = \{s_4, s_6, s_1, s_5\}$ , and finds that  $\mathcal{R}[\hat{\Theta}](s_4) \subseteq \text{closed}$ . Thus,  $\mathcal{U}$  is refined to recognize  $s_4$  and  $s_1$ , the refinement for  $s_5$  and  $s_6$  can be skipped, which are already recognized by  $\mathcal{U}$ . Backward propagation then calls `CheckAndLearn`( $s_1$ ), `CheckAndLearn`( $s_4$ ), as well as `CheckAndLearn`( $s_0$ ). Since  $s_1$  and  $s_4$  have just been labeled, the former calls terminate immediately. The latter call finds that  $\mathcal{R}[\hat{\Theta}](s_0) \not\subseteq \text{closed}$ , so the procedure terminates here. Note that  $s_0$  would not have been reached during backward propagation if line 27 looped just over the parents of  $s_4$ . Search resumes with the expansion of  $s_2$  and  $s_3$ . We come back to this example in the next chapter.*

It is worth noting that the search “knowledge” considered in the above is only the *explicit* knowledge, about states the search has already expanded or pruned. This disregards the *implicit* knowledge potentially present due to generalization: refining  $\mathcal{U}$  on  $\mathcal{R}[\hat{\Theta}](s)$  might recognize dead ends  $t' \notin \mathcal{R}[\hat{\Theta}](s)$ . In

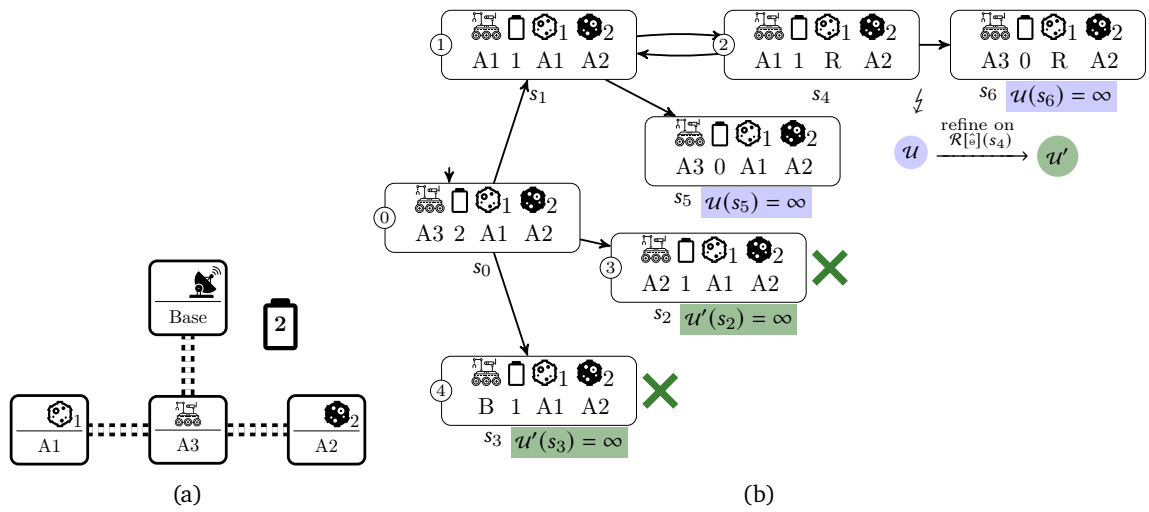


Figure 5.1.: (a) depicts a variant of the rover example from Section 1.2, in which samples from  $A_1$  and  $A_2$  need to be collected, and subsequently dropped at the base. (b) shows an example search space. The states' number annotations give the order in which states are expanded.

particular, the search might have already visited some  $t' \in \text{open} \cup \text{closed}$ , and then, via  $u$ , the search might actually already know that  $t'$  (and potentially some of its ancestors) are dead ends.

One can capture this formally by denoting with  $\hat{S}_\infty = \{s \in \hat{S} \mid u(s) = \infty\}$  the currently recognized dead-end states; defining  $\hat{\Theta}[\hat{S}_\infty]$  to be like  $\hat{\Theta}$  except that all  $s \in \hat{S}_\infty$  have no outgoing transitions; and defining a state to be a  $u$ -known dead end if it is a dead end in all  $\Theta$  that coincide with  $\Theta[\hat{S}_\infty]$  on  $\text{closed} \cup \hat{S}_\infty$ . The  $u$ -known dead ends then are exactly those  $s$  where  $\mathcal{R}[\hat{\Theta}](s) \subseteq \text{closed} \cup \hat{S}_\infty$ . To find all these  $s$  during search – and thus immediately learn from all already identified dead-end states – after every refinement, we would have to reevaluate  $u$  on the entire  $\text{open}$  and  $\text{closed}$  lists (plus backward propagation whenever a new dead end is found). This would cause prohibitive overhead. Hence, we stick to learning only on the known dead ends, explicitly captured by the search.

Algorithm 5.1 has several desirable properties regardless of its concrete instantiation:

- (P1) **Preserving guarantees:** Instantiating the main loop to reflect any standard search algorithm, the optimality and/or completeness guarantees of that algorithm are preserved, as the only change is the pruning of dead-end states.
- (P2) **Unsolvability certificate:** When search terminates with “unsolvable”, we have  $u(I) = \infty$ , due to the final call to `CheckAndLearn`, doing backward propagation when all nodes are closed.

In case an unsolvability certificate is not required, the final call to `CheckAndLearn` is redundant work. In our implementation, we provide an *early stopping* option, which skips that step when the open list is already empty.

- (P3) **Bail-out:** Provided that the unsolvability heuristic is transitive, search terminates without any further state expansion, as soon as an unsolvability certificate is found. Namely, for a transitive  $u$ , when  $u(I) = \infty$ , then  $u(s) = \infty$  necessarily holds for all reachable  $s$ , and thus no more states will be expanded.

We remark that the problem of labeling the known dead-end states relates closely to cost-revision steps,

and the *solved*-labeling procedures available in cyclic AND/OR graph search algorithms (e.g., Jiménez and Torras, 2000; Bonet and Geffner, 2003a). It remains an open question whether such methods can be beneficial for our purposes. Yet, as we shall see next, in depth-first search – which turns out to be most useful in practice – the issue of forward lookaheads disappears.

## 5.2. Depth-First Search

Depth-first search (DFS) is particularly well suited for our purposes, because it fully explores the descendants of a state before proceeding with anything else. In other words, DFS is geared at obtaining  $\mathcal{R}[\hat{\Theta}](s) \subseteq \text{closed}$  as quickly as possible. This is key to identifying conflicts quickly.

But what exactly does DFS look like in our context? The issue is that state spaces are, in general, cyclic, and nodes may have solutions via their parents. A simple way to tackle this is what we will refer to as *depth-oriented search* (DOS), instantiating Algorithm 5.1 with a depth-first search order, ordering the open list by decreasing distance from the initial state.

It turns out that one can do better though. We next design an elegant DFS variant of our approach, similar to backtracking in constraint satisfaction problems. Consider first, to get some intuitions, the acyclic case. This is restricted yet not entirely unrealistic: acyclic state spaces naturally occur, e.g., if every action consumes a non-0 amount of budget or resource. In DFS on an acyclic state space, state  $s$  becomes a known dead end exactly the moment its subtree has been completed, i.e., when we backtrack out of  $s$ . Hence we can simply refine  $\mathcal{U}$  at this point. As the same has previously been done on the children  $s'$  of  $s$ , we will have  $\mathcal{U}(s') = \infty$  for every such  $s'$ , so the conflict component  $\mathcal{R}[\hat{\Theta}](s)$  simplifies to just  $s$ . Overall, the complex *CheckAndLearn* procedure can be replaced by refining  $\mathcal{U}$  on  $s$  at backtracking time. But then, we do not need the *open* and *closed* lists anymore, and can instead fallback to a classical DFS.

In the cyclic case, matters are not that easy. But it turns out that one can obtain a valid DFS algorithm (which defaults to classical DFS in the acyclic case) from Tarjan’s algorithm to compute maximal strongly connected components (Tarjan, 1972), cf. Chapter 3.

Algorithm 5.2 shows the pseudo-code. The key observation is that  $s$  becomes a known dead end exactly at the moment when we have identified the maximal SCC  $S \subseteq \mathcal{S}$  that contains  $s$ , i.e., once DFS backtracks out of the last state in  $S$ . This is simply because, with  $\mathcal{R}[\hat{\Theta}](s) \subseteq \text{closed}$ , we must also have  $\mathcal{R}[\hat{\Theta}](t) \subseteq \text{closed}$  for any ancestor state  $t$  of  $s$  reachable from  $s$ . Thus, to get rid of the expensive *CheckAndLearn* procedure, DFS can use Tarjan’s algorithm to identify the maximal SCCs, and refine  $\mathcal{U}$  whenever a maximal SCC has been found. Henceforth, whenever we say “DFS”, we mean DFS as per Algorithm 5.2.

Regarding the properties of DFS, obviously property **(P1)** (preserving guarantees) from above is not meaningful here; DFS is complete but not optimal. DFS inherits properties **(P2)** (unsolvability certificate) and **(P3)** (bail-out). Like for Algorithm 5.1, we implemented a simple *early stopping* option in case an unsolvability certificate is not desired. DFS furthermore has several desirable properties beyond **(P2)** and **(P3)**:

**(P4) Backjumping:** Due to the pruning test on  $\mathcal{U}(s) = \infty$  inside the state-expansion loop, DFS will backjump across predecessor states  $s$  that are now recognized dead ends. For transitive unsolvability heuristics, the backjump will be across *all* recognized dead ends on the current search path, as  $\mathcal{U}(s) = \infty$  implies  $\mathcal{U}(t) = \infty$  for all  $t$  below  $s$ .

**(P5) Immediate  $\mathcal{U}$ -known learning:** DFS guarantees to learn, before the next state expansion, on all

---

**Algorithm 5.2:** Depth-first search (DFS), with conflict identification and learning following Tarjan’s algorithm.

---

**Input:** Classical planning task  $\Pi$  with initial state  $I$ , actions  $\mathcal{A}$ , and goal  $\mathcal{G}$ ,  
 Unsolvability detector  $\mathcal{U}$

**Output:** Plan  $\pi$  for  $\Pi$ , or “unsolvable”.

```

1 initialize empty search graph  $\hat{\Theta}$ ;
2 insert  $I$  into  $\hat{\Theta}$ , i.e., insert  $I$  into  $\hat{S}$ , and if  $\mathcal{G} \subseteq I$ , insert  $I$  into  $\hat{S}_*$ ;
3  $stack \leftarrow$  empty stack;
4  $minReach \leftarrow$  empty map;
  /*  $conflicts \leftarrow \emptyset$ ; (to illustrate the relation to Algorithm 5.1) */
5 if  $DFS(I)$  then return path in  $\hat{\Theta}$  from  $I$  to the corresponding goal state  $s_*$ ;
6 else return unsolvable;

7 procedure  $DFS(s)$ 
8   if  $\mathcal{G} \subseteq s$  then
9     return  $\top$ ;
10  if  $\mathcal{U}(s) = \infty$  then
11    return  $\perp$ ;
12   $stackIdx \leftarrow |stack|$ ;  $minReach[s] \leftarrow stackIdx$ ; push  $s$  onto  $stack$ ;
13  forall applicable actions  $a \in \mathcal{A}(s)$  do
14     $s' \leftarrow s \parallel a$ ; insert  $s'$  into  $\hat{\Theta}$  if not present;  $\hat{\mathcal{T}} \leftarrow \hat{\mathcal{T}} \cup \{ \langle s, a, s' \rangle \}$ ;
15    if  $s' \in stack$  then
16       $minReach[s] = \min(minReach[s], minReach[s'])$ ;
17    else
18      if  $DFS(s')$  then
19        return  $\top$ ;
20      if  $s' \in stack$  then
21         $minReach[s] = \min(minReach[s], minReach[s'])$ ;
22      if  $\mathcal{U}(s) = \infty$  then /* re-evaluate  $\mathcal{U}$  due to possible refinement */
23        break;
24  if  $minReach[s] = stackIdx$  then /* backtrack from SCC; found conflict */
25     $S \leftarrow \emptyset$ ;
26    while  $s \notin S$  do
27       $t \leftarrow$  pop from  $stack$ ; delete  $minReach[t]$ ;
28       $S \leftarrow S \cup \{ t \}$ ;
29    /* it holds that  $S = \mathcal{R}[\hat{\Theta}](s) \setminus conflicts$  */
30    refine  $\mathcal{U}$  s.t.  $\mathcal{U}(t) = \infty$  for every  $t \in S$ ;
31    /*  $conflicts \leftarrow conflicts \cup S$ ; */
32  return  $\perp$ ;
```

---

dead ends  $t'$  that are  $\mathcal{U}$ -known but not known, and where  $\mathcal{U}(t') \neq \infty$  (there is still something to learn on  $t'$ ). To see this, let  $t'$  be such a state. As  $t'$  is not a known dead end, there must be an open

leaf node reachable from  $t'$ , i.e.,  $t'$  must still be on the stack. Yet, as  $\mathcal{U}(t') \neq \infty$  but  $t'$  is a  $\mathcal{U}$ -known dead end, every leaf node  $s$  reachable from  $t'$  must satisfy  $\mathcal{U}(s) = \infty$ . Before search can make the next state expansion, it must evaluate all those leaf nodes, and backtrack out of the SCC containing  $t'$  – which is exactly what we need to show.

**(P6) Duplicate pruning for free:** As  $\mathcal{U}$  learns to refute the subtree below  $s$ , it subsumes the duplicate pruning that would be afforded by a closed list. Due to generalization, it will often surpass that pruning by far.

Compared to this, in the generic search of Algorithm 5.1, towards **(P4)** one can test, at the start of the `CheckAndLearn` procedure, whether  $\mathcal{U}(s) = \infty$ . This leads to backjumping in depth-oriented search, and leads to aggressive pruning of search paths in other open list based searches like greedy best-first search. For **(P5)**, as discussed this does not hold for Algorithm 5.1 in general, as the new  $\mathcal{U}$ -known states may lie on arbitrary search paths; it does hold for depth-oriented search though. Finally, **(P6)** is specific to DFS, and cannot be exploited by Algorithm 5.1 regardless of the search order, as that algorithm needs to maintain a closed list anyway. Intuitively, depth-first search is closer to the structure of dead-end detection, and combines more gracefully with it than other search algorithms.

## 6. Critical-Path Heuristics: Conflict Refinement & NoGood Learning

The family of *critical-path* heuristic functions  $h^m$ , introduced in its general form by Haslum and Geffner (2000), has a long tradition of use as effective sufficient criteria for unsolvability. If  $h^m(s) = \infty$ , then the goal cannot be reached from  $s$  even when allowing to break up conjunctive subgoals into their *atomic subgoals*, namely the fact conjunctions of size  $\leq m$ . This idea has its roots in the use of  $h^2$  as encoded by the planning graph, GRAPHPLAN’s mechanism for early termination on unsolvable tasks (Blum and Furst, 1997). The idea persisted in delete-relaxation heuristics (Bonet and Geffner, 2001; Hoffmann and Nebel, 2001), as well as some partial delete-relaxation heuristics (Domshlak et al., 2015), which identify a state to be a dead end if and only if  $h^1$  does. In practical regards, the computation of  $h^m$  remains feasible only for small  $m$ , typically restricting  $m \leq 2$ , as the number of atomic subgoals grows exponentially in  $m$ . Dealing efficiently with larger atomic subgoals has been the subject of a series of works (Haslum, 2006; Haslum, 2012; Keyder et al., 2014). Most recently, Hoffmann and Fickert (2015) introduced the heuristic  $h^C$ , whose parameter  $C$  allows one to choose the *atomic conjunctions* to reason about completely freely.

Denote the unsolvability-detector variant of  $h^C$ , that returns  $\infty$  iff  $h^C$  does, by  $u^C$ . It is well known that, for sufficiently large  $m$ ,  $h^m$  delivers perfect goal distance estimates: simply set  $m$  to the number of state variables, reasoning over all relevant conjunctions. As a corollary, for the corresponding set  $C$ ,  $u^C$  detects all dead ends. This choice of conjunctions is impracticable, of course. Yet, for the purpose of recognizing all dead ends, it is reasonable to belief that much smaller sets  $C$  suffice. Namely, first, this requires accuracy not on all states, but only on dead ends. Secondly, it is quite natural for  $\mathcal{G}$  to be unsolvable because some small  $c \subseteq \mathcal{G}$  is. But then, how to find  $C$ ? One possibility is to use known conjunction-learning methods from the literature (Haslum, 2012; Keyder et al., 2014), which iteratively remove flaws in delete-relaxed plans for a given state  $s$ . These methods do guarantee to eventually recognize  $s$  if it is a dead end. But they are not geared to this purpose, and as we shall see, are not effective in practice for that purpose. In this chapter, we introduce two refinement methods specifically designed for dead-end detection.

Like  $h^C$ , the computation of  $u^C$  requires low-order polynomial time in the number of atomic conjunctions  $|C|$ . Nevertheless, as  $C$  becomes larger, computing  $u^C$  on every state in search may cause substantial runtime overhead. We alleviate this overhead through *NoGoods*, formulae  $\varphi$  where  $s \not\models \varphi$  implies that  $u^C(s) = \infty$ . These NoGoods act as a filter in front of the  $u^C$  computation, skipping the computation if  $s \not\models \varphi$  for some learned  $\varphi$ . Here, we observe that it is actually possible to construct a *perfect* NoGood formula  $\Phi^{C*}$ , where  $s \not\models \Phi^{C*}$  if and only if  $u^C(s) = \infty$ . This is mostly of theoretical interest though, because  $\Phi^{C*}$  has size worst-case exponential in the size of the input planning task. We obtain practical variants by instead learning weaker NoGood formulae “online”, i.e.,  $\varphi$  such that  $\neg\varphi \Rightarrow \neg\Phi^{C*}$ , refining  $\varphi$  whenever encountering in search a state  $s$  where  $s \models \varphi$  but  $u^C(s) = \infty$ . We introduce and compare two such approaches: *clause learning*, inspired by prior work (Kolobov et al., 2012b), and *CART learning*, leveraging the perfect NoGood formula construction.

The remainder of this chapter is structured as follows. In Section 6.1, we provide the formal definition of the

critical-path heuristic  $h^C$ , as per Hoffmann and Fickert (2015). In Section 6.2, we discuss the refinement methods. Section 6.3 investigates critical-path NoGoods. Finally, Section 6.4 provides a comprehensive experimental evaluation of search using conflict-driven learning via  $u^C$  for the purposes of finding plans on solvable tasks with dead ends; proving unsolvability on unsolvable tasks; as well as generating *unsolvability certificates* for the latter tasks.

This chapter is based on (Steinmetz and Hoffmann, 2017c; Steinmetz and Hoffmann, 2017a). The refinement procedures, Sections 6.2.1 and 6.2.2, are due to prior work (Steinmetz, 2015). We simplified the procedures for the purpose of unsolvability detection, and applied minor optimizations to the conjunction selection resolving a conflict. The correctness arguments given by Steinmetz (2015) are not affected, and we restate them only for the sake of comprehensibility and completeness.

## 6.1. Preliminaries

In the following, we assume planning tasks in STRIPS notation (cf. Definition 2.1). Let  $\Pi = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  be such a task. To formalize critical-path heuristics, we need the notion of *regression*:

**Definition 6.1** (STRIPS Regression). *Let  $P \subseteq \mathcal{F}$  be a set of facts, and let  $a \in \mathcal{A}$  be an action. If  $\text{add}(a) \cap P \neq \emptyset$  and  $\text{del}(a) \cap P = \emptyset$ , then the **regression** of  $P$  over  $a$  is*

$$\text{regress}(P, a) = (P \setminus \text{add}(a)) \cup \text{pre}(a)$$

*Otherwise, the regression is undefined, and we write  $\text{regress}(P, a) = \perp$ .*

In words, the regression of  $P$  over some  $a$  lists all facts that need to be satisfied in a state  $s$  such that the application of  $a$  results in  $P \subseteq s \llbracket a \rrbracket$ . We denote by  $\mathcal{A}[P] \subseteq \mathcal{A}$  the set of actions, whose  $P$ -regression is defined.

We identify fact conjunctions with fact sets; let  $C \subseteq 2^{\mathcal{F}}$  be any set of conjunctions. The generalized critical-path heuristic  $h^C$  is defined as follows

**Definition 6.2** (Critical-Path Heuristic  $h^C$ ). *The **critical-path heuristic** over  $C$  is the element-wise maximal function  $h^C : \mathcal{S}^\Pi \times 2^{\mathcal{F}} \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$  that satisfies the equation:*

$$h^C(s, P) = \begin{cases} 0, & \text{if } P \subseteq s \\ \min_{a \in \mathcal{A}[P]} (\text{c}(a) + h^C(s, \text{regress}(P, a))), & \text{if } P \in C \\ \max_{c \in C : c \subseteq P} h^C(s, c) & \text{otherwise} \end{cases} \quad (6.1)$$

*for all  $s \in \mathcal{S}^\Pi$ , and  $P \subseteq 2^{\mathcal{F}}$ . We assume that  $\min_\emptyset = \infty$ , and  $\max_\emptyset = 0$ . Let  $s$  be any state. For convenience, we write  $h^C(s)$  to denote  $h^C(s, \mathcal{G})$ .*

Note here that we overload  $h^C$  to denote both, a function of state  $s$  in which case the estimated distance from  $s$  to the global goal  $\mathcal{G}$  is returned, and a function of state  $s$  and subgoal  $P$  in which case the estimated distance from  $s$  to  $P$  is returned. Similarly, we will occasionally use  $h^*(s, P)$  to denote the cost of the cheapest path from  $s$  to any state that satisfies  $P$ ;  $h^*(s, P) = \infty$  if  $P$  is unreachable from  $s$ .

The definition of  $h^C$  distinguishes between three cases. If the subgoal is already true in the considered state  $s$  (top case), then its value is 0. If the subgoal  $P$  is not an atomic conjunction (bottom case), then its  $h^C$  value is estimated by the most expensive atomic subgoal that is a subset of  $P$ . Otherwise (middle case),  $P$  is an atomic subgoal that is not already true in the considered state. Then the  $h^C$  value is set to

the cheapest possible way of achieving  $P$ , by minimizing over the actions that can be used to achieve  $P$ , and computing the resulting costs recursively for each.

If  $h^C(s, c) = \infty$  for some atomic conjunction  $c \in C$ , then the overall outcome might be  $h^C(s) = \infty$ .  $h^C(s, c) = \infty$  holds trivially, if  $c$  is not satisfied in  $s$ , and there exists no action that can be used to achieve  $c$ , i.e., the minimization in the middle case of (6.1) evaluates to  $\infty$ .  $h^C(s, c) = \infty$  may furthermore hold if the regression of  $c$  over every action  $a \in \mathcal{A}[c]$  contains some other  $c' \in C$  with  $h^C(s, c') = \infty$ . Note that this might be the case, even if all atomic conjunctions have a non-empty set of achievers, and thus even if none of them falls into the trivial base case. For example, consider the singleton conjunctions, i.e.,  $h^1$ , and suppose there are just two actions with  $\text{pre}(a_1) = \{p\}$  and  $\text{add}(a_1) = \{q\}$ , respectively  $\text{pre}(a_2) = \{q\}$  and  $\text{add}(a_2) = \{p\}$ . If neither  $p$  nor  $q$  is satisfied in  $s$ , then  $h^1(s, \{p\}) = h^1(s, \{q\}) = \infty$  as per the element-wise maximality requirement. As we are interested only in dead-end detection, not goal distance estimation,  $h^C(s) = \infty$  is the main ability of  $h^C$  we are interested in. Consequently, most of the time we will consider not  $h^C$  but the *critical-path unsolvability detector*, defined by  $u^C(s) = \infty$  if  $h^C(s) = \infty$ , and  $u^C(s) = 0$  otherwise. We will sometimes use  $u^m$  to denote the unsolvability detector variant of  $h^m$ .

**Example 6.1.** *Reconsider Example 5.1. The task can be represented in STRIPS as shown Example 2.1. The initial state is  $\mathcal{I} = \{\text{rov}(A_3), \text{bat}(2), \text{samp}_1(A_1), \text{samp}_2(A_2)\}$ . The goal is  $\mathcal{G} = \{\text{samp}_1(B), \text{samp}_2(B)\}$ . We assume unit cost (where relevant).*

*Consider the states  $s_4$  and  $s_6$  depicted in Figure 5.1b:  $s_4 = \{\text{rov}(A_1), \text{bat}(1), \text{samp}_1(R), \text{samp}_2(A_2)\}$ ,  $s_6 = \{\text{rov}(A_3), \text{bat}(0), \text{samp}_1(R), \text{samp}_2(A_2)\}$ .*

*Note that  $u^1(s_6) = u^1(s_6, \mathcal{G}) \geq u^1(s_6, \{\text{samp}_1(B)\}) = \infty$ . This is true because even when allowed to split subgoals into facts, it remains impossible to move the rover to  $B$  with no energy left, which is needed for dropping  $\text{samp}_1$  at  $B$ .*

*In contrast,  $u^1(s_4) < \infty$ , because under  $u^1$ 's relaxing assumptions, one loses the information that a single battery unit is not sufficient to move the rover to all locations. It holds that  $u^1(s_4, \{\text{rov}(A_3)\}) < \infty$ , because  $\text{move}(A_1, A_3, 1)$  achieves  $\text{rov}(A_3)$ , and all its preconditions are contained in  $s_4$ . Then,  $u^1(s_4, \{\text{rov}(A_2)\}) < \infty$  because  $\text{move}(A_3, A_2, 1) \in \mathcal{A}[\{\text{rov}(A_2)\}]$  and  $u^1(s_4, \{\text{bat}(1)\}) < \infty$  and  $u^1(s_4, \{\text{rov}(A_3)\}) < \infty$  yield  $u^1(s_4, \text{pre}(\text{move}(A_3, A_2, 1))) < \infty$ .  $u^1(s_4, \{\text{rov}(B)\}) < \infty$  follows similarly. Since none of the positions are detected unreachable, all samples can be collected from their initial positions, and dropped at  $B$ .*

*Consider the set of conjunctions  $C$ , which contains  $c = \{\text{rov}(A_3), \text{bat}(1)\}$  in addition to the singletons. Observe that the only actions whose add effects intersect with  $c$  are  $\text{move}(x, y, k)$  where  $y = A_3$  or  $k = 2$ . If  $y \neq A_3$ , then  $x = A_3$  as per the connections in the example, and the corresponding action deletes some parts of  $c$ . Similarly for  $k = 1$ . In conclusion, the achievers  $\mathcal{A}[c]$  are exactly the move actions with  $y = A_3$  and  $k = 2$ . However, the preconditions of all those actions is unreachable from  $s_4$  even under  $u^1$ , and thus  $u^C(s_4, c) = \infty$ . Since  $c \subseteq \text{pre}(\text{move}(A_3, B, 1))$ , it follows that  $u^C(s_4, \{\text{rov}(B)\}) = \infty$ , and consequently  $u^C(s_4) = \infty$ .*

For all sets  $C$ ,  $h^C$  is goal-aware and consistent, and thus admissible. In other words, whenever  $u^C(s) = \infty$  then  $s$  is a dead end, i.e.,  $u^C$  indeed is an unsolvability detector as per Definition 5.1. Furthermore, as an implication of consistency,  $u^C$  is transitive, i.e., as  $h^C(s) \leq h^C(s \llbracket a \rrbracket) + c(a)$ , for all  $s \in \mathcal{S}^\Pi$  and  $a \in \mathcal{A}(s)$ , if  $h^C(s) = \infty$  then so is  $h^C(s \llbracket a \rrbracket) = \infty$ .

One can compute  $u^C$ , solving Equation 6.1, in time polynomial in  $|C|$  and the size of  $\Pi$ , using simple dynamic programming algorithms similar to those for  $h^m$  (Haslum and Geffner, 2000).

## 6.2. Conflict Analysis & Refinement

We now tackle the refinement step in Algorithms 5.1 and 5.2 for the unsolvability detector  $u^C$ . Given a set of dead ends  $S \subseteq S^\Pi$ , how to refine  $u^C$  to recognize the states from  $S$ ? Naturally, the refinement will extend  $C$  by new atomic conjunctions  $X \subseteq 2^{\mathcal{F}}$  so that  $u^{C \cup X}(s) = \infty$  for all  $s \in S$ . But which conjunctions  $X$  are actually necessary to this end, and how to find these?

In Section 6.2.1 and Section 6.2.2, we introduce two refinement approaches, specifically designed for dead-end detection: *path-cut refinement* and *neighbors refinement*. The major difference between the two methods lies in their applicability. Neighbors refinement applies only to  $S$  that satisfy what we call the  $u^C$  *recognized-neighbors* property, i.e., where  $u^C(t) = \infty$  holds for every  $t \notin S$  that is a successor of some  $s \in S$ . It turns out that this can be exploited for an especially effective refinement method, neighbors refinement.

The recognized-neighbors property necessarily holds if  $u^C$  is the only unsolvability detector used in search. But if  $u^C$  is combined with some other  $u$ , then some of  $S$ 's successor states  $t$  may be recognized only by  $u$ . For the general case, we design the alternative path-cut refinement method. It computes conjunctions  $X$  by cutting off the critical paths in a  $h^C$  computation reaching the goal. One such refinement step guarantees to strictly increase the value of  $h^C$ . To render  $h^C$  infinite as desired, we need to iterate these refinement steps, recomputing  $h^C$  in between iterations. Neighbors refinement, in contrast, is a constructive method, identifying the new conjunctions  $X$  directly from those for the neighbor states, without necessitating any intermediate recomputations of  $u^C$ .

Section 6.2.3 closes the discussion with a worst-case analysis, shedding light on the question how expensive a single  $u^C$  refinement can be. At first glance, this seems obvious, as given the complexity of the plan existence problem versus the computational cost of  $u^C$ , letting  $u^C$  recognize an *arbitrary* dead end generally requires exponentially many conjunctions. However, how is the situation if we consider particularly a refinement step on a conflict  $S = \mathcal{R}[\hat{\Theta}](s)$  that is identified by search? As we shall see, there are in fact pathological cases where expanding the dead-end detection capabilities of  $u^C$  even by just a single transition step requires an exponentially large refinement  $X$ .

### 6.2.1. Path-Cut Refinement

Path-cut refinement assumes some arbitrary dead-end state  $s$  as input, and augments  $C$  to recognize  $s$ . To recognize all dead ends within the search-conflict component  $\mathcal{R}[\hat{\Theta}](s)$ , we run the method on  $s$  only. Due to the aforementioned transitivity property of  $u^C$ , this suffices to recognize all states in  $\mathcal{R}[\hat{\Theta}](s)$ .

The refinement is based on cutting off *critical paths*, i.e., the recursion paths in the definition of  $h^C$  (Equation 6.1). The refinement is iterative, where each iteration identifies a set  $X$  of conjunctions, adding which into  $C$  guarantees to strictly increase  $h^C(s)$ . Given this, the method really pertains to  $h^C$  rather than the simplified  $u^C$ , and it applies not only to dead-end states, but to any state  $s$  where  $h^C(s) < h^*(s)$ . Therefore, for the remainder of this subsection, we will talk about  $h^C$ , not  $u^C$ . At the end of the refinement on a dead-end state  $s$ , we will have  $h^C(s) = u^C(s) = \infty$ . For simplicity, we assume uniform action cost, i.e.,  $c(a) = 1$  for all  $a \in \mathcal{A}$ . This comes without loss of generality in our context, as for  $h^C(s) = \infty$  action costs are irrelevant. As shown by Steinmetz (2015), the refinement method can be extended to general cost functions, so to compute  $C$  with  $h^C(s) = h^*(s)$ , for arbitrary states  $s$ .

We consider now in detail a single refinement step (one iteration of the overall refinement). The  $h^C$  recursion path on a current subgoal  $P$  is cut off by identifying a small conjunction  $x \subseteq P$  that cannot be

---

**Algorithm 6.1:** A single step of the path-cut refinement.
 

---

**Input:** Set of conjunctions  $C$ ,  
 State  $s$  for which  $h^C(s) < h^*(s)$

**Output:** Set of conjunctions  $X$  such that  $h^{C \cup X}(s) > h^C(s)$

```

1  $X \leftarrow \emptyset$ ;
2 PathCutRefine( $\mathcal{G}, h^C(s)$ );
3 return  $X$ ;

4 procedure PathCutRefine( $P, N$ )
5   if  $N = 0$  then
6     /* We know here that  $P \not\subseteq s$  */
7     let  $p \in (P \setminus s)$ ;
8      $x \leftarrow \{p\}$ ;
9   else
10    /* Select an atomic conjunction (invariant:  $h^C(s, P) \geq N$ ) */
11    let  $c \in C$  be s.t.  $c \subseteq P$  and  $h^C(s, c) \geq N$ ;
12     $x \leftarrow c$ ;
13    if  $h^C(s, c) = N$  then
14      /* Cut each path that achieves  $c$  */
15      for every action  $a \in \mathcal{A}[c]$  do
16        if  $\text{del}(a) \cap P \neq \emptyset$  then
17          let  $p \in \text{del}(a) \cap P$ ;
18           $x \leftarrow c \cup \{p\}$ ;
19          /*  $\Rightarrow a$  is no longer an achiever of  $x$  */
20        else
21           $x' \leftarrow \text{PathCutRefine}(\text{regress}(P, a), N - 1)$ ;
22           $x \leftarrow x \cup (x' \setminus \text{pre}(a))$ ;
23          /*  $\Rightarrow \text{regress}(x, a)$  contains  $x'$  */
24     $X \leftarrow X \cup \{x\}$ ;
25  return  $x$ ;
```

---

achieved with action sequences of length at most  $h^C(s, P)$  (recall that we assume unit action cost, under which cost and length become the same). The union of these  $x$  over all recursion paths yields the desired set  $X$ . Algorithm 6.1 shows the pseudo-code.

To understand Algorithm 6.1, consider the initializing call on  $P = \mathcal{G}$  and  $N = h^C(s)$ . Our aim is to identify a (small) conjunction  $x \subseteq \mathcal{G}$  that cannot be achieved from  $s$  by any action sequence of length at most  $h^C(s, \mathcal{G})$ . Towards finding such  $x$ , we start by selecting an arbitrary atomic conjunction  $c \in C$ ,  $c \subseteq \mathcal{G}$ , with maximal  $h^C(s, c)$  value. We initialize  $x = c$ . As, by selection,  $h^C(s, c) = N$ ,  $c$  is achieved under the  $h^C$  approximation by an action sequence with length  $N$ . However, as  $N = h^C(s, \mathcal{G}) < h^*(s, \mathcal{G})$ , we know that we can extend  $x$  with additional facts  $p \in \mathcal{G} \setminus c$  in a way excluding that case, i.e., making  $x$  achievable under  $h^C$  only by action sequences with length strictly larger than  $N$ .

To find suitable facts  $p$  for extending  $x$ , we recursively consider the actions  $a \in \mathcal{A}[c]$ , i.e., the actions that

can achieve  $c$  (that add part of  $c$  and delete none of it). For each of these, we augment  $x$  so that there is a conjunction  $x' \subseteq \text{regress}(x, a)$  that cannot be achieved by action sequences with length  $< N$ . If  $a$  deletes part of  $P$ , we can tackle  $a$  simply by adding one such deleted fact  $p$  into  $x$ , effectively removing  $a$  from the set of achievers of  $x$ . For the remaining actions  $a$ , we recursively identify a suitable  $x' \subseteq \text{regress}(P, a)$ . The latter is necessarily possible as we will always have  $h^C(s, P) < h^*(s, P)$  (in particular,  $P \not\subseteq s$  at the recursion termination  $N = 0$ ). We then extend  $x$  in a way ensuring that  $x'$  is contained in the regression  $\text{regress}(x, a)$ , implying that  $x$  cannot be reached at time  $N$ .

Note that it suffices to consider only the achievers of  $c$ , despite that adding new facts into  $x$  may make other actions  $a \notin \mathcal{A}[c]$  become achievers of  $x$ . Yet, for these it necessarily holds that  $c \subseteq \text{regress}(x, a)$ , and thus they already satisfy  $h^C(s, \text{regress}(x, a)) \geq N$  as per the selection of  $c$ .

Spelling out these arguments, one obtains that `PathCutRefine` is correct:

**Theorem 6.1.** *Suppose  $\mathfrak{c}(a) = 1$  for all  $a \in \mathcal{A}$ . Let  $C$  be any set of atomic conjunctions. Let  $s$  be a state with  $h^C(s) < h^*(s)$ . Then:*

- (i) *The execution of `PathCutRefine`( $\mathcal{G}, h^C(s)$ ) terminates eventually, and is well defined, i.e., (a) in any call `PathCutRefine`( $P, N$ ) there exists  $c \in C$  so that  $c \subseteq P$  and  $h^C(s, c) \geq N$ ; and (b) if  $N = 0$ , then  $P \not\subseteq s$ .*
- (ii) *If  $X$  is the set of conjunctions resulting from `PathCutRefine`( $\mathcal{G}, h^C(s)$ ), then  $h^{C \cup X}(s) > h^C(s)$ .*

Appendix B.2.1 spells out the detailed proof arguments.

As a single call to `PathCutRefine` only guarantees to increase  $h^C(s)$  by at least 1, for dead-end refinement we need to iterate these calls, setting  $C = C \cup X$  after each call, until  $h^C(s) = \infty$  holds. This is guaranteed to eventually happen, simply because every iteration adds at least one new conjunction to  $C$  (otherwise, the value of  $h^C$  could not have increased), and the number of conjunctions is finite. In the worst case,  $C$  eventually contains all conjunctions, and  $h^C(s) = \infty$  holds trivially.

**Example 6.2.** *Reconsider Example 5.1. Recall from Example 6.1 that  $\mathcal{U}^1$  recognizes  $s_6$  as a dead end. For the same reasons, it holds that  $\mathcal{U}^1(s_5) = \infty$ , i.e.,  $\mathcal{U}^1$  prunes exactly those states that were pruned by the hypothetical unsolvability detector  $\mathcal{U}$  used in the search from Example 5.1. Consider the first conflict identified in that search:  $s_4$ . Say that we conduct path-cut refinement on  $s = s_4$  to find a suitable extension  $X$  to the singleton conjunctions.*

*In the first call to `PathCutRefine`, we have*

$$P_4 := \mathcal{G} = \{ \text{samp}_1(B), \text{samp}_2(B) \}$$

*and  $N = h^1(s) = 4$ . There is only one option for the selection of  $c$ , because  $h^1(s, \{ \text{samp}_1(B) \}) = 3 < 4 = h^1(s, \{ \text{samp}_2(B) \})$ . So we choose  $c_4 = \{ \text{samp}_2(B) \}$  and initialize the conflict to  $x_4 := c_4$ . To see whether  $\text{samp}_2(B)$  can be reached with an action sequence of length no longer than 4, and thus to determine whether we have to augment  $x_4$  by  $\text{samp}_1(B)$ , we continue with the recursion on the only achiever of  $c_4$ ,  $\text{drop}(\text{samp}_2, B)$ . This yields a recursive call on*

$$P_3 := \text{regress}(P_4, \text{drop}(\text{samp}_2, B)) = \{ \text{samp}_1(B), \text{rov}(B), \text{samp}_2(R) \}$$

*In the call to `PathCutRefine`( $P_3, 3$ ), there are two options for choosing the conjunction  $c_3$ :  $\{ \text{samp}_1(B) \}$  and  $\{ \text{samp}_2(R) \}$ . Say that we proceed along the top-level goal  $\text{samp}_2(B)$ , i.e., we choose  $c_3 = \{ \text{samp}_2(R) \}$ , and we set  $x_3 := c_3$ . The actions  $\text{collect}(\text{samp}_2, x)$  with  $x \neq A_2$  readily satisfy  $h^1(s, \text{pre}(\text{collect}(\text{samp}_2, x))) \geq$*

3, i.e., the respective recursive calls can terminate immediately by selecting an according fact from their preconditions.  $x_3$  is not extended. Consider  $\text{collect}(\text{samp}_2, A_2)$ . This yields a recursive call on

$$P_2 := \text{regress}(P_3, \text{collect}(\text{samp}_2, A_2)) = \{ \text{samp}_1(B), \text{rov}(B), \text{rov}(A_2), \text{samp}_2(B) \}$$

From the options in that recursive call, say we consider  $c_2 = \{ \text{rov}(A_2) \}$  and  $x_2 := c_2$ .  $c_2$  has two achievers:  $\text{move}(A_3, A_2, 1)$  and  $\text{move}(A_3, A_2, 2)$ . The latter can be trivially handled via its precondition  $\text{bat}(2)$ ,  $h^1(s, \{ \text{bat}(2) \}) = \infty$ . The former yields a recursive call on

$$P_1 := \text{regress}(P_2, \text{move}(A_3, A_2, 1)) = \{ \text{samp}_1(B), \text{rov}(B), \text{samp}_2(B), \text{rov}(A_3), \text{bat}(1) \}$$

Say that we finally select  $c_1 = \{ \text{rov}(A_3) \}$  and set  $x_1 := c_1$ . Again, the only supporting action, whose precondition does not already contain unreachable facts under  $h^1$ , is  $\text{move}(A_1, A_3, 1)$ .  $\text{PathCut}$  can stop here, as  $\text{move}(A_1, A_3, 1)$  deletes  $\text{bat}(1) \in P_1$ . Say we extend  $x_1$  with the deleted fact  $\text{bat}(1)$ ,  $x_1 = \{ \text{rov}(A_3), \text{bat}(1) \}$ .

The recursion now goes back up the recursion path over the levels  $i \in \{2, 3, 4\}$ , corresponding to the actions  $\text{move}(A_3, A_2, 1)$ ,  $\text{collect}(\text{samp}_2, A_2)$ ,  $\text{drop}(\text{samp}_2, B)$ , with the current conjunctions  $x_2 = \{ \text{rov}(A_2) \}$ ,  $x_3 = \{ \text{samp}_2(B) \}$ ,  $x_4 = \{ \text{samp}_2(B) \}$ . At each step, we extend  $x_i$  with  $x_{i-1}$  minus the respective action's precondition. At  $i = 2$ , both facts of  $x_1$  are contained in the precondition of  $\text{move}(A_3, A_2, 1)$ , so  $x_2$  remains the same. But then,  $x_3$  and  $x_4$  also remain the same in the remaining steps. Upon termination the only non-singleton conjunction in  $\mathcal{X}$  is  $\{ \text{rov}(A_3), \text{bat}(1) \}$ . Recall from Example 6.1 that this is exactly the single conjunction needed to obtain  $u^C(s_4) = \infty$ .

### 6.2.2. Neighbors Refinement

As already indicated, neighbors refinement assumes as input a set  $S$  of dead-end states that satisfies the  $u^C$  recognized-neighbors property:

**Definition 6.3** (Recognized-Neighbors Property). Suppose  $u$  is an unsolvability detector, and  $S \subseteq S^\Pi$  is a set of states. Denote by  $T = \text{Succ}(S) \setminus S$  the successors of the states from  $S$  that are not contained in  $S$  themselves. Then,  $S$  satisfies the  $u$  **recognized-neighbors property** if it holds  $u(t) = \infty$  for all  $t \in T$ .

Say that Algorithms 5.1 or 5.2 requests a refinement on  $\mathcal{R}[\hat{\Theta}](s) \setminus \text{conflicts}$ . Let  $S = \{ s' \in \mathcal{R}[\hat{\Theta}](s) \setminus \text{conflicts} \mid u(s') = 0 \}$ . Provided that  $u$  is the only unsolvability detector used, it is easy to see that  $S$  satisfies the  $u$  recognized-neighbors property. Namely, by construction,  $\mathcal{R}[\hat{\Theta}](s)$  contains only closed states. Every closed state was either expanded or is recognized as dead end by  $u$ . Since the states from  $S$  are closed and not recognized by  $u$ , they must have been expanded, so by construction,  $\mathcal{R}[\hat{\Theta}](s)$  contains all successors of  $S$ . But then, for all  $t \in \text{Succ}(S)$ , it holds that either  $t \in S$ ; or if  $t \notin S$  but  $t \in \mathcal{R}[\hat{\Theta}](s) \setminus \text{conflicts}$ , then  $u(t) = \infty$ , as per the selection of  $S$ ; or if  $t \in \text{conflicts}$ , then  $u(t) = \infty$  must be satisfied due to some prior refinement. In conclusion:

**Proposition 6.1.** For every refinement call in Algorithm 5.1 (line 26) and Algorithm 5.2 (line 29) on  $\mathcal{R}[\hat{\Theta}](s)$ , using  $u$  as the only unsolvability detector,  $S = \{ s' \in \mathcal{R}[\hat{\Theta}](s) \mid u(s') = 0 \}$  satisfies the  $u$  recognized-neighbors property.

For illustration, consider again the example search space from Figure 5.1b. At the refinement step on  $s_4$ , we have  $S = \{ s' \mid s' \in \mathcal{R}[\hat{\Theta}](s_4), u^C(s') = 0 \} = \{ s_4, s_1 \}$ . The neighbor states are  $T = \{ s_6, s_5 \}$ . The  $u^C$  recognized-neighbors property is satisfied: each of the neighbor states is already recognized by  $u^C$ , using the singleton conjunctions only, as there is no energy left.

Given a set of dead ends  $S$  that satisfy the  $u^C$  recognized-neighbors property, to refine  $C$ , we make use of what we refer to as the  $u^C$  neighbors information: the values  $u^C(t, c)$  for all recognized neighbors  $t \in T$  and  $c \in C$ . We compute this information once, at the start of the refinement procedure. Thanks to that information, in contrast to path-cut refinement, as well as all previous conjunction learning methods, we do not require any intermediate re-computation of  $u^C$  during the refinement. Instead, neighbors refinement uses the  $u^C$  neighbors information to directly pick suitable conjunctions  $x$  for the desired set  $\mathcal{X}$ . One can, in principle, even avoid the  $u^C$  computations prior to the refinement by caching the  $u^C$  neighbors information during search, whenever recognizing a dead end  $u^C(s) = \infty$ . But this turned out to be detrimental. Intuitively, as new conjunctions are continuously added to  $C$ , the cached  $u^C$  information is “outdated”. Using up-to-date  $C$  yields more effective learning. The construction of  $\mathcal{X}$  is inspired by the following simple characterizing condition for  $u^C$  dead-end recognition:

**Lemma 6.1.** *Let  $C$  be any set of atomic conjunctions, let  $s$  be a state, and let  $P \subseteq \mathcal{F}$ . Then,  $u^C(s, P) = \infty$  if and only if there exists  $c \in C$  such that:*

- (i)  $c \subseteq P$  and  $c \not\subseteq s$ ; and
- (ii) for every  $a \in \mathcal{A}[c]$ ,  $u^C(s, \text{regress}(c, a)) = \infty$ .

*Proof.* “ $\Rightarrow$ ”: By definition of  $u^C$ , there must be a conjunction  $c \in C$  so that  $c \subseteq P$  and  $u^C(s, c) = \infty$ . The latter directly implies that  $c \not\subseteq s$ , and that  $u^C(s, \text{regress}(c, a)) = \infty$  for every  $a \in \mathcal{A}[c]$ .

“ $\Leftarrow$ ”: As  $c \subseteq P$ , we have  $u^C(s, P) \geq u^C(s, c)$ . As  $c \not\subseteq s$ , we have

$$u^C(s, c) = \min_{a \in \mathcal{A}[c]} u^C(s, \text{regress}(c, a))$$

For every  $a \in \mathcal{A}[c]$ ,  $u^C(s, \text{regress}(c, a)) = \infty$ , so we have  $u^C(s, c) = \infty$  as desired.  $\square$

Suppose  $s$  is any state in  $S$ . We need to find  $\mathcal{X}$  so that  $u^{C \cup \mathcal{X}}(s) = u^{C \cup \mathcal{X}}(s, \mathcal{G}) = \infty$ . Given Lemma 6.1, we can do so by (i) picking some conjunction  $c$  with  $c \subseteq \mathcal{G}$  and  $c \not\subseteq s$ , and then, recursively in the same manner, (ii) picking for every possible supporter  $a \in \mathcal{A}[c]$  an unreachable conjunction  $c'$  for  $\text{regress}(c, a)$ . As  $s$  is a dead end, and as  $u^C$  recognizes all dead ends in the limit, Lemma 6.1 tells us that a suitable conjunction  $c$  exists at every recursion level. But the lemma does not tell us what that conjunction is. In particular,  $c$  must actually be unreachable, i.e., it must hold that  $h^*(s, c) = \infty$ . But, given  $s$  and any one conjunction  $c$ , this is the same as asking whether a plan for  $c$  exists, which is PSPACE-COMplete to decide.

Now, we already know that the states  $s \in S$  are dead ends. Therefore, we can in principle use the full subgoals as our conjunctions, i.e., in (i) we can use  $c = \mathcal{G}$  because we know that  $h^*(s, \mathcal{G}) = \infty$ , and in (ii) we can use  $c' = \text{regress}(c, a)$  because we know that  $h^*(s, \text{regress}(c, a)) = \infty$ . However, this naive solution is pointless. It effectively constructs a full regression search tree from  $\mathcal{G}$ , selecting conjunctions corresponding to the regressed search states. For the method to be practically useful, what we need to find are *small* unreachable subgoals. It turns out that one can exploit the  $u^C$  recognized-neighbors property to this end.

Algorithm 6.2 shows the pseudo-code of the neighbors refinement procedure. The purpose of a call to  $\text{NeighborsRefine}(P)$ , invoking the refinement on a target subgoal  $P$ , is to include conjunctions into  $\mathcal{X}$  making  $P$  unreachable from  $S$  under  $u^{C \cup \mathcal{X}}$ , i.e., so that  $u^{C \cup \mathcal{X}}(s, P) = \infty$  for all  $s \in S$ . For this to be possible, of course,  $P$  must be unreachable from  $S$ , i.e.,  $h^*(s, P) = \infty$  for every  $s \in S$ . That prerequisite is obviously true at the top-level call, where  $P = \mathcal{G}$ , because the states  $s \in S$  are dead ends. As we shall see below, the

**Algorithm 6.2:** Neighbors refinement.

---

**Input:** Set of conjunctions  $C$ ,  
Set of dead-end states  $S$ , with neighbors  $T = \text{Succ}(S) \setminus S$ , satisfying the  $\mathcal{U}^C$  recognized-neighbors property,  
 $\mathcal{U}^C$  neighbors information (the values of  $\mathcal{U}^C(t, c)$  for all  $t \in T$  and  $c \in C$ )

**Output:** Set of conjunctions  $\mathcal{X}$  such that  $\mathcal{U}^{C \cup \mathcal{X}}(s) = \infty$  for all  $s \in S$

---

```

1  $\mathcal{X} \leftarrow \emptyset$ ;
2 NeighborsRefine( $\mathcal{G}$ );
3 return  $\mathcal{X}$ ;

4 procedure NeighborsRefine( $P$ )
5    $x \leftarrow \text{Extract}(P)$ ;
6    $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$ ;
7   for  $a \in \mathcal{A}[x]$  where  $\exists s \in S$  s.t.  $\mathcal{U}^C(s, \text{regress}(x, a)) = 0$  do
8     if there is no  $x' \in \mathcal{X}$  s.t.  $x' \subseteq \text{regress}(x, a)$  then
9       NeighborsRefine( $\text{regress}(x, a)$ );

10 procedure Extract( $P$ )
11    $x \leftarrow P$ ;
12   while  $P \neq \emptyset$  do
13     let  $p \in P$ ;
14      $P \leftarrow P \setminus \{p\}$ ;  $x \leftarrow x \setminus \{p\}$ ;
15     /* Lemma 6.2 (i) */
16     if  $\exists t \in T. \forall c_t \in C, c_t \subseteq x: \mathcal{U}^C(t, c_t) = 0$  then
17        $x \leftarrow x \cup \{p\}$ ;
18       /* Lemma 6.2 (ii) */
19     else if  $\exists s \in S: x \subseteq s$  then
20        $x \leftarrow x \cup \{p\}$ ;
21   return  $x$ ;
```

---

prerequisite is invariant over calls to  $\text{Extract}(P)$ , i.e., the returned  $x$  also is unreachable from  $S$ . As, for an unreachable subgoal, all regressed subgoals also are unreachable, the prerequisite thus holds at every invocation of  $\text{NeighborsRefine}(P)$ .

But how to identify the conjunctions  $\mathcal{X}$ ? To this end,  $\text{NeighborsRefine}(P)$  closely resembles the structure of Lemma 6.1. Following Lemma 6.1 (i), it starts by calling  $\text{Extract}(P)$ , which identifies a subgoal  $x \subseteq P$  unreachable from  $S$ . Following Lemma 6.1 (ii), the procedure then finds conjunctions making  $x$ , and thus the target subgoal  $P$  which contains  $x$ , unreachable from  $S$  under  $\mathcal{U}^{C \cup \mathcal{X}}$ . To this end, the refinement is called recursively on the regressed subgoals  $\text{regress}(x, a)$  for the actions  $a$  supporting  $x$ .

More precisely, a recursive call is needed only for those supporters  $a$  not dealt with by the previous conjunctions  $C$ , i.e., those where, on some state  $s \in S$ , the regressed subgoal  $\text{regress}(x, a)$  is actually reachable under  $\mathcal{U}^C$ . Furthermore, such a supporting action has already been dealt with by the new conjunctions  $\mathcal{X}$  in case there is some  $x' \in \mathcal{X}$  s.t.  $x' \subseteq \text{regress}(x, a)$ . That is so because the conjunctions  $\mathcal{X}$  are constructed so that, upon termination, they are unreachable from  $S$  under  $\mathcal{U}^{C \cup \mathcal{X}}$ .

Consider now the  $\text{Extract}(P)$  sub-procedure, called on a target subgoal  $P$ . The procedure assumes that (a)  $P$  is unreachable from  $S$ . It attempts to find a small subgoal  $x \subseteq P$ , giving the guarantees that (b)  $x$  is still unreachable from  $S$ , and (c)  $x$  is unreachable from the neighbor states  $T$  under  $\mathcal{U}^C$ . To be precise, (c) means that, for every  $t \in T$ , there exists  $c_t \in C$  such that  $c_t \subseteq x$  and  $\mathcal{U}^C(t, c_t) = \infty$ . To guarantee this property, the sub-procedure relies on the  $\mathcal{U}^C$  neighbors information.

Ideally, we would like  $\text{Extract}(P)$  to compute the smallest possible such  $x \subseteq P$ , as smaller conjunctions tend to cause less work in the remainder of the refinement, and more importantly, have higher chances to be useful for recognizing dead ends other than the states in  $S$ . Unfortunately, computing the size-minimal  $x$  is in general infeasible, which one can show straightforwardly via a reduction from the minimal vertex cover problem, a well-known NP-COMPLETE decision problem (cf. Appendix B.2.2). Hence,  $\text{Extract}$  instead follows a simple greedy procedure, which starting from the entire subgoal  $x = P$ , iteratively removes facts from  $x$ , while guaranteeing that (b) and (c) remain satisfied. The resulting  $x$  is guaranteed to be minimal in the sense that no further facts can be removed without violating at least one of the two conditions.

To show that the  $x$  constructed by a call  $\text{Extract}(P)$  indeed satisfies (c), note that in each such call, there must exist a conjunction  $c_t \in C$ ,  $c_t \subseteq P$ , such that  $\mathcal{U}^C(t, c_t) = \infty$ , for every neighbor  $t \in T$ . Namely, for the top-level goal  $P = \mathcal{G}$ , as per the recognized-neighbors property, we have that  $\mathcal{U}^C(t, \mathcal{G}) = \infty$ , so by the definition of  $\mathcal{U}^C$  there exists  $c_t \subseteq \mathcal{G}$  with  $\mathcal{U}^C(t, c_t) = \infty$ . For later invocations of  $\text{Extract}(P)$ , we have that  $P = \text{regress}(x, a)$ , where  $x$  was constructed by a previous invocation of  $\text{Extract}(P)$ . By property (c) of that previous invocation, there exists  $c'_t \in C$  such that  $c'_t \subseteq x$  and  $\mathcal{U}^C(t, c'_t) = \infty$ . But then, in particular, we have that  $\mathcal{U}^C(t, x) = \infty$ . Given this, we must also have that  $\mathcal{U}^C(t, \text{regress}(x, a)) = \infty$ , i.e.,  $\mathcal{U}^C(t, P) = \infty$ . Hence, at the entry of  $\text{Extract}(P)$ ,  $x = P$ , (c) is satisfied. It remains satisfied throughout as per the first **if** statement of the minimization loop.

In order to show property (b), note that  $x$  is not contained in any state  $s \in S$ . Again, this holds at the beginning,  $x = P$ , because by assumption (a),  $P$  is not reachable from any of these  $s$ . So, in particular,  $P \not\subseteq s$ .  $x \not\subseteq s$  remains satisfied as per the second **if** check of the minimization loop. Property (b) follows via the follow lemma:

**Lemma 6.2.** *Let  $C$  be any set of atomic conjunctions. Let  $S$  be a set of dead-end states, and let  $T = \text{Succ}(S) \setminus S$  be its neighbors. Let  $x \subseteq \mathcal{F}$ . If*

- (i) *for every  $t \in T$ , there exists  $c_t \in C$  such that  $c_t \subseteq x$  and  $\mathcal{U}^C(t, c_t) = \infty$ ; and*
- (ii) *for every  $s \in S$ ,  $x \not\subseteq s$ ;*

*then  $h^*(s, x) = \infty$  for every  $s \in S$ .*

*Proof.* Assume for contradiction that there is a state  $s \in S$  where  $h^*(s, x) < \infty$ . Then, there exists a path  $s = s_0, s_1, \dots, s_n$  from  $s$  to some state  $s_n$  with  $x \subseteq s_n$ . Let  $i$  be the largest index such that  $s_i \in S$ . Such  $i$  exists because  $s_0 = s \in S$ , and  $i < n$  because otherwise we get a contradiction to (ii). But then,  $s_{i+1} \notin S$ , and thus  $s_{i+1} \in T$  by definition. By (i), there exists  $c_{s_{i+1}} \subseteq x$  such that  $\mathcal{U}^C(s_{i+1}, c_{s_{i+1}}) = \infty$ . This implies that  $h^*(s_{i+1}, c_{s_{i+1}}) = \infty$ , which implies that  $h^*(s_{i+1}, x) = \infty$ . The latter is in contradiction to the selection of the path. The claim follows.  $\square$

Altogether, Algorithm 6.2 is correct in the following sense:

**Theorem 6.2.** *Let  $C$  be any set of atomic conjunctions. Let  $S$  be a set of dead-end states, and let  $T = \text{Succ}(S) \setminus S$  be its neighbors with the  $\mathcal{U}^C$  recognized-neighbors property. Then:*

- (i) The execution of  $\text{NeighborsRefine}(\mathcal{G})$  terminates.
- (ii) Upon termination of  $\text{NeighborsRefine}(\mathcal{G})$ ,  $u^{C \cup X}(s) = \infty$  for every  $s \in S$ .

*Proof.* (i) holds because every recursive call adds a new conjunction  $x \notin X$ : before the recursive call to  $\text{NeighborsRefine}(\text{regress}(x, a))$  in the top-level procedure, there is no  $x' \in X$  s.t.  $x' \subseteq \text{regress}(x, a)$ ; but that condition holds for the  $x'$  constructed in that recursive call. The number of possible conjunctions is finite, so the recursion must eventually terminate.

Finally, (ii) is a corollary of the aforementioned property that, upon termination, every  $x \in X$  is unreachable from  $S$  under  $u^{C \cup X}$ . Without loss of generality assume unit action costs. To see that the latter property holds, assume to the contrary that  $u^{C \cup X}(s, x) = 0$ . Then  $h^{C \cup X}(s, x) = N$  for some finite  $N$ . Let  $a$  be a best achiever of  $x$  under  $h^{C \cup X}$ , i.e.,  $h^{C \cup X}(s, \text{regress}(x, a)) = N - 1$ . By construction, in the recursive call that included  $x$  into  $X$ , either an  $x' \subseteq \text{regress}(x, a)$  was already present in  $X$ , or such an  $x'$  was included in the recursive call on  $\text{regress}(x, a)$ . But then,  $h^{C \cup X}(s, x') \leq N - 1$ . Iterating this argument, we obtain a conjunction  $x_0 \in X$  where  $h^{C \cup X}(s, x_0) = 0$ , i.e.,  $x_0 \subseteq s$ . Such  $x_0$  are never included into  $X$  by construction, in contradiction, concluding the argument. □

**Example 6.3.** Consider once again Example 5.1. Recall that  $u^1(s_5) = u^1(s_6) = \infty$ , i.e., given the search conflict  $\mathcal{R}[\hat{\Theta}](s_4) = \{s_4, s_6, s_1, s_5\}$ , the set  $S = \{s \in \mathcal{R}[\hat{\Theta}](s_4) \mid u^1(s) = 0\} = \{s_4, s_1\}$  satisfies the  $u^1$  recognized-neighbors property, with neighbors  $T = \{s_5, s_6\}$ . Consider the neighbors refinement for  $S$  and  $u^C$ , starting with the set of singleton conjunctions  $C$ .

We initialize  $X = \emptyset$  and call  $\text{NeighborsRefine}(\{\text{samp}_1(B), \text{samp}_2(B)\})$ . Consider the corresponding call  $\text{Extract}(\{\text{samp}_1(B), \text{samp}_2(B)\})$ . Here, we find that  $x = \{\text{samp}_1(B)\}$  is suitable for each of  $s_5$  and  $s_6$ : it is unreachable under  $u^1$  because in both states there is no energy left. Furthermore,  $x = \{\text{samp}_1(B)\}$  is neither contained in  $s_4$  nor in  $s_1$ . So we return  $x$ . Note that this is not a new conjunction; it is already contained in the current  $C$ . The  $x$  extracted is guaranteed to not already be in  $X$ , but it may be an element of  $C$ . In other words, as the  $x$  in each recursive call, we may use a conjunction that was already atomic beforehand.

Back in the  $\text{NeighborsRefine}$  call, we see that  $x$  can be achieved (only) by dropping  $\text{samp}_1$  at  $B$ . We handle the regressed subgoal via the call  $\text{NeighborsRefine}(\{\text{rov}(B), \text{samp}_1(R)\})$ . Here, the extraction sub-procedure may choose  $x = \{\text{rov}(B)\}$ , which is suitable for the same reasons as above.  $x$  has two achievers  $\text{move}(A_3, B, 2)$  and  $\text{move}(A_3, B, 1)$ . The regression of  $x$  over the former contains  $\text{bat}(2)$ , which is already detected unreachable from  $s_4$  and  $s_1$  by  $u^1$ . The latter is handled via  $\text{NeighborsRefine}(\{\text{rov}(A_3), \text{bat}(1)\})$ .

Consider finally the extraction sub-procedure in that recursive call. For the neighbor states  $s_5$  and  $s_6$ , where the rover is at  $A_3$  but the battery is empty,  $\text{bat}(1)$  needs to stay in  $x$ . However,  $\text{bat}(1)$  is contained in the states  $s_4$  and  $s_1$ , i.e.,  $\text{rov}(A_3)$  also needs to stay in  $x$ . So we end up with  $x = P = \{\text{rov}(A_3), \text{bat}(1)\}$ . Observe that this last  $x$  is the first “new” conjunction extracted.

The refinement process stops here, because the actions achieving  $x$ ,  $\text{move}(z, A_3, 2)$ , all contain  $\text{bat}(2)$  as precondition, for which  $u^1(s_4, \{\text{bat}(2)\}) = u^1(s_1, \{\text{bat}(2)\}) = \infty$  is already satisfied. Hence the set  $X$  returned contains, like for path-cut refinement above, just the one new conjunction  $\{\text{rov}(A_3), \text{bat}(1)\}$ , which is exactly what is needed for  $u^{C \cup X}(s_4) = u^{C \cup X}(s_1) = \infty$ .

### 6.2.3. Learning Effectiveness: Worst-Case Analysis

Unsurprisingly, there are tasks  $\Pi$  and dead ends  $\hat{s}$  where  $C$  has to contain exponentially many conjunctions in the size of  $\Pi$  so that  $\mathcal{U}^C(\hat{s}) = \infty$ . In other words,  $\mathcal{U}^C$  refinement (no matter of the exact algorithm) on an arbitrary dead-end state must in the worst case generate an exponential, in  $|\Pi|$ , number of conjunctions. This worst-case result directly applies to search-conflict refinements, if search uses an additional  $\mathcal{U}$  for dead-end detection; one can then simply choose  $\mathcal{U}$  in a way such that the first requested  $\mathcal{U}^C$  refinement will have to be on such a worst-case state  $\hat{s}$ . But what if  $\mathcal{U}^C$  is the only dead-end detector used? Refinements can then no longer be requested by search on arbitrary dead ends. In particular, the conflicts identified by search necessarily satisfy the  $\mathcal{U}^C$  recognized-neighbors property. This intuitively makes the refinements of  $\mathcal{U}^C$  become more incremental; when search identifies a conflict  $\mathcal{R}[\hat{\Theta}](s)$ , the corresponding  $\mathcal{U}^C$  refinement must only close the gap between  $\mathcal{R}[\hat{\Theta}](s)$  and  $\mathcal{R}[\hat{\Theta}](s)$ 's recognized neighbors. Moreover, the conflict component  $\mathcal{R}[\hat{\Theta}](s)$  may itself contain exponentially many states, relativizing the need of an exponential  $C$ .

This raises the question of how many conjunctions need to be generated in the worst case, if one assumes dead-end sets  $S$  as input where (1)  $S$  satisfies the recognized-neighbors property, and (2) the input size  $|S|$  is polynomially bounded in the task's size. Unfortunately, as the following example shows, there are indeed pathological cases, where exponentially many conjunctions are necessary in order to expand the dead-end detection capabilities of  $\mathcal{U}^C$  even by just a single transition step:

**Proposition 6.2.** *There are planning tasks  $\Pi$ , and dead-end states  $s \in \mathcal{S}^\Pi$  such that  $\mathcal{U}^C(s) = \infty$  entails that  $C$  contains exponentially many conjunctions in  $|\Pi|$ , even if restricting the states  $s$  to ones that satisfy the  $\mathcal{U}^{C'}$  recognized-neighbor property for some polynomially bounded  $C'$ .*

*Proof sketch.* Such an example is given by a task with initial state  $\mathcal{I} = \{p_1, \dots, p_n\}$  and goal  $\mathcal{G} = \{q_1, \dots, q_n\}$ . The action achieving  $q_i$  requires  $p_i$  and an additional fact  $r$  as precondition, and deletes  $p_i$ .  $r$  can only be achieved by deleting one of the  $p_i$  facts. The task is unsolvable because one cannot achieve all  $q_i$  facts simultaneously. The initial state satisfies the  $\mathcal{U}^1$  recognized-neighbors property, because initially it is only possible to trade in some  $p_i$  for  $r$ , preventing to reach  $q_i$  afterwards even under  $\mathcal{U}^1$ 's relaxing assumptions. However, in order that  $\mathcal{U}^C(\mathcal{I}) = \infty$ ,  $C$  needs to enumerate all combinations  $\{X_i \mid 1 \leq i \leq n, X_i \in \{p_i, q_i\}\}$ . Note that each strict subset is reachable from  $\mathcal{I}$ . If a single one of these combinations was missing in  $C$ , then one can construct a goal regression trace that splits in  $\mathcal{U}^C$  into individually reachable parts.  $\square$

The detailed example is provided in Appendix B.2.3.

### 6.3. Critical-Path NoGoods

As previously discussed, the computation of  $\mathcal{U}^C$  is low-order polynomial time in the number  $|C|$  of atomic conjunctions. Yet, in practice, it may incur a substantial runtime overhead as  $C$  is growing. In the following, we show how to alleviate this overhead through *critical-path NoGoods*, i.e., sufficient conditions to  $\mathcal{U}^C(s) = \infty$  that are easier to evaluate than  $\mathcal{U}^C$  itself.

In Section 6.3.1, we first observe that it is actually possible to characterize the dead ends recognized by  $\mathcal{U}^C$  exactly through a NoGood formula, specifically  $\Phi^{C^*}$  s.t.  $s \not\models \Phi^{C^*}$  if and only if  $\mathcal{U}^C(s) = \infty$ . In principle,

once constructed,  $\Phi^{C*}$  can thus substitute  $\mathcal{U}^C$  entirely. This is however mainly of theoretical interest as  $\Phi^{C*}$  has size worst-case exponential in  $|C|$ , a blow-up which tends to happen as our experiments show.

In Section 6.3.2, we introduce practical variants, learning weaker formulae  $\varphi$ ,  $\Phi^{C*} \Rightarrow \varphi$ , during search. Given that  $s \not\models \varphi$  then implies that  $s \not\models \Phi^{C*}$ , and therewith  $\mathcal{U}^C(s) = \infty$ , these formulae still allow to avoid some – yet not *all* –  $\mathcal{U}^C$  evaluations. Whenever  $\mathcal{U}^C$  is evaluated on a state  $s$  such that  $s \models \varphi$  but  $\mathcal{U}^C(s) = \infty$ , we refine  $\varphi$  so that  $s \not\models \varphi$  holds afterwards. Future states  $s'$  other than  $s$  with  $s' \not\models \varphi$  then no longer need to be evaluated. We consider two approaches to generate such  $\varphi$ : *clause learning*, inspired by prior work, and *CART learning*, leveraging the perfect NoGood formula construction.

In order to separate concerns, in Section 6.3.3, we provide a brief experimental evaluation, studying the feasibility of the  $\Phi^{C*}$  construction, and comparing the two NoGood learning variants. Our main experiments, evaluating search with conflict-driven learning via  $\mathcal{U}^C$ , are given in Section 6.4.

### 6.3.1. Equivalent $\mathcal{U}^C$ NoGood Characterization

We have identified two alternative ways of constructing such a  $\Phi^{C*}$ , which differ in terms of formula structure and size. We detail in what follows the simpler formula, which turns out to be more useful in practice; we discuss the alternative at the end of this section.

Our construction is based on what we call *atomic regression traces*. These are sets of atomic conjunctions, gleaned from the recursion in Equation (6.1) by selecting a single  $c$  in the maximization (bottom case) and following all  $a$  in the minimization (middle case). This yields a disjunction of atomic conjunctions. We observe that every state along a path to the goal must satisfy every such disjunction, and that the states not satisfying at least one such a disjunction are exactly the dead-end states recognized by  $\mathcal{U}^C$ .

**Definition 6.4** (*C-Atomic Regression Trace*). *Let  $C$  be a set of conjunctions. A  $C$ -atomic regression trace, short *CART*, is a subset  $\sigma \subseteq C$  so that*

- (i) *there exists  $c \in \sigma$  with  $c \subseteq \mathcal{G}$ , and*
- (ii) *for every  $c \in \sigma$  and for every  $a \in \mathcal{A}[c]$ , there exists  $c' \in \sigma$  with  $c' \subseteq \text{regress}(c, a)$ .*

We identify a *CART*  $\sigma$  with the disjunction  $\bigvee_{c \in \sigma} c$  of its element conjunctions. Note that each *CART* is, hence, a DNF formula.

To see how conditions (i) and (ii) capture why  $\mathcal{U}^C$  evaluates to  $\infty$ , consider a state  $s$  and an atomic conjunction  $c \in C$ . In the simplest case,  $\mathcal{U}^C(s, c)$  is  $\infty$  because  $c$  is not true in  $s$ , and there is no action that achieves  $c$ , i.e., if  $c \not\subseteq s$  and  $\mathcal{A}[c] = \emptyset$ . If  $c \subseteq \mathcal{G}$ , then  $\{c\}$  is a *CART* on its own: if a state does not satisfy  $c$ , there is no way in which  $c$  (and hence the goal) can be made true from that state. In contrast, for an atomic conjunction  $c$  with non-empty set of achievers,  $\mathcal{U}^C(s, c) = \infty$  depends on the reachability of other atomic conjunctions. For  $\mathcal{U}^C(s, c) = \infty$  to be true, the regression of  $c$  over each of its achievers must contain an atomic conjunction  $c'$  with  $\mathcal{U}^C(s, c') = \infty$ . Condition (ii) ensures that a *CART*  $\sigma$  covers all achievers. But then, as  $\mathcal{U}^C(s, c) < \infty$  only if  $s$  supports at least one achiever,  $\mathcal{U}^C(s, c) < \infty$  entails that  $s$  satisfies  $\sigma$ :

**Lemma 6.3.** *Let  $C \subseteq 2^{\mathcal{F}}$  be a set of conjunctions, and let  $\sigma$  be a  $C$ -atomic regression trace. For every state  $s$ , if  $s \not\models \sigma$ , then  $\mathcal{U}^C(s, c) = \infty$  for every  $c \in \sigma$ .*

*Proof.* Assume that  $\mathcal{U}^C(s, c) < \infty$  for some  $c \in \sigma$ . Assume, w.l.o.g., unit cost (action costs are irrelevant for  $\mathcal{U}^C$ ). Let  $c_0 \in \sigma$  be an element of  $\sigma$  with minimal  $h^C(s, c_0)$  value. In particular,  $h^C(s, c_0) < \infty$ . If  $c_0 \subseteq s$ ,

then  $s \models \sigma$  and there is nothing to prove. Assume for contradiction that  $c_0 \not\subseteq s$ . Consider the action  $a$  from the second case of Equation (6.1), in the computation of  $h^C(s, c_0)$ . By the definition of  $\sigma$ , there exists  $c' \in \sigma$  with  $c' \subseteq \text{regress}(c_0, a)$ . But then,  $\infty > h^C(s, c_0) \geq h^C(s, \text{regress}(c_0, a)) + 1 \geq h^C(s, c') + 1$ , i.e.,  $h^C(s, c_0) > h^C(s, c')$ , which contradicts the selection of  $c_0$ .  $\square$

By Definition 6.4 (i), each CART contains at least one goal conjunction. Hence, by Lemma 6.3, the existence of an unsatisfied CART implies that  $u^C(s) = \infty$ . We next show that the opposite direction also holds:

**Lemma 6.4.** *For every state  $s$ , if  $u^C(s) = \infty$ , then there exists a  $C$ -atomic regression trace  $\sigma$  so that  $s \not\models \sigma$ .*

*Proof.* We construct  $\sigma$  through a simple recursive procedure. Initially, let  $\sigma = \emptyset$ . Starting with  $\mathcal{G}$ , there must exist  $c \in C$  so that  $u^C(s, c) = \infty$  and  $c \subseteq \mathcal{G}$ . Since  $u^C(s, c) = \infty$ , we must have  $c \not\subseteq s$ , and for every  $a \in \mathcal{A}[c]$ , we must have  $u^C(s, \text{regress}(c, a)) = \infty$ . We add  $c$  to  $\sigma$ , and we repeat on  $\text{regress}(c, a)$  for all  $a \in \mathcal{A}[c]$ . We terminate the recursion when the selected  $c$  is already contained in  $\sigma$  (which happens at the latest when  $\sigma = C$ ). The resulting  $\sigma$  satisfies Definition 6.4, and  $s \not\models \sigma$  by construction.  $\square$

**Example 6.4.** *Reconsider the state  $s_6$  from Example 6.1 and recall that  $u^1(s_6) = \infty$ . We construct a CART which is violated by  $s_6$ . Starting with the goal, we have the choice between  $\{ \text{samp}_1(B) \}$  and  $\{ \text{samp}_2(B) \}$ , both being unreachable from  $s_6$  under  $u^1$ . Suppose with choose*

$$\sigma := \{ \{ \text{samp}_1(B) \} \}$$

*The only achiever of the selected atomic conjunction is  $\text{drop}(\text{samp}_1, B)$ , which has precondition  $\text{rov}(B)$  and  $\text{samp}_1(R)$ . The latter fact is contained in  $s_6$ , forcing us to continue with the former*

$$\sigma := \{ \{ \text{samp}_1(B) \}, \{ \text{rov}(B) \} \}$$

*This conjunction now has two achievers:  $\text{move}(A3, B, 2)$  and  $\text{move}(A3, B, 1)$ . Since the rover is at  $A_3$  in  $s_6$ , the only possibility to cover their preconditions is to include both  $\text{bat}(1)$  and  $\text{bat}(2)$ , yielding:*

$$\sigma := \{ \{ \text{samp}_1(B) \}, \{ \text{rov}(B) \}, \{ \text{bat}(1) \}, \{ \text{bat}(2) \} \}$$

*At this point, we can terminate, because  $\text{bat}(2)$  has no achievers, and every achiever of  $\text{bat}(1)$  requires  $\text{bat}(2)$  in its precondition. The constructed CART is equivalent to the following fact disjunction*

$$\text{samp}_1(B) \vee \text{rov}(B) \vee \text{bat}(1) \vee \text{bat}(2)$$

*covering all states with no energy left, and where neither rover nor sample  $\text{samp}_1$  is at  $B$ .*

Lemma 6.3 and Lemma 6.4 together give the desired exact characterization of  $u^C$  dead-end detection:

**Theorem 6.3.** *Let  $\Sigma^C$  be the set of all  $C$ -atomic regression traces. Define  $\Phi^{C*} = \bigwedge_{\sigma \in \Sigma^C} \sigma$ . Then, for every state  $s$ ,  $s \not\models \Phi^{C*}$  if and only if  $u^C(s) = \infty$ .*

The construction of  $\Phi^{C*}$  requires the enumeration of all CARTs. Obviously, the number of CARTs is worst-case exponential in the number of atomic conjunctions  $|C|$ . This blow-up can be alleviated to some extent by identifying and avoiding redundancies, during the enumeration of  $\Sigma^C$ , and by leveraging *mutex* information to reduce the size of  $\Phi^{C*}$  post generation:

**Subsumption Pruning** Consider two CARTs  $\sigma \neq \sigma'$  s.t.  $\sigma'$  *subsumes*  $\sigma$ , i.e.,  $\sigma' \subset \sigma$ . Notice that removing  $\sigma$  from  $\Phi^{C*}$  results in an equisatisfiable formula. Namely, denote by  $\Phi^{C*} \setminus \{\sigma\}$  the reduced formula. Clearly, if  $\Phi^{C*}$  is satisfied, then so is  $\Phi^{C*} \setminus \{\sigma\}$ . On the other hand, if  $s \models \Phi^{C*} \setminus \{\sigma\}$ , then it holds in particular that  $s \models \sigma'$ , which as per the subsumption relation implies that  $s \models \sigma$ , so  $s \models \Phi^{C*}$ .

Instead of naively removing subsumed CARTs after  $\Phi^{C*}$  was already fully generated, one can identify, during the generation process, CARTs that *will* be subsumed. Organizing the generation process as a tree search where tree leaves are CARTs, we do so via checking, prior to expanding an atomic subgoal  $c$ , whether  $c$  was explored on a previous search path already. If so, and if the prefix before  $c$  on that previous search path subsumes our current prefix, then the re-expansion of  $c$  is pruned. This discards many CARTs long before they are generated.

**Mutex Reasoning** For the purpose of dead-end detection in forward search it is enough if  $s \models \Phi^{C*} \Leftrightarrow \mathcal{U}^C(s) = \infty$  for those  $s$  that are actually reachable from the initial state. In particular, we may disregard states that violate *mutex* constraints: we say that two atomic conjunctions  $c_1$  and  $c_2$  are *mutually exclusive* if it holds for all reachable states  $s \in \mathcal{R}(s_I)$  that  $c_1 \not\subseteq s$  or  $c_2 \not\subseteq s$ . Now, consider two elements  $\sigma_1$  and  $\sigma_2$  of  $\Phi^{C*}$ . Let  $c \in \sigma_1 \setminus \sigma_2$ . Notice that we can remove  $c$  from  $\sigma_1$  if  $c$  is mutually exclusive with all  $c' \in \sigma_2$ . To see this, let  $s$  be any reachable state. If  $s$  satisfies  $(\sigma_1 \setminus \{c\}) \wedge \sigma_2$ , then  $s$  clearly also satisfies  $\sigma_1 \wedge \sigma_2$ . Say that  $s$  satisfies  $\sigma_1 \wedge \sigma_2$ . Since  $s \models \sigma_2$ , there exists some  $c' \in \sigma_2$  s.t.  $c' \subseteq s$ . Since  $c$  is mutually exclusive with  $c'$ ,  $c \not\subseteq s$ . But  $s \models \sigma_1$ , so there must exist some  $c'' \in \sigma_1$ ,  $c'' \neq c$  s.t.  $c'' \subseteq s$ . Hence,  $s \models (\sigma_1 \setminus \{c\}) \wedge \sigma_2$ .

As advertised, we have also explored an alternative way to build an exact offline NoGood formula. So far, when gleaning our regression traces from the recursion in Equation (6.1), we chose to select only a single  $c$  in the bottom-case maximization. The alternative is to select all these  $c$ , recursively constructing an AND/OR tree, and therewith a corresponding formula, over the atomic conjunctions. That formula is equivalent to  $\Phi^{C*}$  as defined above, and it may be exponentially smaller. However, this alternative suffers from its complex formula structure, with arbitrarily deep nesting of conjunction and disjunction. This has two major practical implications. First, the AND/OR tree construction is not amenable to our subsumption pruning, and in our experiments was much less feasible than constructing  $\Phi^{C*}$ . Secondly, the simple structure of  $\Phi^{C*}$  lends itself to constructing that formula incrementally, enabling online NoGood learning, as we discuss next.

### 6.3.2. NoGood Learning

We first discuss the clause learning approach, and then show how to leverage the  $\Phi^{C*}$  construction for an effective online NoGood learning variant.

#### Clause Learning

The clause learning method is technically quite simple. It follows earlier proposals (Kolobov et al., 2012b; Muise et al., 2012a). Consider any state  $s$  where  $\mathcal{U}^C(s) = \infty$ . Denote by

$$\psi(s) = \bigvee_{p \in \mathcal{F} \setminus s} p$$

the disjunction of facts false in  $s$ . Then,  $\psi(s)$  is a *valid clause*: for any state  $s'$ , if  $s'$  does not satisfy  $\psi(s)$ , written  $s' \not\models \psi(s)$ , then  $\mathcal{U}^C(s') = \infty$ ; in particular, if  $s'$  does not satisfy  $\psi(s)$  then it is a dead end. To see

this, just note that, as  $u^C(s) = \infty$ , the goal is unreachable from  $s$  under  $u^C$ . To make the goal reachable under  $u^C$ , we need to make true at least one of the facts that are false in  $s$ .

The clause  $\psi(s)$  just defined is, per se, not useful as it generalizes to only those states subsumed by  $s$ , i.e., whose true facts are contained in those of  $s$ . This changes when *minimizing*  $\psi(s)$ , testing whether individual facts  $p$  can be removed while preserving validity. In other words, we aim at obtaining a *minimal reason* for  $u^C(s) = \infty$ . Our minimization method is straightforward, testing the facts  $p \in \mathcal{F} \setminus s$  one by one.

We start with  $s' := s$ . We then loop over all  $p \in \mathcal{F} \setminus s$ . In each loop iteration, we test whether  $u^C(s' \cup \{p\}) = \infty$ ; if so, we set  $s' := s' \cup \{p\}$ . Upon termination of the loop,  $s'$  is a set-inclusion maximal superset of  $s$  that preserves goal unreachability under  $u^C$ , i.e., where  $u^C(s') = \infty$ . We then obtain our clause as the disjunction of the remaining facts,  $\psi(s')$ .

**Example 6.5.** Consider one final time  $s_6$  from Example 6.1. Suppose we execute clause learning on  $u^1$ .

The minimization loop starts with

$$s' := s_6 = \{ \text{rov}(A3), \text{bat}(0), \text{samp}_1(R), \text{samp}_2(A2) \}$$

Adding the other possible locations of  $\text{samp}_2$ , i.e., the facts  $\text{samp}_2(x)$  for all  $x \in \{A_1, A_3, B, R\}$ , the goal is still unreachable under  $u^1$  because we cannot achieve the goal  $\text{samp}_1(B)$ . So we set

$$s' := s' \cup \{ \text{samp}_2(B), \text{samp}_2(A1), \text{samp}_2(A3), \text{samp}_2(R) \}$$

Considering now the other possible locations of  $\text{samp}_1$ , i.e., the facts  $\text{samp}_1(x)$  for  $x \in \{B, A_1, A_2, A_3\}$ , the  $\text{samp}_1(A_i)$  facts can be added, as we still cannot reach  $\text{samp}_1(B)$ . But  $\text{samp}_1(B)$  cannot be added as we would then have  $\mathcal{G} \subseteq s'$ . So, we set

$$s' := s' \cup \{ \text{samp}_1(A1), \text{samp}_1(A2), \text{samp}_1(A3) \}$$

We cannot add any amount of energy,  $\text{bat}(1)$  or  $\text{bat}(2)$ , which would make the goal become reachable under  $u^1$ . Finally, considering the other possible locations of the rover, i.e., the facts  $\text{rov}(x)$  for  $x \in \{B, A_1, A_2\}$ , we can add  $\text{rov}(A1)$  and  $\text{rov}(A2)$ , as  $\text{rov}(B)$  then still remains unreachable. But we cannot add  $\text{rov}(B)$  as, then,  $\text{samp}_1$  could be dropped at  $B$  without any energy consumption.

We end up with

$$\begin{aligned} s' = & \{ \text{rov}(x) \mid x \in \{A_1, A_2, A_3\} \} \\ & \cup \{ \text{bat}(0) \} \\ & \cup \{ \text{samp}_1(x) \mid x \in \{A_1, A_2, A_3, R\} \} \\ & \cup \{ \text{samp}_2(x) \mid x \in \{B, A_1, A_2, A_3, R\} \} \end{aligned}$$

This yields the clause:

$$\psi(s') = \text{rov}(B) \vee \text{bat}(1) \vee \text{bat}(2) \vee \text{samp}_1(B)$$

Note that the clause we end up with depends on the ordering of facts in the minimization loop. If, for example, we test  $\text{rov}(B)$  at the very beginning of the loop, then it can be added: with  $\text{samp}_2$  being only at  $A_2$ , the rover still needs to move to  $A_2$  to collect  $\text{samp}_2$ , which requires energy. On the other hand, if we do add  $\text{rov}(B)$  into  $s'$ , then later on we cannot add  $\text{samp}_2(A_3)$ . We use an arbitrary fact ordering in our implementation, i.e., we do not attempt to find clever orderings.

The re-computation of  $\mathcal{U}^C(s' \cup \{p\})$  for each fact  $p$  in the minimization loop can be optimized by doing it in an incremental manner. Omitting implementation details, we essentially store the dynamic programming table (an index from atomic conjunctions into  $\{0, \infty\}$ ) of the previous iteration, identify the table cells changing from  $\infty$  to 0 due to the inclusion of  $p$ , and propagate these changes. Nevertheless, the minimization sometimes incurs a significant computational overhead. To reduce that overhead, in our implementation we slightly diverge from the above. Our implementation being based on Fast Downward (Helmert, 2006), we leverage the internal FDR planning task representation by processing in one minimization step entire variables  $v$  at once, replacing  $p$  in the above by the set of all facts  $\{v \mapsto d \mid d \in \mathcal{D}_v\}$ . This yields weaker clauses, but takes less runtime.

During search, we maintain a conjunction  $\Psi^{\text{CL}}$  of clauses, starting with the empty  $\Psi^{\text{CL}} := \top$ . We consult  $\Psi^{\text{CL}}$  prior to every  $\mathcal{U}^C(s)$  evaluation, skipping the computation if  $s \not\models \Psi^{\text{CL}}$ . When we encounter during search a state  $s \models \Psi^{\text{CL}}$ , where  $\mathcal{U}^C(s) = \infty$ , we compute a clause  $\psi(s')$  as just sketched, and update  $\Psi^{\text{CL}} := \Psi^{\text{CL}} \wedge \psi(s')$ .

### CART NoGood Learning

Turning the approach from Section 6.3.1 into an incremental NoGood learning approach is straightforward. We initialize  $\Phi^{\text{ART}} = \top$ . When we encounter a state  $s \models \Phi^{\text{ART}}$  where  $\mathcal{U}^C(s) = \infty$ , we extend  $\Phi^{\text{ART}}$  by a new CART  $\sigma$ , setting  $\Phi^{\text{ART}} := \Phi^{\text{ART}} \wedge \sigma$ , guaranteeing that  $s \not\models \Phi^{\text{ART}}$  holds afterwards. As per Lemma 6.4, this is always possible. Moreover, its proof is constructive showing how to find the desired  $\sigma$ .

Observe that, at any point in time,  $\Phi^{\text{ART}}$  is a sub-conjunction of  $\Phi^{C*}$ , consisting of CARTs representing the  $\mathcal{U}^C$  dead ends encountered so far. Observe, furthermore, that CART learning is inspective, based on an analysis of the *reasons* for  $\mathcal{U}^C(s) = \infty$ , in contrast to clause learning which treats  $\mathcal{U}^C$  like a blackbox. One advantage of this is that CART learning does not involve any intermediate reevaluation of  $\mathcal{U}^C$ . On the other hand,  $\Phi^{\text{ART}}$  has a more complex structure, a conjunction of DNF formulae, versus the CNF structure of  $\Psi^{\text{CL}}$ . So,  $\Phi^{\text{ART}}$  can be more expensive to evaluate. In our implementation, we simplify the evaluation of  $\Phi^{\text{ART}}$  by representing the atomic conjunctions in  $\Phi^{\text{ART}}$  via auxiliary propositions. Before evaluating  $\Phi^{\text{ART}}$ , we compute the satisfied atomic conjunctions once, collecting the corresponding propositions. The evaluation of  $\Phi^{\text{ART}}$  itself then boils down to checking a much simpler CNF formula.

### 6.3.3. Experimental Evaluation

We implemented the different  $\mathcal{U}^C$  NoGood variants in FAST DOWNWARD (Helmert, 2006).<sup>1</sup> Given that by design, they primarily affect  $\mathcal{U}^C$  computations on recognized dead ends, we focus on proving unsolvability where such states naturally abound. We use the benchmarks from the unsolvability planning competition (UIPC'16). In addition, we consider *resource-constrained planning (RCP)* (e.g., Haslum and Geffner, 2001; Nakhost et al., 2012), where the goal must be achieved subject to a limited resource budget. We use the benchmarks by Nakhost et al. (2012), which are *controlled* in that the minimum required budget  $\mathbf{b}_{\min}$  is known, and the actual budget is set to  $\mathfrak{C} * \mathbf{b}_{\min}$ , where the *constrainedness level*  $\mathfrak{C}$  is a benchmark instance parameter. We follow Hoffmann et al. (2014), and use  $\mathfrak{C} \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$ , so that the tasks are unsolvable. All experiments were run on a cluster of Intel Xeon E5-2650v3 machines, with runtime (memory) limits of 30 minutes (4 GB).

<sup>1</sup><https://doi.org/10.5281/zenodo.6992688>

Domain	#	$u^1$				$u^2$				$\Phi^{C^*}$ built	
		–	Clause	ART	$\Phi^{C^*}$	–	Clause	ART	$\Phi^{C^*}$	$u^1$	$u^2$
Unsolvability IPC (UIPC) 2016 Benchmarks											
BagBarman	20	8	8	8	0	0	0	0	0	0	0
BagGripper	25	3	3	3	1	0	0	0	0	1	0
BagTransport	29	6	6	6	1	16	16	16	0	1	0
Bottleneck	25	20	20	20	13	19	21	21	0	13	0
CaveDiving	25	7	7	7	3	6	6	6	0	3	0
ChessBoard	23	5	5	5	4	4	4	4	0	4	0
Diagnosis	20	5	5	5	1	4	5	5	0	6	0
DocTransfer	20	7	6	7	1	8	7	8	0	2	0
NoMystery	20	2	2	2	2	2	2	2	0	12	0
PegSol	24	24	24	24	0	24	22	22	0	0	0
PegSolRow5	15	5	5	5	3	4	4	4	1	3	1
Rovers	20	7	7	7	7	7	7	7	0	20	0
SlidingTiles	20	10	10	10	10	10	10	10	0	20	0
Tetris	20	5	5	5	0	5	5	5	0	0	0
TPP	30	16	16	16	16	14	15	15	0	25	0
$\Sigma$ UIPC	336	130	129	130	62	123	124	125	1	110	1
Unsolvability Resource-Constrained Planning (RCP) Benchmarks (Nakhost et al., 2012)											
NoMystery	150	45	45	45	43	74	81	81	0	129	0
Rovers	150	5	5	5	5	66	67	67	0	150	0
TPP	25	6	6	6	0	4	7	7	0	0	0
$\Sigma$ RCP	325	56	56	56	48	144	155	155	0	279	0
$\Sigma$ Total	661	186	185	186	110	267	279	280	1	389	1

Table 6.1.: Coverage results (number of benchmark instances proved unsolvable), comparing the different  $u^C$  No-Good techniques. Best results are highlighted in **bold**. “ $u^1$ ” and “ $u^2$ ” forward state space search using  $u^1$  respectively  $u^2$  for dead-end pruning; “–” not using any NoGood; “Clause” clause learning; “ART” CART learning; “ $\Phi^{C*}$ ” search using the offline constructed perfect NoGood formula; “ $\Phi^{C*}$  built” shows the number of benchmark instances on which  $\Phi^{C*}$  was successfully constructed.

The main purposes of this experiment are to (1) evaluate the feasibility of the perfect NoGood  $\Phi^{C*}$  construction, and to compare the overhead of the  $u^C$  computation to the evaluation of  $\Phi^{C*}$  when  $\Phi^{C*}$  can be constructed; and to (2) evaluate the effectiveness of the two NoGood learning techniques in reducing the  $u^C$  computation overhead. As basis for this evaluation, we consider forward search, pruning dead ends recognized by  $u^1$  respectively  $u^2$ , the two canonical critical-path unsolvability detector baselines. We run four variants of those base configurations: “–” not using any NoGood technique; “Clause” online clause learning; “ART” online CART learning; and “ $\Phi^{C*}$ ” constructing the perfect NoGood formula  $\Phi^{C*}$  for the respective critical-path heuristic before search, and substituting the heuristic by  $\Phi^{C*}$  during search. The mutex constraints used for the  $\Phi^{C*}$  construction were obtained from FAST DOWNWARD’s FDR planning task representation, i.e., using that every state variable can have only one value at any time.

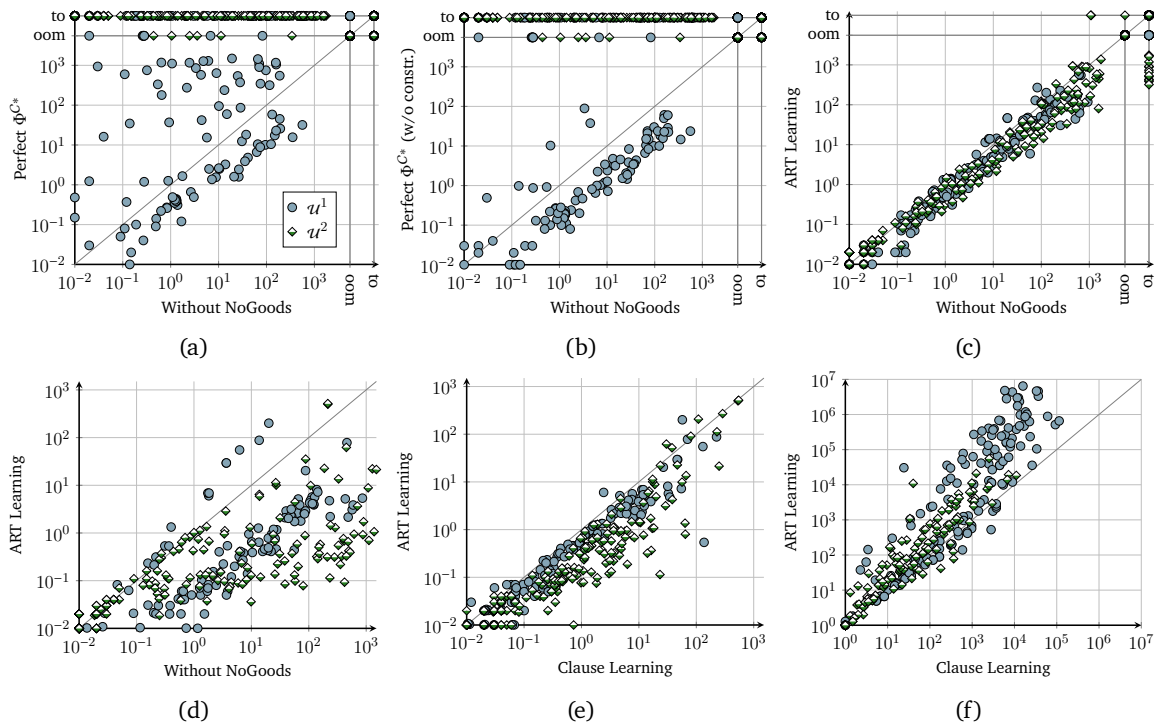


Figure 6.1.: Per-instance comparisons of (a) – (e) time spent on dead-end detection (in seconds), and (f) NoGood generalization factor: (a) compares the total time spent on the evaluation of  $u^1/u^2$  without NoGoods (x-axis) to the time spent on the construction and evaluation of  $\Phi^{C^*}$  (y-axis); (b) same as (a) but without the construction; (c) compares the evaluation of  $u^1/u^2$  without NoGoods (x-axis) to the evaluation of  $u^1/u^2$  with CART learning (y-axis), counting in the latter the time spent on evaluating and refining the NoGood formula plus evaluating  $u^1/u^2$ ; (d) like (c) but restricted to the dead ends recognized by  $u^1/u^2$ ; (e) compares the evaluation of  $u^1/u^2$  with clause learning (x-axis) to the evaluation of  $u^1/u^2$  with CART learning (y-axis), again restricted to the recognized dead ends; (f) compares the NoGood generalization factor between clause learning and CART learning.

Table 6.1 provides the coverage results. On the right, we see that constructing  $\Phi^{C^*}$  is feasible about 59% of the time for  $u^1$ , and is hardly ever feasible for  $u^2$ , exhibiting the mentioned blow-up. On the left, we see that none of the NoGood methods improves coverage for  $u^1$ . This is expected as computing  $u^1$  is comparatively cheap to evaluate, so there is only little to be gained by avoiding  $u^1$  computations.

Matters are different for  $u^2$ , computing which is much more costly (on average in our experiments, 3 orders of magnitude slower). Observe first that, somewhat surprisingly perhaps – a common perception being that  $u^2$  is way too costly to be recomputed on every state in a forward search –  $u^2$  outperforms  $u^1$  clearly here, mostly but not exclusively due to the resource-constrained benchmarks. In BagBarman and BagGripper, computing  $u^2$  is prohibitively costly, and the  $u^2$  configurations run out of time or memory during the construction of the  $u^2$  data structures already. More importantly for our work here, in contrast to  $u^1$ , online NoGood learning *does* have a considerable beneficial effect on coverage for  $u^2$ .

We now analyze the data in more detail, beyond the coarse measurement afforded by coverage. Figure 6.1 shows the effect of NoGood learning on the time spent in dead-end detection. In (a) and (b), we see that, despite its size blow-up, on those instances where  $\Phi^{C^*}$  can be constructed, dead-end detection using  $\Phi^{C^*}$  is typically much more effective than with the equivalent critical-path heuristic. (a) shows that this is often so even when taking into account the time spent constructing  $\Phi^{C^*}$ ; (b) shows that this is almost consistently

so when disregarding that time, i.e., when considering the effort spent during search only.

Figure 6.1c and 6.1d examine the runtime impact of online NoGood learning via CARTs. We see in Figure 6.1c that CART learning improves performance almost consistently, and especially for  $u^2$ . We see in Figure 6.1d that the improvement is quite strong – up to 4 orders of magnitude – when considering only those states recognized as dead ends by  $u^1/u^2$ , i.e., those states where NoGood learning may actually help (on all other states, it merely incurs an additional overhead).

Figure 6.1f examines the power of generalization, in terms of the impact of online NoGood learning on the number of  $u^1/u^2$  evaluations. We consider only those states where NoGood learning actually makes a difference – those recognized as dead-ends by  $u^1/u^2$  – and we show the improvement factor, i.e., the ratio between the number of  $u^1/u^2$  evaluations with vs. without NoGood learning. We see that both learning variants generalize quite well, yielding large improvement factors. Yet, CART learning is clearly superior, with improvement factors up to 3 orders of magnitude larger than those of clause learning. It reduces the number of evaluations by factors in the thousands for  $u^2$ , and up to millions for  $u^1$ .

Figure 6.1e compares the evaluation time of  $u^1/u^2$  between clause and CART learning, disregarding the  $u^1/u^2$  evaluation time on states not recognized as dead ends, which is identical on both sides. We see that the generalization advantage of CART learning carries over to runtime, with CART learning dominating clause learning almost consistently, and outperforming it especially for  $u^2$ . We remark though that the identical runtime share on both sides is often so large as to overshadow this advantage.

## 6.4. Experimental Evaluation

We evaluate search with dead-end learning via  $u^C$  for three different objectives: finding plans in the presence of dead ends; proving a planning task unsolvable; and generating unsolvability certificates. We next detail the general experiment setup, and then proceed over the different subjects in the given order.

### 6.4.1. Algorithm Configurations, Competing Approaches, and Benchmarks

We implemented the search algorithms from Chapter 5 and  $u^C$  conflict refinement in FAST DOWNWARD (FD) (Helmert, 2006). The source code is publicly available<sup>2</sup>. The experiments were run on a cluster of Intel Xeon E5-2660 machines running at 2.20 GHz, with runtime (memory) limits of 30 minutes (4 GB).

We next describe our parameter choices for instantiating our techniques. Afterwards, we discuss the approaches to which we compare. Finally, we describe our benchmark selection.

#### Variants of Our Techniques

In preliminary experiments, we found that search algorithms other than depth-first search hardly ever benefited from conflict-driven learning, as they did not identify enough conflicts. This pertains especially to  $A^*$ , which explores the search space in a breadth-oriented fashion, considering many options. In contrast, for the identification of conflicts, it is beneficial to search deeply not broadly, pushing the state at hand to either the goal or a dead-end situation. Given this, we focus primarily on depth-first searches. In particular, for solvable planning tasks, we consider *satisficing planning*, not providing plan quality guarantees. We focus on the DFS variant from Section 5.2, our most elegant and effective search algorithm. For

<sup>2</sup><https://doi.org/10.5281/zenodo.6992688>

ordering children nodes in DFS, we focus on an ordering by smaller value of the delete-relaxation heuristic  $h^{\text{FF}}$  (Hoffmann and Nebel, 2001), which turns out to be beneficial for both, finding plans and (to a lesser degree) proving unsolvability.

We always start with  $\mathcal{U}^C = \mathcal{U}^1$ , i.e., initializing  $C$  to the set of all singleton conjunctions. We experiment with three different conjunction learning methods, namely path-cut refinement and neighbors refinement as introduced in Section 6.2, as well as Keyder et al.’s (2014) method as a representative of prior conjunction-learning methods. Keyder et al.’s (2014) method, like Haslum’s (2012) preceding one, is based on iteratively removing conflicts in delete-relaxed plans, and we will refer to it as *relaxed-plan refinement*. We also run DFS without any refinement, as a direct comparison pointing out the impact of learning vs. an identical search without learning. For finding plans and for proving unsolvability, we run DFS with *early stopping*, stopping search, and thus skipping the remaining conflict refinements, when there are no more open nodes. For generating unsolvability certificates, we run DFS without early stopping, refining  $\mathcal{U}^C$  until it refutes the initial state.

We also run offline learning variants, refining the conjunction set only prior to search, for proving unsolvability and for dead-end detection. In either case, we use path-cut refinement and relaxed-plan refinement, as neighbors refinement is not applicable in the offline context. For proving unsolvability, we simply refine  $\mathcal{U}^C$  on the initial state until  $\mathcal{U}^C(I) = \infty$ . For dead-end detection, we refine  $\mathcal{U}^C$  on the initial state until a *size bound*  $\alpha$  is reached. Then, we use the same set  $C$  for  $\mathcal{U}^C$  dead-end detection throughout the search. Like in previous work on partial delete-relaxation heuristics (Keyder et al., 2012; Keyder et al., 2014), the size bound  $\alpha$  is multiplicative, enforcing the ratio:

$$\sum_{c \in C} |\mathcal{A}[c]| \leq \alpha \sum_{p \in \mathcal{F}} |\mathcal{A}[\{p\}]|$$

which attempts to more directly control the computational overhead of  $\mathcal{U}^C$  over  $\mathcal{U}^1$ .

To evaluate the complementarity of our method vs. strong competing methods, we design simple combinations with the two strongest alternate dead-end detection techniques, namely unsolvability heuristics  $\mathcal{U}$  obtained from merge-and-shrink abstractions respectively a potential heuristic variant (see below). The combinations additionally use  $\mathcal{U}$  to test whether a state is a dead end at state generation time. The only subtlety here is that, during refinement of  $\mathcal{U}^C$ , because of the additional unsolvability detector  $\mathcal{U}$ , the  $\mathcal{U}^C$  recognized-neighbors property may no longer be satisfied. Distinguishing the two possible cases, (a) if the identified search conflict satisfies the  $\mathcal{U}^C$  recognized-neighbors property, then we use neighbors refinement which generally works best if applicable; (b) if the property is not satisfied, then we fall back to path-cut refinement. Observe though that the latter will force  $\mathcal{U}^C$  to recognize also all the dead ends below the conflict component that were recognized by  $\mathcal{U}$ . Therefore, we experiment with two combination variants, one using both (a) and (b), and one refining  $\mathcal{U}^C$  only via (a).

We experimented with combinations of  $\mathcal{U}^C$  dead-end learning and both NoGood learning variants. Yet similar to the results in Section 6.3.3, the difference in terms of the overall performance between the two NoGood learning variants was negligible. Clause learning yielded slightly better results on the solvable part, CART learning had a slight lead on the unsolvability part. In the following, we report results for clause learning only. All observations apply equally to CART NoGood learning. By default, clause learning is switched on in all our algorithm configurations.

For ablation study purposes, we also run variants of our strongest configuration, DFS with neighbors refinement, replacing DFS by depth-oriented search (open-list based search, preferring deepest states), respectively disabling NoGood learning.

### Competing Approaches

Apart from the search algorithms and heuristic functions, the relevant techniques in our context are dead-end detection, admissible pruning techniques, and other methods for proving unsolvability.

For all of these, the state of the art at the time of writing is represented by the participants of the 2016 inaugural *Unsolvability International Planning Competition (UIPC'16)*. To provide a comprehensive picture, we include all UIPC'16 participants, except those vastly dominated in that competition. However, our interest is in understanding algorithm behavior, as opposed to running a systems competition. For systems composed of several distinct algorithm components, we therefore consider, not the systems, but their components. This pertains primarily to the winning system *AIDOS* (Seipp et al., 2016), an algorithm portfolio; for the other UIPC'16 participants, our modifications are minor. Throughout, we use the original planning task representation produced by FD's translator component (Helmert, 2009).

**Dead-End Detectors** We run *merge-and-shrink (M&S)* abstractions, in the two most competitive configurations of Hoffmann et al. (2014). One of the two M&S variants computes the perfect unsolvability detector  $\mathcal{U}^*$ , the other imposes an abstraction size limit and yields an approximate unsolvability detector. Very similar M&S heuristics were used in UIPC'16 (Torralba et al., 2016), in combination with additional irrelevance pruning (Torralba and Kissmann, 2015) and dominance pruning (Torralba and Hoffmann, 2015); we separate out the latter components to keep things clean. We furthermore run the *pattern database* unsolvability heuristic (Pommerening and Seipp, 2016), which participated in UIPC'16 on its own and as one component of *AIDOS*. Finally, we run *AIDOS*'s dead-end *potential heuristic* (Seipp et al., 2016), an enhancement of the state-equation LP heuristic (Bonet, 2013) by additional constraints over fact pairs.

**Pruning Techniques** We run *simulation-based dominance* pruning (Torralba and Hoffmann, 2015) used in the UIPC'16 M&S system (Torralba et al., 2016), which finds a simulation relation over states and prunes dominated states during search. We run *strong stubborn sets (SSS)* pruning, used in two UIPC'16 entries (Gnad et al., 2016a; Seipp et al., 2016), a long-standing partial-order reduction method (e.g., Valmari, 1989; Wehrle and Helmert, 2014) exploiting permutability of actions. We run *irrelevance pruning* (Torralba and Kissmann, 2015), also used in two UIPC'16 entries (Torralba, 2016; Torralba et al., 2016), which detects irrelevant operators (that cannot be part of an optimal solution) based on dominance analysis in a merge-and-shrink abstraction.

**Other Methods** We run BDD-based *symbolic search* (e.g., Bryant, 1986; McMillan, 1993; Edelkamp et al., 2015), specifically the *SYMPA* system (Torralba, 2016), separating out the irrelevance pruning, as for M&S above. We run *property-directed reachability (PDR)* (Bradley, 2011; Suda, 2014) as in UIPC'16 (Balyo and Suda, 2016). We run *resource-variable detection*, another component of *AIDOS*, which performs domain analysis to identify a state variable encoding a consumed resource-budget, and which during search uses a *Cartesian abstraction* (Seipp and Helmert, 2018) lower bound to prune against the remaining budget. We run *star-topology decoupled state-space search* (Gnad et al., 2015), a decomposition technique exploiting possible factorizations into star topologies. We separate the standard state-space search and strong stubborn sets pruning components used as alternatives to star-topology decoupled state-space search in the UIPC'16 system. We run partial delete-relaxation via *red-black planning* (Domshlak et al., 2015; Gnad et al., 2016b) in its most competitive configuration established after UIPC'16 (Gnad et al., 2016c). This approach searches in a relaxation where “red” state variables (but not “black” ones) are delete relaxed. If there is no relaxed plan, there cannot be a real plan either, so unsolvability of the input task can be

proved this way. The relaxation is iteratively strengthened by painting one more variable black. The only difference between our version and the UIPC'16 one is the variable order in which that is done.

We do *not* run the UIPC'16 theorem proving approach (Korovin and Suda, 2016) as this was vastly outperformed by the other competition entries. We do not run Haslum's (2016) UIPC'16 entry, which also performed badly, and is very similar to our configuration doing offline learning with relaxed-plan refinement. The main difference is that it uses a less effective representation, where relaxed plans are computed in a compiled planning task  $\Pi^C$ , whose size is worst-case exponential in  $|C|$ .

### Benchmarks

In terms of unsolvability benchmarks, we consider the same ones as in our previous experiment (Section 6.3.3), i.e., the benchmarks of the unsolvability planning competition UIPC'16, as well as the unsolvable resource-constrained planning (RCP) benchmarks, rescaling the resource-constrainedness factors in Nakhost et al.'s (2012) solvable benchmark instances.

As solvable benchmarks, naturally we use the benchmark suites of the International Planning Competition (IPC). We consider IPC editions 1998 – 2014, specifically the STRIPS benchmarks for satisficing planning (where these distinctions are made). Our learning techniques are interesting only in domains that actually contain conflicts, i.e., dead ends unrecognized under  $\mathcal{U}^1$ . Therefore, from IPC'98 – IPC'08 we use the subset of domains, where Hoffmann's analysis (Hoffmann, 2005; Hoffmann, 2011) showed this to be the case. From IPC'11 and IPC'14, where a formal analysis has not yet been carried out, we use those domains where DFS with  $\mathcal{U}^1$  dead-end detection identifies at least one conflict, i.e., where DFS backtracks out of a strongly connected component at least once, for at least one instance. Similarly to the unsolvability part, in addition to the competition benchmarks, we also consider RCP benchmarks. For the solvable case, we use the exact benchmark suites provided by Nakhost et al. (2012).

#### 6.4.2. Dead-End Detection in Solvable Planning Tasks

We consider first the solvable case. We run the eight variants of our technique described above: DFS without learning; DFS with learning using one of the three refinement methods; and four DFS learning combinations with the M&S, respectively potential heuristics. In all these algorithms, we order the children in DFS for expansion by smaller  $h^{\text{FF}}$  (Hoffmann and Nebel, 2001) value. We include results for DFS with arbitrary children ordering (expanding states in the order in which they were generated) for comparison.

We also compare to more traditional heuristic search approaches, specifically to greedy best-first search (GBFS) with  $h^{\text{FF}}$  and a dual open queue for preferred operators (Helmert, 2006). This is a canonical baseline algorithm for satisficing heuristic search planning, and yields competitive performance while being reasonably simple. We use the M&S respectively potential heuristic for dead-end detection in that baseline search. Moreover, we use simulation-based dominance, strong stubborn sets (SSS), respectively irrelevance pruning to prune the state space in that baseline search.

We run offline learning with a size bound  $\alpha$  to generate static  $\mathcal{U}^C$  unsolvability detectors. We use these in both, DFS without learning for direct comparison to our techniques, and in the GBFS baseline search for direct comparison to the other static unsolvability detectors. We experiment with  $\alpha = 2$  as that was the best size bound in prior work on partial delete relaxation heuristics, and we experiment with  $\alpha = 32$  as a larger yet still reasonable setting.

		GBFS $h^{FF}$	DFS $h^{FF}$	DFS $h^{FF}$ w/o Backtracking	
Domain	#	Coverage	Coverage	Coverage	Average runtime
IPC Benchmarks					
Airport	50	30	<b>34</b>	14	0.33
Childsnack	20	<b>1</b>	0	0	
Floortile	40	<b>10</b>	8	0	
Freecell	80	77	<b>78</b>	63	2.65
Mprime	35	<b>35</b>	27	26	2.14
Mystery	30	<b>20</b>	16	11	0.37
NoMystery	20	<b>13</b>	5	1	0.18
Openstacks	100	100	100	100	98.28
ParcPrinter	50	50	50	50	0.04
Pathways	30	23	23	22	1.0
PegSol	50	50	50	3	0.01
Pipesworld-Tankage	50	<b>40</b>	37	35	78.15
Sokoban	50	32	<b>40</b>	2	0.11
Thoughtful	20	<b>13</b>	9	9	1.65
Tidybot	20	13	13	13	60.05
TPP	30	30	30	30	2.88
Trucks	30	<b>18</b>	17	5	0.01
Woodworking	50	8	<b>40</b>	<b>40</b>	257.94
$\Sigma$ IPC	755	563	<b>577</b>	424	56.38
Solvable Resource-Constrained Planning (RCP) Benchmarks (Nakhost et al., 2012)					
NoMystery	210	<b>63</b>	27	7	0.17
Rovers	210	1	<b>8</b>	2	0.38
TPP	30	5	<b>13</b>	3	2.71
$\Sigma$ RCP	450	<b>69</b>	48	12	0.84
$\Sigma$ Total	1205	<b>632</b>	625	436	54.86

Table 6.2.: Solvable benchmarks. Coverage results (number of instances solved within the time/memory limits) for the two base algorithms. We additionally include results for DFS without backtracking, counting as solved only those tasks, where the first DFS search branch finds a goal state. Best results are highlighted in **bold**. Abbreviations: “#” total number of instances; “GBFS  $h^{FF}$ ” greedy best-first search with  $h^{FF}$  and preferred operators; “DFS  $h^{FF}$ ” depth-first search as per Algorithm 5.2, ordering children nodes via  $h^{FF}$ .

### Baseline Comparison: DFS $h^{FF}$ versus GBFS $h^{FF}$

The base algorithm our learning methods start from, DFS with  $h^{FF}$  children ordering, is quite different from the GBFS  $h^{FF}$  baseline. So, we start with a brief experiment comparing the two, without learning. Consider the coverage results from Table 6.2. We see that the two base algorithms have complementary strengths in different domains. Sometimes the differences are drastic, most notably in Sokoban, Woodworking, as well as resource-constrained Rovers and TPP where DFS is much stronger; and Mprime, Mystery, NoMystery, and Thoughtful where GBFS is much stronger. In total, these differences cancel each other out though, and the two algorithms are on a similar level. This is remarkable in itself, seeing as GBFS is widely used in heuristic search planning while, to the authors’ knowledge, DFS has not been used at all in this context yet. For our purposes here though, the main conclusion is that, overall, performance is comparable so we are not a priori much disadvantaging either side.

Table 6.2 also includes a variant of DFS disallowing backtracking. This serves to point out the cases where, with  $h^{\text{FF}}$  tie breaking, despite the presence of conflicts in principle, no learning will happen simply because a goal state is found without ever encountering a conflict. As the table shows, this happens to a surprising extent. In particular, this simplistic search procedure is an extremely effective solver for ParcPrinter, Pathways, and TPP, where it solves all instances solved by the common baseline (almost all, in case of Pathways), but within split seconds or a few seconds at most.

### Refinement Algorithms & Baseline Comparison

Table 6.3 shows the coverage results (number of instances solved). We will consider offline learning separately below. For the resource-constrained domains, as the constrainedness level has a large impact on performance for most algorithms, we show data for each level separately. Note here that IPC Mprime, Mystery, and NoMystery also are resource-constrained domains. However, their constrainedness levels are not known or only partially known, so we do not separate these.

Consider first the different refinement variants within DFS  $h^{\text{FF}}$ , i.e., our neighbors refinement (“Nei”) and path-cut refinement (“Pat”) methods vs. the relaxed-plan refinement (“RP”) from prior work. There are large performance differences in many domains, showing that the way we learn conjunctions is important. In particular, relaxed-plan refinement has the lead, and a marginal one at that, in only one case (IPC Trucks), showing that it is important to target the conjunction learning to dead-end detection. Path-cut refinement is best overall on the IPC, but neighbors refinement has a huge advantage in the resource-constrained benchmarks so has best overall coverage.

Consider now the effect of learning vs. no learning (“No”). On the resource-constrained domains, the improvement is consistently dramatic, with some minor exceptions in TPP. On the IPC benchmarks, the picture is much more mixed. On Airport, Freecell, PegSol, Pipesworld-Tankage, Sokoban, and Trucks, the learning has a detrimental effect. We will analyze in some detail below why that is so. On Tidybot and Woodworking, as well as ParcPrinter, Pathways, and TPP for the DFS  $h^{\text{FF}}$  variants, the learning has no impact at all. That is mostly because DFS does not identify many conflicts here, so the learning is seldom, or never, invoked (we also get back to this in more detail below). On the other domains, improvements are possible. These are most pronounced in Floortile for path-cut and relaxed-plan refinement; in Mystery, and NoMystery for neighbors refinement; and in ParcPrinter and Pathways for neighbors refinement in DFS without  $h^{\text{FF}}$  children ordering.

Overall, compared to the baselines without learning (including also GBFS), on the IPC the learning is detrimental, as the size of its losses outweighs that of its gains. On the resource-constrained domains, the picture is very different, with a substantial and consistent win over the baselines. The only exception is NoMystery with  $\mathfrak{C} = 2.0$ , where fuel is relatively plentiful and the baseline does not struggle with unrecognized dead ends as much as it does with constrainedness levels closer to 1.0.

Ordering children nodes in DFS by  $h^{\text{FF}}$  (“DFS  $h^{\text{FF}}$ ”) outperforms arbitrary children ordering (“DFS”) dramatically overall, and almost consistently across domains. Therefore, from now on, we consider  $h^{\text{FF}}$  children ordering exclusively. We will do so not only for the solvable case, but also for unsolvable benchmarks, where  $h^{\text{FF}}$  children ordering does not have as large an impact, but is never worse and sometimes helps (intuitively because DFS is drawn towards more relevant dead-end situations, learning more relevant knowledge). Henceforth, whenever we say “DFS” we mean “DFS with  $h^{\text{FF}}$  children ordering”.

Domain	#	GBFS $h^{FF}$	DFS		DFS $h^{FF}$				DFS $h^{FF}$ comb.				GBFS $h^{FF}$ +					
			No	Nei	No	Nei	Pat	RP	MSa	Pot	N	NP	Msa	Pot	Sim	SSS	Pruning	Irr
IPC Benchmarks																		
Airport	50	30	35	30	34	24	21	21	17	17	24	23	18	<b>38</b>	12	35	34	
Childsnack	20	1	0	0	0	1	1	1	1	1	0	0	0	0	3	0	<b>4</b>	
Floortile	40	10	6	0	8	4	<b>37</b>	27	4	4	4	4	15	15	8	12	13	
Freecell	80	77	78	72	78	71	72	69	50	50	71	71	51	76	51	77	<b>80</b>	
Mprime	35	<b>35</b>	18	19	27	28	27	27	17	17	27	30	18	30	25	30	34	
Mystery	30	20	14	20	16	26	23	23	20	20	24	26	19	<b>28</b>	15	25	18	
NoMystery	20	13	6	9	5	13	10	8	8	12	5	8	10	5	<b>20</b>	8	14	
Openstacks	100	<b>100</b>	<b>100</b>	<b>100</b>	<b>100</b>	98	<b>100</b>	<b>100</b>	32	29	66	49	30	66	50	90	58	
ParcPrinter	50	<b>50</b>	25	38	<b>50</b>	<b>50</b>	<b>50</b>	<b>50</b>	<b>50</b>	10	<b>50</b>	10	<b>50</b>	<b>50</b>	<b>50</b>	<b>50</b>	<b>50</b>	
Pathways	30	23	4	5	23	23	23	23	23	23	23	23	23	23	<b>24</b>	22	22	
PegSol	50	<b>50</b>	<b>50</b>	41	<b>50</b>	41	30	12	41	41	48	23	<b>50</b>	<b>50</b>	<b>50</b>	<b>50</b>	<b>50</b>	
Pipesworld-Tankage	50	<b>40</b>	17	14	37	35	35	34	8	8	17	17	8	16	16	<b>40</b>	30	
Sokoban	50	32	42	11	40	9	6	3	10	8	9	9	42	35	5	42	<b>48</b>	
Thoughtful	20	<b>13</b>	5	1	9	9	9	9	5	5	10	7	5	7	8	7	8	
Tidybot	20	<b>13</b>	8	8	<b>13</b>	<b>13</b>	<b>13</b>	<b>13</b>	0	0	<b>13</b>	<b>13</b>	0	12	0	12	0	
TPP	30	<b>30</b>	24	23	<b>30</b>	<b>30</b>	<b>30</b>	<b>30</b>	<b>30</b>	<b>30</b>	<b>30</b>	<b>30</b>	<b>30</b>	<b>30</b>	20	<b>30</b>	25	
Trucks	30	<b>18</b>	11	6	17	9	16	17	10	10	9	9	<b>18</b>	16	<b>18</b>	16	<b>18</b>	
Woodworking	50	8	6	6	40	40	40	40	13	13	28	28	8	7	49	22	<b>50</b>	
$\Sigma$ IPC	755	563	449	403	<b>577</b>	524	543	507	339	298	458	380	395	504	424	568	556	
Solvable Resource-Constrained Planning (RCP) Benchmarks (Nakhost et al., 2012)																		
NoMystery (1.0)	30	2	0	10	1	12	7	4	19	7	3	7	20	0	<b>26</b>	1	8	
NoMystery (1.1)	30	3	0	10	0	12	8	5	17	8	3	8	<b>20</b>	1	<b>20</b>	2	9	
NoMystery (1.2)	30	5	0	7	4	14	13	9	18	10	7	10	21	1	<b>26</b>	2	16	
NoMystery (1.3)	30	6	0	7	5	13	9	7	15	11	5	7	23	3	<b>27</b>	3	25	
NoMystery (1.4)	30	12	0	5	3	17	13	9	11	16	8	10	24	5	<b>29</b>	10	25	
NoMystery (1.5)	30	12	0	6	5	17	13	9	10	15	6	10	24	5	<b>29</b>	9	<b>29</b>	
NoMystery (2.0)	30	23	0	8	9	19	18	14	12	17	11	15	28	7	<b>30</b>	16	<b>30</b>	
$\Sigma$ NoMystery	210	63	0	53	27	104	81	57	102	84	43	67	160	22	<b>187</b>	43	142	
Rovers (1.0)	30	0	0	19	0	<b>23</b>	9	8	7	17	<b>23</b>	<b>23</b>	6	0	15	0	0	
Rovers (1.1)	30	0	0	16	0	<b>23</b>	12	13	7	19	22	21	5	0	15	0	2	
Rovers (1.2)	30	0	0	<b>19</b>	0	18	7	9	5	<b>19</b>	18	18	1	0	14	0	2	
Rovers (1.3)	30	0	0	17	0	<b>20</b>	7	6	8	17	<b>20</b>	<b>20</b>	2	0	16	0	1	
Rovers (1.4)	30	0	0	<b>19</b>	1	<b>19</b>	8	7	8	18	<b>19</b>	<b>19</b>	4	0	16	0	3	
Rovers (1.5)	30	0	0	19	1	<b>21</b>	12	8	10	19	<b>21</b>	<b>21</b>	5	0	15	0	6	
Rovers (2.0)	30	1	0	17	6	21	16	15	12	18	21	21	11	0	<b>23</b>	0	10	
$\Sigma$ Rovers	210	1	0	126	8	<b>145</b>	71	66	57	127	144	143	34	0	114	0	24	
TPP (1.0)	5	0	0	0	1	1	0	0	1	1	1	0	0	0	<b>2</b>	0	1	
TPP (1.1)	5	0	0	0	0	2	0	0	1	2	0	0	1	0	<b>3</b>	0	<b>3</b>	
TPP (1.2)	5	0	0	0	2	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>	0	0	2	0	<b>3</b>	0	<b>3</b>	
TPP (1.3)	5	2	0	0	2	<b>5</b>	3	3	4	4	0	0	4	0	4	2	4	
TPP (1.4)	5	3	0	0	3	<b>5</b>	4	<b>5</b>	<b>5</b>	<b>5</b>	0	0	4	0	4	2	4	
TPP (1.5)	5	0	0	0	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	0	0	<b>5</b>	0	<b>5</b>	0	2	
$\Sigma$ TPP	30	5	0	0	13	<b>21</b>	15	16	19	20	1	0	16	0	<b>21</b>	4	17	
$\Sigma$ RCP	450	69	0	179	48	270	167	139	178	231	188	210	210	22	<b>322</b>	47	183	
$\Sigma$ Total	1205	632	449	582	625	<b>794</b>	710	646	517	529	646	590	605	526	746	615	739	

Table 6.3.: Solvable benchmarks. Coverage results, comparing to the state of the art. Best results are highlighted in **bold**. Abbreviations: “GBFS  $h^{FF}$ ” greedy best-first search with preferred operators using  $h^{FF}$ ; “DFS” DFS with arbitrary children ordering; “DFS  $h^{FF}$ ” DFS ordering children via  $h^{FF}$ ; “DFS  $h^{FF}$  comb.” combination with another unsolvability detector (N/NP, see below); “No” no learning; “Nei” neighbors refinement; “Pat” path-cut refinement; “RP” relaxed-plan refinement; “N” combination using neighbors refinement only; “NP” combination using neighbors refinement if applicable, else path-cut refinement; “MSa” approximate merge-and-shrink abstraction; “Pot” potential heuristic; “Sim” simulation dominance pruning; “SSS” strong stubborn sets pruning; “Irr” irrelevance pruning.

### Competing Approaches

Consider next the competing dead-end detection and pruning techniques in Table 6.3. On the IPC benchmarks, compared to their GBFS baseline, only strong stubborn sets pruning (“SSS”) improves overall coverage, and only SSS and irrelevance pruning (“Irr”) have higher overall coverage than our methods (DFS  $h^{\text{FF}}$  with neighbors refinement respectively path-cut refinement). The strengths of these two methods lie in not deteriorating the baseline as much, and in the improvements they yield in Airport, Sokoban, and Woodworking. The strong cases for our methods are Floortile where DFS with path-cut refinement, and also with relaxed-plan refinement, vastly outperforms all competitors; Openstacks, Tidybot, TPP, and Woodworking where the DFS baseline is much better than the GBFS baseline, and there is (almost) no learning overhead. The coverage decrease in Openstacks when using neighbors refinement is an artifact of how we track the recognized neighbors during search.

On the resource-constrained benchmarks, the picture is again much clearer. Potential heuristics (“Pot”) and SSS are basically useless. Irrelevance pruning, merge-and-shrink (“MSa”), and especially simulation-based dominance pruning (“Sim”) excel in NoMystery. In the other two domains, our learning methods tend to be superior. In Rovers, DFS with neighbors refinement (“Nei”) vastly outperforms all UIPC’16 competitors. Similarly in TPP, except for simulation-based dominance which does equally well in coverage (though typically a lot worse in runtime, on commonly solved instances). In terms of overall coverage, DFS with neighbors refinement is the clear winner. Other algorithms perform better in one part (IPC vs. RCP) of the benchmark set, but DFS with neighbors refinement is most consistently good overall.

### Combination with Other Unsolvability Detectors

Consider finally the combinations of neighbors-refinement DFS with merge-and-shrink respectively potential heuristics. For merge-and-shrink, this basically does not work well here. The DFS  $h^{\text{FF}}$  component dominates the combined methods almost consistently on the IPC benchmark set. On the RCP part, the combinations vastly improve over the DFS  $h^{\text{FF}}$  baseline, yet the configurations using learning and merge-and-shrink alone do even better. So, so this is not a valuable result. In particular, there is no case where a combination outperforms both its components.

For the combination with potential heuristics, the picture is similar, though not quite as bleak. The “N” combination outperforms the DFS  $h^{\text{FF}}$  component on PegSol, where the potential heuristic prevents most (but not all) of the loss compared to the DFS baseline. The “N” combination does better than both of its components on Thoughtful (though by only 1 instance relative to the DFS component).

### Offline versus Online Conjunction Learning

Table 6.5 shows the coverage results of the offline versus conflict-driven online conjunction learning variants. On the IPC part, with  $\alpha = 2$ , though not with  $\alpha = 32$ , the offline methods are, generally, superior on these benchmarks compared to the same refinement methods when used online. This is, however, simply because having a small size bound means to be less risky in the sense that the additional overhead is limited even when the dead-end detection accuracy is not increased. In particular, the offline methods with  $\alpha = 2$  avoid the dramatic performance losses in Airport, PegSol, and Sokoban. However, this advantage diminishes the larger we choose the size bound  $\alpha$ . Moreover, the risk reduction also comes with a benefits reduction on those domains where the online variants excel, most notably Floortile and NoMystery. Indeed, the offline-learning DFS variants can beat the coverage of the DFS no-learning baseline (“DFS No”) in just NoMystery, doing so by only 1 instance, whereas the online learning variants beat it in 6 domains.

Domain	#	GBFS	DFS (No / Online)				DFS (Offline)				GBFS (Offline)			
			No	Nei	Pat	RP	$\alpha = 2$		$\alpha = 32$		$\alpha = 2$		$\alpha = 32$	
							Pat	RP	Pat	RP	Pat	RP	Pat	RP
IPC Benchmarks														
Airport	50	30	34	24	21	21	33	34	27	33	29	35	27	33
Childsnack	20	1	0	1	1	1	0	0	0	0	1	0	1	0
Floortile	40	10	8	4	37	27	8	8	7	8	10	15	10	15
Freecell	80	77	78	71	72	69	78	78	74	78	77	78	75	77
Mprime	35	35	27	28	27	27	22	27	11	27	29	30	19	30
Mystery	30	20	16	26	23	23	15	16	13	15	17	21	14	21
NoMystery	20	13	5	13	10	8	5	5	6	5	13	8	12	8
Openstacks	100	100	100	98	100	100	99	68	93	53	100	68	92	53
ParcPrinter	50	50	50	50	50	50	48	50	45	50	47	50	46	50
Pathways	30	23	23	23	23	23	23	23	5	23	22	23	4	23
PegSol	50	50	50	41	30	12	50	50	50	50	49	50	49	50
Pipesworld-Tankage	50	40	37	35	35	34	36	37	30	37	39	40	32	39
Sokoban	50	32	40	9	6	3	38	40	22	40	30	42	26	42
Thoughtful	20	13	9	9	9	9	9	9	9	9	13	8	13	8
Tidybot	20	13	13	13	13	13	12	13	3	12	12	13	5	12
TPP	30	30	30	30	30	30	30	30	30	28	30	30	29	28
Trucks	30	18	17	9	16	17	17	17	14	16	18	18	16	15
Woodworking	50	8	40	40	40	40	38	40	31	40	8	8	4	8
$\Sigma$ IPC	755	563	577	524	543	507	561	545	470	524	544	537	474	512
Solvable Resource-Constrained Planning (RCP) Benchmarks (Nakhost et al., 2012)														
NoMystery (1.0)	30	2	1	12	7	4	1	1	1	1	3	2	3	2
NoMystery (1.1)	30	3	0	12	8	5	0	0	0	0	7	3	6	3
NoMystery (1.2)	30	5	4	14	13	9	4	4	3	4	12	5	11	5
NoMystery (1.3)	30	6	5	13	9	7	5	5	5	5	19	6	17	6
NoMystery (1.4)	30	12	3	17	13	9	3	3	3	3	23	12	21	12
NoMystery (1.5)	30	12	5	17	13	9	5	5	5	5	23	12	20	12
NoMystery (2.0)	30	23	9	19	18	14	9	9	8	9	29	23	29	20
$\Sigma$ NoMystery	210	63	27	104	81	57	27	27	25	27	116	63	107	60
Rovers (1.0)	30	0	0	23	9	8	0	0	0	0	1	0	1	0
Rovers (1.1)	30	0	0	23	12	13	0	0	0	0	2	0	2	0
Rovers (1.2)	30	0	0	18	7	9	0	0	0	0	1	0	1	0
Rovers (1.3)	30	0	0	20	7	6	0	0	0	0	2	0	1	0
Rovers (1.4)	30	0	1	19	8	7	1	1	1	1	1	0	1	0
Rovers (1.5)	30	0	1	21	12	8	1	1	1	1	2	0	1	0
Rovers (2.0)	30	1	6	21	16	15	6	6	3	6	9	1	8	1
$\Sigma$ Rovers	210	1	8	145	71	66	8	8	5	8	18	1	15	1
TPP (1.0)	5	0	1	1	0	0	1	1	0	0	0	0	0	0
TPP (1.1)	5	0	0	2	0	0	0	0	0	0	0	0	0	0
TPP (1.2)	5	0	2	3	3	3	2	2	1	0	2	0	2	0
TPP (1.3)	5	2	2	5	3	3	2	2	0	0	4	2	3	0
TPP (1.4)	5	3	3	5	4	5	3	3	2	0	4	3	3	0
TPP (1.5)	5	0	5	5	5	5	5	5	5	0	5	0	3	0
$\Sigma$ TPP	30	5	13	21	15	16	13	13	8	0	15	5	11	0
$\Sigma$ RCP	450	69	48	270	167	139	48	48	38	35	149	69	133	61
$\Sigma$ Total	1205	632	625	794	710	646	609	593	508	559	693	606	607	573

Table 6.4.: Solvable benchmarks. Coverage results, comparing offline versus online conjunction learning methods. Best results are highlighted in **bold**. Abbreviations: GBFS: the GBFS  $h^{\text{FF}}$  baseline; DFS: DFS ordering children by  $h^{\text{FF}}$ ;  $\alpha = 2$  and  $\alpha = 32$  offline C-learning with size bound 2 and 32, respectively. Other abbreviations as before.

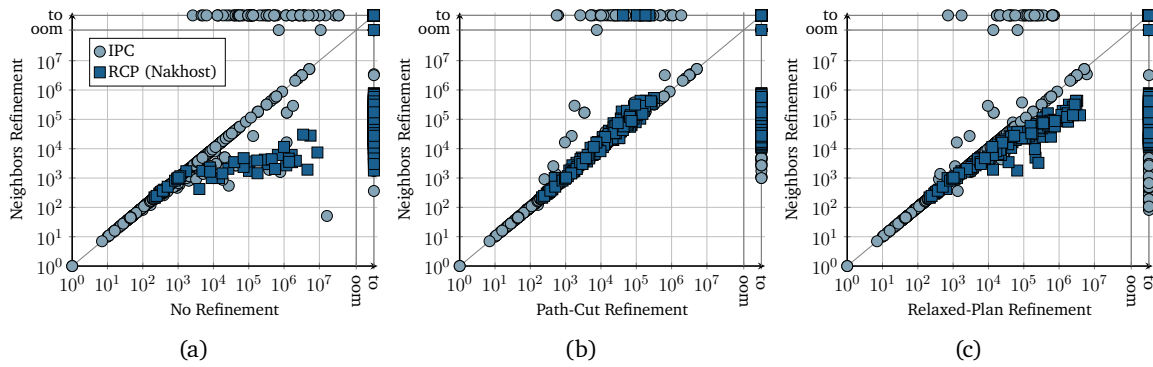


Figure 6.2.: Solvable benchmarks. Per-instance DFS search space size (number of visited states) comparison between different learning techniques. We compare neighbors refinement on the  $y$ -axis to (a) no learning, (b) path-cut refinement, respectively (c) relaxed-plan refinement on the  $x$ -axis. “to”: time limit and “oom” memory limit exceeded.

On the resource-constrained benchmarks, matters are very clear-cut. The baselines are weak, and online learning improves them vastly. The same is not true for offline-learning DFS, which never improves over the baseline at all. Offline-learning GBFS is more successful, improving over the GBFS baseline in almost all cases with path-cut refinement. But it becomes competitive with online-learning DFS only in NoMystery for large values of  $\mathfrak{C}$ .

### Search Reduction

Figure 6.2 provides a view on the search space sizes (number of states visited by search) under our different DFS learning techniques, i.e., no learning vs. the three different refinement methods. We use neighbors refinement as the comparison baseline.

Consider first the comparison between neighbors refinement and no learning, Figure 6.2a. On the IPC benchmarks, in line with the above, we see that the learning is risky, reducing search effort in some cases, while not in many others; but causing a substantial overhead. On the resource-constrained benchmarks, on the other hand, the benefits of learning are dramatic. Most benchmark instances are not solved at all without learning. On those that are solved, we get search space reductions of several orders of magnitude. Observe that the *only* reason for this is generalization, i.e., refinements of  $\mathcal{U}^C$  on conflict states  $s$  leading to pruning states other than  $s$ . Without generalization, the search spaces would be identical, including tie-breaking. Generalization is what lifts a hopeless planner (DFS with  $\mathcal{U}^1$  dead-end detection) to a planner competitive with the state of the art in resource-constrained planning.

In the comparison between neighbors refinement and path-cut refinement, Figure 6.2b, we see that the methods either perform very similarly, or are highly complementary (with one of the two methods failing to solve the task).

The comparison in Figure 6.2c between neighbors refinement and relaxed-plan refinement provides further evidence that tailoring the refinement method to dead-end detection is typically beneficial. In the vast majority of cases where learning takes place (where conflicts are identified by search), neighbors refinement leads to better generalization, and thus to smaller search spaces, than relaxed-plan refinement.

Domain	#	(A) Analysis			(B) Ablation Study			
		# Confl. Id.	Confl. Id., Common		DFS		DOS	
			Slow.	Redu.	Cl	NoCl	Cl	NoCl
IPC Benchmarks								
Airport	50	10	12.7	1.2	24	24	24	24
Childsnack	20	1			1	0	1	1
Floortile	40	4	78.2	6.2	4	4	4	4
Freecell	80	8	2.9	1	71	71	72	72
Mprime	35	2	20.2	1.8	28	28	28	28
Mystery	30	13	5.1	316.1	26	26	25	25
NoMystery	20	12	4.7	100	13	12	11	12
Openstacks	100	0			98	100	99	99
ParcPrinter	50	0			50	50	50	50
Pathways	30	1	2.1	1.2	23	23	23	23
PegSol	50	38	29.7	1.7	41	42	37	40
Pipesworld-Tankage	50	1	24.6	1	35	36	38	39
Sokoban	50	7	59.8	2.8	9	9	12	12
Thoughtful	20	0			9	9	9	9
Tidybot	20	0			13	13	13	13
TPP	30	0			30	30	30	30
Trucks	30	4	63.1	2.6	9	9	9	9
Woodworking	50	0			40	40	33	33
Σ IPC	755	101			524	526	518	523
Solvable Resource-Constrained Planning (RCP) Benchmarks (Nakhost et al., 2012)								
NoMystery (1.0)	30	12	1.2	114	12	10	10	7
NoMystery (1.1)	30	12			12	11	12	11
NoMystery (1.2)	30	14	1.4	16.2	14	14	14	12
NoMystery (1.3)	30	12	3.3	79.5	13	11	12	11
NoMystery (1.4)	30	17	1.4	19.2	17	14	16	14
NoMystery (1.5)	30	16	1.5	17.8	17	15	16	14
NoMystery (2.0)	30	14	1.6	31.1	19	19	19	19
Σ NoMystery	210	97			104	94	99	88
Rovers (1.0)	30	23			23	20	22	20
Rovers (1.1)	30	23			23	18	20	16
Rovers (1.2)	30	18			18	16	17	14
Rovers (1.3)	30	20			20	17	18	16
Rovers (1.4)	30	19	2.3	12.8	19	17	18	16
Rovers (1.5)	30	21	3.8	29.8	21	17	20	16
Rovers (2.0)	30	19	3.4	96.8	21	19	20	17
Σ Rovers	210	143			145	124	135	115
TPP (1.0)	5	1	54.8	115.1	1	0	0	0
TPP (1.1)	5	2			2	1	1	0
TPP (1.2)	5	3	1.5	111.4	3	3	3	3
TPP (1.3)	5	5	2	326.1	5	3	4	3
TPP (1.4)	5	4	7.1	31.1	5	5	5	5
TPP (1.5)	5	3	1.7	5.7	5	5	5	5
Σ TPP	30	18			21	17	18	16
Σ RCP	450	258			270	235	252	219
Σ Total	1205	359			794	761	770	742

Table 6.5.: Solvable benchmarks. Results for DFS with neighbors refinement. (A) performance analysis: “# Confl. Id.” number of solved instances in which at least one conflict was identified; “Confl. Id., Common” results over instances solved with and without learning, in which at least one conflict was identified; “Slow.”  $u^C$  slowdown with respect to  $u^1$ , measured in terms of the geometric mean size increase factor  $\alpha$  of  $u^C$  upon termination; “Redu.” geometric mean search space size (number of visited states) reduction factor with learning relative to without. (B) ablation study: coverage results for “DOS” depth-oriented search; “Cl”  $u^C$  clause learning enabled, respectively “NoCl” disabled. Best results are highlighted in **bold**.

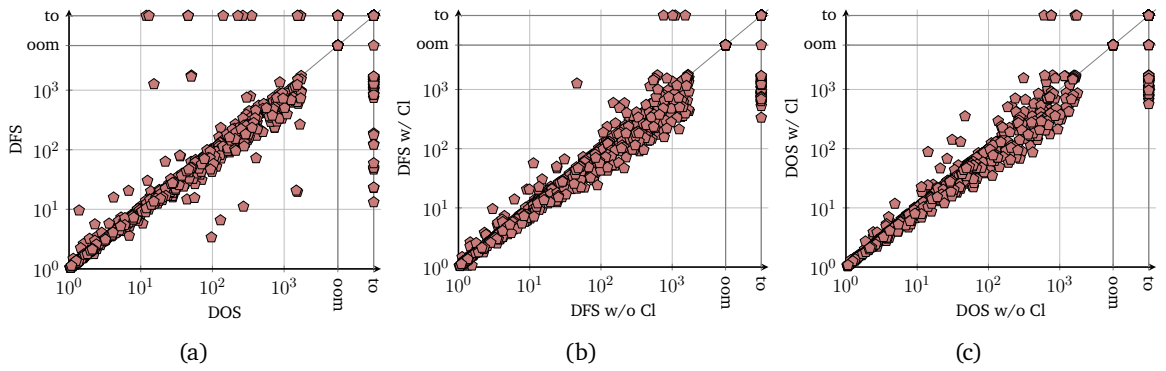


Figure 6.3.: Solvable benchmarks. Per-instance runtime (in seconds) comparisons between (a) DOS ( $x$ -axis) vs. DFS ( $y$ -axis); (a) DFS without clause learning ( $x$ -axis) vs. with clause learning ( $y$ -axis); (a) DOS without clause learning ( $x$ -axis) vs. with clause learning ( $y$ -axis). All configurations use neighbors refinement. “to”: time limit exceeded; respectively “oom” memory limit exceeded.

### Performance Analysis

Part (A) of Table 6.5 shows additional data supporting a performance analysis with respect to the three prerequisites for online learning to work well: (1) *conflict identification*, i.e., the ability of forward search to find conflicts and thus enable the learning in the first place; (2) *learning effectiveness*, i.e., the ability of recognizing dead-ends with small conjunction sets  $C$ ; (3) *strong generalization*, i.e., the ability of  $\mathcal{U}^C$  to detect states  $s'$  it was not trained on. (1) is captured in terms of the “# Confl. Id.” data, the number of instances on which at least one conflict was identified, and hence some learning was done. (2) is captured in terms of “Slow.”, the slowdown relative to  $\mathcal{U}^1$ , i.e., the size of the representation underlying the  $\mathcal{U}^C$  computation as a multiple of that for singleton conjunctions. Finally, (3) is captured in terms of the “Redu.” data, the search space size reduction factor relative to using only  $\mathcal{U}^1$  for dead-end detection.

On commonly solved instances, the slowdown and search reduction factors are directly comparable, and a performance advantage should be expected, roughly, when the former exceeds the latter. Indeed, this is a good indicator in our data here. The domains with large reduction yet small slowdown are IPC Mystery and NoMystery, as well as all resource-constrained domains, where indeed neighbors refinement vastly improves the coverage of DFS. Conversely, small reductions yet large slowdowns occur in Airport, Floortile, Mprime, PegSol, Pipesworld-Tankage, Sokoban, and Trucks. Except for Mprime, these are precisely the cases where neighbors refinement is detrimental. In Mprime, neighbors refinement actually (slightly) improves coverage, yet there are only few commonly solved instances where at least one conflict is identified, which may contribute to the unclear picture. Similarly in Childsnack, where coverage is improved from 0 to 1 so there is no common instance basis to use for comparison. In all other domains – Openstacks, ParcPrinter, Pathways, Thoughtful, Tidybot, TPP, and Woodworking– the lack of advantages through learning are due to a lack of ability (1), with no or hardly any conflicts being identified by forward search.

### Ablation Studies

Let us finally consider the ablation study, Table 6.5 part (B), fixing neighbors refinement but varying the search algorithm – DFS vs. DOS (depth-oriented search, cf. Chapter 5) – as well as switching NoGood clause learning on and off. On the IPC benchmarks, both DFS and clause learning, as opposed to DOS and no clause learning, have little impact on coverage. On the resource-constrained benchmarks though, both clearly and significantly improve coverage. Overall, they are useful algorithm improvements. Figure 6.3

provides further evidence towards this through a per-instance runtime comparison.

### 6.4.3. Proving Unsolvability

We now consider the unsolvable case. We run the same eight variants of DFS with conflict-driven learning as before. In all variants, we stop learning new conjunctions once no more states are left for expansion, i.e., not guaranteeing to generate unsolvability certificates. We run offline learning without a size bound to prove unsolvability directly on the initial state; and we run offline learning with size bound  $\alpha$  to generate static  $\mathcal{U}^C$  unsolvability detectors. We use  $\alpha \in \{2, 32\}$  like above, and we use the resulting static unsolvability detectors in DFS without learning.

We run blind forward search as a simple reference baseline, exhaustively exploring the reachable part of the state space. We run search with  $\mathcal{U}^1$  as a canonical unsolvability detector. We run all the competing algorithms described above, i.e., search with alternate unsolvability detectors, search with admissible pruning techniques, as well as the other UIPC'16 techniques including BDD-based symbolic search etc. The admissible pruning techniques are run along with  $\mathcal{U}^1$ , which is more competitive than blind search.

Table 6.6 shows the main coverage results. As before, we will consider offline learning separately below. We again distinguish different constrainedness levels for resource-constrained domains. In difference to the IPC, in the UIPC the latter is possible for the resource-constrained domains, so we separate these from the remaining UIPC benchmarks.

### Refinement Algorithms & Baseline Comparison

Compared to DFS without learning, all refinement methods result in a performance boost on resource-constrained domains, where conflict learning is key, especially with constrainedness levels close to 1 where conflicts abound. On the non-resource UIPC'16 benchmarks though, this kind of learning simply does not work well. It helps only, for some configurations, in Diagnosis and DocTransfer. We will analyze the reasons below.

Compared to the baselines, DFS without learning has basically the same coverage as search with  $\mathcal{U}^1$ , which makes sense as both use the same dead-end detector throughout. Blind search is consistently outclassed except in BagBarman where it is state of the art (the pattern database heuristic does not detect any dead-ends in this domain, so defaults to blind search).

Comparing the different refinement variants within our DFS framework, they behave similarly overall, though there are remarkable differences in individual domains, namely Bottleneck, Diagnosis, DocTransfer, and PegSol. On resource-constrained domains, neighbors refinement is clearly superior, followed by path-cut refinement and the relaxed-plan refinement from prior work.

### Competing Approaches

Consider next the UIPC'16 algorithms in Table 6.6. Potential heuristics clearly dominate on the non-resource UIPC'16 benchmarks, outclassing the competition (including our techniques), beat only on BagBarman, Diagnosis, and DocTransfer. On resource-constrained domains, on the other hand, potential heuristics are very weak. The next-best techniques from UIPC'16 are approximate merge-and-shrink, PDBs, and red-black state space search, with reasonable results on non-resource domains, and with strong results on resource-constrained ones. Strong stubborn sets pruning does not yield any benefits here. Dominance

Domain	Base			DFS		DFS comb.				UIPC'16 Algorithms															
	#	Bli	$\mathcal{U}^1$	No	Nei	Pat	RP	MSa		Pot		Detection			Prun. + $\mathcal{U}^1$			Other							
								N	NP	N	NP	MSp	MSa	PDB	Pot	Sim	SSS	Irr	BDD	PDR	Res	Sta	RB		
Unsolvability IPC (UIPC) 2016 Benchmarks																									
BagBarman	20	12	4	4	0	0	0	0	0	0	0	0	1	12	4	2	4	12	8	0	0	0	0		
BagGripper	25	5	3	3	0	0	0	2	3	2	0	2	3	3	3	0	3	0	7	0	0	0	1		
BagTransport	29	6	6	6	5	5	6	5	5	24	12	1	6	7	24	7	5	7	7	0	0	10	1		
Bottleneck	25	9	20	21	9	13	14	9	9	25	18	5	20	19	25	0	20	0	0	21	0	0	25		
CaveDiving	25	7	7	7	5	4	4	5	6	5	5	3	7	8	7	7	7	7	7	5	0	0	5		
ChessBoard	23	5	5	5	2	1	1	2	2	23	5	2	5	5	23	5	6	5	9	1	0	0	4		
Diagnosis	20	6	7	7	9	5	12	6	7	9	9	5	4	4	4	0	7	0	0	7	0	0	10		
DocTransfer	20	5	7	6	4	5	8	8	9	4	3	5	10	12	7	15	6	12	10	0	0	0	3		
PegSol	24	24	24	24	14	12	4	14	14	20	14	24	24	24	22	24	24	24	24	0	0	0	22		
PegSolRow5	15	5	5	5	4	3	2	4	4	15	9	3	4	5	15	5	5	5	4	3	0	0	6		
SlidingTiles	20	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	10		
Tetris	20	5	5	5	5	5	5	5	5	20	0	5	5	10	20	0	5	10	5	0	0	0	0		
$\Sigma$	266	99	103	103	67	63	66	70	74	157	85	65	99	119	164	75	102	92	91	37	0	10	87		
NoMystery (0.4)	5	2	2	2	3	4	3	5	5	5	5	5	5	5	5	5	2	5	5	3	5	4	4		
NoMystery (0.85)	5	0	0	0	2	2	2	2	2	0	2	2	2	2	0	2	0	2	2	1	2	2	2		
NoMystery (0.99)	5	0	0	0	2	1	0	1	2	0	2	2	1	2	0	2	0	2	2	0	1	2	2		
NoMystery (0.999)	5	0	0	0	2	0	0	1	2	0	2	2	1	2	0	2	0	2	2	0	1	2	2		
Rovers (0.99)	20	4	7	7	12	11	8	10	12	12	12	15	9	12	6	11	7	7	13	10	14	8	19		
TPP (0.5)	15	9	10	9	14	12	9	15	15	14	13	14	14	15	14	11	9	11	15	4	15	0	9		
TPP (0.99)	15	6	6	6	5	3	3	9	6	5	5	10	9	10	4	6	5	6	9	1	14	0	6		
$\Sigma$	70	21	25	24	40	33	25	43	44	36	41	50	41	48	29	39	23	35	48	19	52	18	44		
$\Sigma$ UIPC	336	120	128	127	107	96	91	113	118	193	126	115	140	167	193	114	125	127	139	56	52	28	131		
Unsolvable Resource-Constrained Planning (RCP) Benchmarks (Nakhost et al., 2012)																									
NoMystery (0.5)	30	14	25	25	30	30	30	30	30	30	30	30	30	30	29	30	25	29	30	29	30	30	27		
NoMystery (0.6)	30	2	15	15	30	28	27	30	30	20	29	30	30	30	12	30	15	25	30	28	30	30	27		
NoMystery (0.7)	30	0	5	7	29	23	21	29	30	5	26	30	29	30	2	28	6	20	30	22	28	30	29		
NoMystery (0.8)	30	0	0	0	26	18	11	26	28	2	18	30	26	30	0	21	0	15	30	17	25	30	30		
NoMystery (0.9)	30	0	0	0	14	10	7	24	25	1	14	29	24	30	0	13	0	10	25	8	18	29	29		
$\Sigma$ NoMystery	150	16	45	47	129	109	96	139	143	58	117	149	139	150	43	122	46	99	145	104	131	149	142		
Rovers (0.5)	30	1	3	5	30	30	29	30	30	30	30	30	29	30	1	24	4	14	29	26	20	6	30		
Rovers (0.6)	30	0	2	2	30	26	27	25	30	30	30	29	25	28	0	19	2	10	26	26	16	4	30		
Rovers (0.7)	30	0	0	0	30	25	26	23	30	30	30	29	23	19	0	9	0	3	21	26	12	0	30		
Rovers (0.8)	30	0	0	0	29	24	24	21	30	29	29	24	21	13	0	5	0	2	13	21	8	0	30		
Rovers (0.9)	30	0	0	0	24	17	20	13	26	24	24	16	13	6	0	1	0	0	9	16	7	0	30		
$\Sigma$ Rovers	150	1	5	7	143	122	126	112	146	143	143	128	111	96	1	58	6	29	98	115	63	10	150		
TPP (0.5)	5	2	4	4	5	5	5	5	5	5	5	5	5	5	5	5	2	5	5	0	5	0	1		
TPP (0.6)	5	0	1	1	5	3	3	5	5	4	2	5	5	5	4	5	0	1	5	0	5	0	0		
TPP (0.7)	5	0	0	0	2	2	0	3	5	1	1	5	3	5	1	4	0	0	3	0	5	0	0		
TPP (0.8)	5	0	0	0	1	0	0	0	0	0	1	1	1	1	4	1	2	0	0	1	0	5	0		
TPP (0.9)	5	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	5	0	0		
$\Sigma$ TPP	25	2	5	5	13	10	8	13	15	11	9	16	14	20	11	16	2	6	14	0	25	0	1		
$\Sigma$ RCP	325	19	55	59	285	241	230	264	304	212	269	293	264	266	55	196	54	134	257	219	219	159	293		
$\Sigma$ Total	661	139	183	186	392	337	321	377	422	405	395	408	404	433	248	310	179	261	396	275	271	187	424		

Table 6.6.: Unsolvability benchmarks. Coverage results (number of instances proved unsolvable), comparing to the state of the art. Best results are highlighted in **bold**. Abbreviations: “Bli” blind search; “ $u^1$ ” search with  $u^1$  dead-end detection; “MSP” perfect merge-and-shrink abstraction; “PDB” Aidos’ dead-end PDB heuristic; “BDD” BDD-based search in SYMPA; “PDR” property-directed reachability; “Res” resource-variable detection; “Star” star-topology decoupled search; “RB” red-black state space search. Other abbreviations as before.

pruning and property-directed reachability work well on resource-constrained domains, yet still worse than the other competitors. The latter applies also to resource-variable detection. Irrelevance pruning is typically detrimental here, with benefits mainly in Diagnosis, DocTransfer, and NoMystery.

Comparing the UIPC'16 algorithms to our techniques, there is only a single strong case for conflict learning on the non-resource domains, DFS with relaxed-plan refinement in Diagnosis. On the resource-constrained domains though, our techniques, especially neighbors refinement, are competitive. On Nakhost et al.'s (2012) RCP benchmarks, DFS with neighbors refinement is second in overall coverage only to approximate merge-and-shrink and red-black state space search, both completely unrelated algorithms. It beats approximate merge-and-shrink in Rovers, and it beats red-black state space search in TPP.

### Combination with Other Unsolvability Detectors

In difference to the solvable benchmarks discussed above, on the unsolvability benchmarks, combinations of our learning methods with other dead-end detectors exhibit considerable synergy. This is most pronounced for the “NP” combination with merge-and-shrink, using neighbors refinement or path-cut refinement depending on the situation. This combination outperforms its components in each of the Nakhost et al. domains, and it has the best overall coverage on these domains, of all the algorithms tested here.

For the other combinations, the synergy is weaker. The only case where the combination outperforms its components is “N” with merge-and-shrink in UIPC'16 TPP with  $\mathfrak{C} = 0.5$ . Across domains, though, almost all combinations exhibit more consistent strength than their components. Indeed, all combinations except “N” with merge-and-shrink dominate their components in overall coverage.

### Offline versus Online Conjunction Learning

Table 6.7 shows the coverage results of online versus offline conjunction learning. Regarding unbounded offline learning, where unsolvability is proved without search on the initial state, the data shows that this has very little merit compared to online learning. Each of path-cut refinement and relaxed-plan refinement is almost consistently dominated by the respective online learning method, the only noteworthy exceptions being BagTransport, PegSol, and NoMystery. Comparing to neighbors refinement which is not available in the offline context, the only strong cases for offline refinement are BagTransport for offline path-cut refinement, and Diagnosis for offline relaxed-plan refinement. In all cases, the online learning approaches are superior in overall coverage.

For bounded offline learning, i.e., static  $\mathcal{U}^C$  dead-end detectors, the picture changes dramatically on the non-resource UIPC'16 benchmarks. This is, however, simply due to the size bound, which avoids the slow-down incurred by learning too many conjunctions – all the bounded offline learning configurations are dominated consistently here by DFS without any learning at all. Furthermore, on the resource-constrained benchmarks, the bounded offline learning configurations are dominated, typically outperformed, by their online learning counterparts. The single exception to the latter is the UIPC TPP domain with  $\mathfrak{C} = 0.99$ , where the static dead-end detectors do better. Overall, the picture is quite clearly in favor of search with online learning.

### Search Reduction

Similarly to our discussion of solvable benchmarks above, we next provide a view of search space sizes under our different DFS learning techniques, with neighbors refinement as the comparison baseline. Fig-

Domain	#	Base		DFS (No / Online)				DFS (Offline)				Init	
		Bli	$\mathcal{U}^1$	No	Nei	Pat	RP	$\alpha = 2$		$\alpha = 32$		$\alpha = \infty$	
								Pat	RP	Pat	RP	Pat	RP
Unsolvability IPC (UIPC) 2016 Benchmarks													
BagBarman	20	12	4	4	0	0	0	4	4	0	4	0	0
BagGripper	25	5	3	3	0	0	0	2	2	0	0	0	0
BagTransport	29	6	6	6	5	5	6	5	6	3	6	9	4
Bottleneck	25	9	20	21	9	13	14	20	21	18	21	13	14
CaveDiving	25	7	7	7	5	4	4	7	7	6	7	3	1
ChessBoard	23	5	5	5	2	1	1	5	5	4	5	1	1
Diagnosis	20	6	7	7	9	5	12	5	7	5	7	4	12
DocTransfer	20	5	7	6	4	5	8	6	6	6	6	5	5
PegSol	24	24	24	24	14	12	4	24	24	24	24	14	4
PegSolRow5	15	5	5	5	4	3	2	5	5	4	5	4	3
SlidingTiles	20	10	10	10	10	10	10	10	10	10	10	0	0
Tetris	20	5	5	5	5	5	5	5	5	5	5	0	0
$\Sigma$	266	99	103	103	67	63	66	98	102	85	100	53	44
NoMystery (0.4)	5	2	2	2	3	4	3	2	2	2	2	5	4
NoMystery (0.85)	5	0	0	0	2	2	2	0	0	0	0	2	2
NoMystery (0.99)	5	0	0	0	2	1	0	0	0	0	0	0	1
NoMystery (0.999)	5	0	0	0	2	0	0	0	0	0	0	0	1
Rovers (0.99)	20	4	7	7	12	11	8	7	7	6	7	3	7
TPP (0.5)	15	9	10	9	14	12	9	10	9	10	9	12	9
TPP (0.99)	15	6	6	6	5	3	3	5	6	5	6	3	2
$\Sigma$	70	21	25	24	40	33	25	24	24	23	24	25	26
$\Sigma$ UIPC	336	120	128	127	107	96	91	122	126	108	124	78	70
Unsolvable Resource-Constrained Planning (RCP) Benchmarks (Nakhost et al., 2012)													
NoMystery (0.5)	30	14	25	25	30	30	30	25	25	30	25	30	30
NoMystery (0.6)	30	2	15	15	30	28	27	17	15	23	15	30	28
NoMystery (0.7)	30	0	5	7	29	23	21	10	7	14	7	25	22
NoMystery (0.8)	30	0	0	0	26	18	11	0	0	7	0	21	10
NoMystery (0.9)	30	0	0	0	14	10	7	0	0	2	0	12	6
$\Sigma$ NoMystery	150	16	45	47	129	109	96	52	47	76	47	118	96
Rovers (0.5)	30	1	3	5	30	30	29	20	5	25	5	27	28
Rovers (0.6)	30	0	2	2	30	26	27	16	2	23	2	25	26
Rovers (0.7)	30	0	0	0	30	25	26	10	0	14	0	22	26
Rovers (0.8)	30	0	0	0	29	24	24	4	0	10	0	13	17
Rovers (0.9)	30	0	0	0	24	17	20	1	0	6	0	8	9
$\Sigma$ Rovers	150	1	5	7	143	122	126	51	7	78	7	95	106
TPP (0.5)	5	2	4	4	5	5	5	3	4	4	4	5	5
TPP (0.6)	5	0	1	1	5	3	3	1	1	2	1	3	2
TPP (0.7)	5	0	0	0	2	2	0	0	0	1	0	2	0
TPP (0.8)	5	0	0	0	1	0	0	0	0	0	0	1	0
TPP (0.9)	5	0	0	0	0	0	0	0	0	0	0	0	0
$\Sigma$ TPP	25	2	5	5	13	10	8	4	5	7	5	11	7
$\Sigma$ RCP	325	19	55	59	285	241	230	107	59	161	59	224	209
$\Sigma$ Total	661	139	183	186	392	337	321	229	185	269	183	302	279

Table 6.7.: Unsolvability benchmarks. Coverage results, comparing offline versus online conjunction learning methods. Best results are highlighted in **bold**. Abbreviations: “Init  $\alpha = \infty$ ”: offline  $C$ -learning without size bound, proving unsolvability on the initial state. Other abbreviations as before.

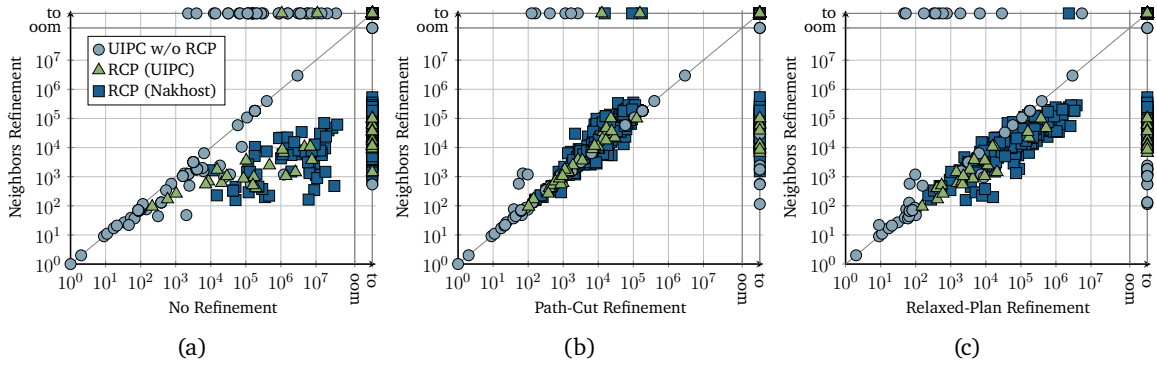


Figure 6.4.: Unsolvability benchmarks. Per-instance DFS search space size (number of visited states) comparison between different learning techniques. We compare neighbors refinement on the  $y$ -axis to (a) no learning, (b) path-cut refinement, respectively (c) relaxed-plan refinement on the  $x$ -axis. “to”: time limit and “oom” memory limit exceeded.

ure 6.4 gives the data.

The main conclusions are very similar to those we made for solvable planning tasks (cf. Figure 6.2). The comparison to no learning in Figure 6.4a shows that learning is often detrimental on the non-resource UIPC’16 benchmarks, yet has dramatic benefits on the resource-constrained ones. Remember that the only reason for this is generalization. Search space size reduction factors provide an impressive view on how dramatic the improvements are. Over those instances commonly solved by DFS without learning and DFS with neighbors refinement, the minimum/geometric mean/maximum reduction factors on the Nakhost et al. domains are 6.7/436.5/37561.5 for NoMystery; 65.0/1286.6/69668.1 for Rovers; and 190.0/711.9/1900.5 for TPP. That is, we get reductions of 2-3 orders of magnitude on average, and even the minimum reductions are of 1-2 orders of magnitude.

The comparison between refinement methods in Figures 6.4b and 6.4c yields, like on the solvable benchmarks, the conclusion that neighbors refinement and path-cut refinement either perform very similarly or are highly complementary, and that tailoring the refinement method to dead-end detection is typically beneficial.

### Performance Analysis

Table 6.8, part (A), shows data assessing the dependency of the conflict-driven learning approach on (1) conflict identification, (2) learning effectiveness, and (3) strong generalization. Regarding (1), on the resource-constrained benchmarks, unsurprisingly, the “# Confl. Id.” data shows that conflicts are identified on (almost) every instance. From the other benchmarks though, on half of the domains ability (1) is not given, i.e., no or not enough learning can take place. Namely, in all Bag-domains, in SlidingTiles, and in Tetris, no conflicts are identified at all; in PegSolRow5 it is almost that bad. In SlidingTiles and Tetris, this is simply because all actions are invertible, so the state space is strongly connected, and the first conflict is identified only after the entire state space is already explored. In the Bag\* domains, in the rare cases where a conflict is identified, the learning is ineffective and prevents the search from terminating successfully.

Regarding (2) and (3), consider the slowdown “Slow.” and search space reduction “Redu.” columns. We should expect good performance if the value for “Redu.” is larger than that for “Slow”, on commonly solved instances. On the resource-constrained benchmarks, this is consistently the case. On the non-resource UIPC,

Domain	#	(A) Analysis			(B) Ablation Study			
		# Confl. Id.	Confl. Id., Common		DFS		DOS	
			Slow.	Redu.	Cl	NoCl	Cl	NoCl
Unsolvability IPC (UIPC) 2016 Benchmarks								
BagBarman	20	0			0	0	0	0
BagGripper	25	0			0	0	3	3
BagTransport	29	0			5	5	5	5
Bottleneck	25	7	29.6	1.9	9	9	9	9
CaveDiving	25	5	23.5	8.3	5	6	6	7
ChessBoard	23	2	169.7	2.1	2	2	2	2
Diagnosis	20	9	2.9	10.4	9	9	9	9
DocTransfer	20	4	17.3	5.7	4	4	5	5
PegSol	24	14	123.7	1.8	14	14	14	14
PegSolRow5	15	3		2.5	4	4	4	4
SlidingTiles	20	0			10	10	10	10
Tetris	20	0			5	5	5	5
Σ	266	44		3.2	67	68	72	73
NoMystery (0.4)	5	3	1.6	76	3	3	3	3
NoMystery (0.85)	5	2			2	2	2	2
NoMystery (0.99)	5	2			2	2	2	2
NoMystery (0.999)	5	2			2	2	2	2
Rovers (0.99)	20	12	4	690.9	12	12	12	12
TPP (0.5)	15	14	4.6	24.4	14	14	14	14
TPP (0.99)	15	5	60.6	43.6	5	5	5	5
Σ	70	40	7.2	89.4	40	40	40	40
Σ UIPC	336	84		10.4	107	108	112	113
Unsolvable Resource-Constrained Planning (RCP) Benchmarks (Nakhost et al., 2012)								
NoMystery (0.5)	30	29	2.6	433.5	30	30	30	30
NoMystery (0.6)	30	30	3	355.6	30	30	30	30
NoMystery (0.7)	30	29	9.9	724.5	29	29	29	29
NoMystery (0.8)	30	26			26	24	25	25
NoMystery (0.9)	30	14			14	14	14	13
Σ NoMystery	150	128	3.3	439.4	129	127	128	127
Rovers (0.5)	30	29	1.3	2460.2	30	30	30	30
Rovers (0.6)	30	29	1.2	254.5	30	30	30	30
Rovers (0.7)	30	30			30	30	30	30
Rovers (0.8)	30	29			29	27	28	27
Rovers (0.9)	30	24			24	24	24	24
Σ Rovers	150	141	1.3	1286.6	143	141	142	141
TPP (0.5)	5	5	6.6	710.3	5	5	5	5
TPP (0.6)	5	5	29.5	718.6	5	5	5	5
TPP (0.7)	5	2			2	2	2	2
TPP (0.8)	5	1			1	1	1	1
TPP (0.9)	5	0			0	0	0	0
Σ TPP	25	13	8.9	711.9	13	13	13	13
Σ RCP	325	282	3.2	521.5	285	281	283	281
Σ Total	661	366		69	392	389	395	394

Table 6.8.: Unsolvability benchmarks. Results for DFS with neighbors refinement. (A) performance analysis and (B) ablation study. Abbreviations as in Table 6.5.

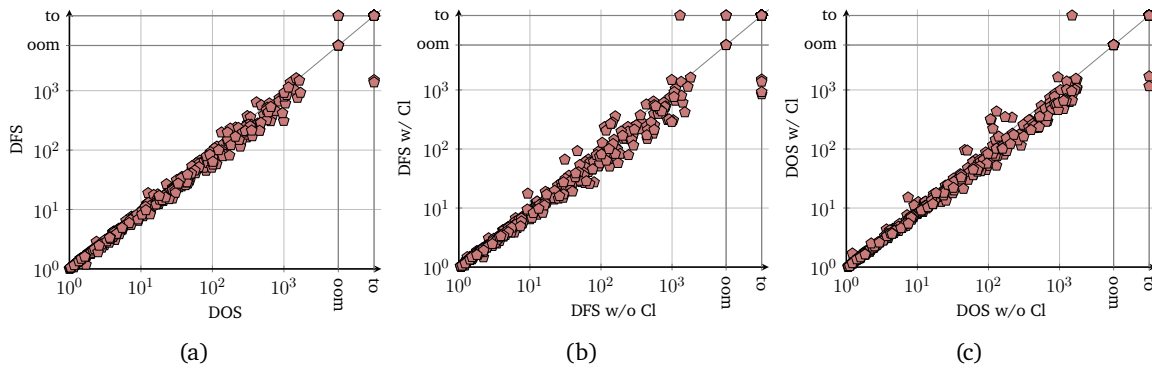


Figure 6.5.: Unsolvability benchmarks. Per-instance runtime (in seconds) comparisons between (a) DOS (x-axis) vs. DFS (y-axis); (a) DFS without clause learning (x-axis) vs. with clause learning (y-axis); (a) DOS without clause learning (x-axis) vs. with clause learning (y-axis). All configurations use neighbors refinement. “to”: time limit exceeded; respectively “oom” memory limit exceeded.

the only domain where the reduction exceeds the slowdown is Diagnosis, precisely the only domain where neighbors refinement improves coverage relative to the baseline. In all other domains where ability (1) is given, the slowdown is much larger than the reduction, to a particularly striking extent in Bottleneck, ChessBoard, and PegSol, precisely the domains where neighbors refinement is most detrimental.

### Ablation Studies

Let us finally consider the ablation study, Table 6.8, part (B). We again fix neighbors refinement, and vary the search algorithm, DFS vs. DOS, as well as switching  $\mathcal{U}^C$  NoGoods on and off. The different configuration settings have little impact on coverage here. DFS does a little worse than DOS on the non-resource UIPC, but a little better on the resource-constrained benchmarks; similarly for clause learning vs. no clause learning. As Figure 6.5 shows, however, both DFS and clause learning are useful algorithm improvements, that rarely hurt runtime performance, while improving it in the most challenging cases.

#### 6.4.4. Generating Unsolvability Certificates

Let us finally consider the generation of unsolvability certificates. An unsolvability certificate must (1) be verifiable in its size; must (2) be feasible to compute; and (3) is useful only if it is *compact*, i.e., loosely speaking, it is much smaller than the state space itself. Conjunction sets, where  $\mathcal{U}^C(I) = \infty$  qualify for (1). But how feasible is it to compute them, and how compact are they?

All our online learning variants guarantee to terminate with  $\mathcal{U}^C(I) = \infty$ , provided the early stopping option is switched off. In difference to the previous section, we now consider this setting here. For comparison, we run the two unbounded offline learning variants. Table 6.9 shows the data.

We consider DFS with neighbors refinement, the overall most effective refinement method. Our main interest lies in comparing this online learning method to offline learning. We include both refinement methods applicable to the latter purpose. We include a comparison to DFS with (neighbors refinement and) early stopping, not producing an unsolvability certificate, to assess the overhead incurred by the final refinement step when the search space is already empty. We include the baselines only for reference.

Consider first coverage, part (A) of Table 6.9, measuring how effective the three different strategies are at

Domain	#	(A) Coverage						(B) Geomean $ C $			(C) Geomean ratio $ C / S $		
		Base		DFS Nei		Offline		DFS	Offline		DFS	Offline	
		Bli	$\mathcal{U}^1$	Cer	Cer	Pat	RP	Nei	$\alpha = \infty$		Nei	$\alpha = \infty$	
Unsolvability IPC (UIPC) 2016 Benchmarks													
BagBarman	20	12	4	0	0	0	0						
BagGripper	25	5	3	0	0	0	0						
BagTransport	29	6	6	5	2	9	4	509.0	13156.0	3152.0	1 : 12.6	2.1 : 1	1 : 2.0
Bottleneck	25	9	20	9	9	13	14	829.7	144.5	19.8	1 : 24.0	1 : 137.6	1 : 1003.2
CaveDiving	25	7	7	5	6	3	1	66.0	15.0	85.0	1 : 1.5	1 : 6.4	1 : 1.1
ChessBoard	23	5	5	2	2	1	1	1463.0	9220.0	691.0	2.8 : 1	17.4 : 1	1.3 : 1
Diagnosis	20	6	7	9	8	4	12	26.8	392.5	1274.0	1 : 11006.4	1 : 751.3	1 : 231.5
DocTransfer	20	5	7	4	5	5	5	7417.0	515.0	17.0	1 : 1121.7	1 : 16154.9	1 : 489398.2
PegSol	24	24	24	14	14	14	4	386.0	901.5	747.9	1.2 : 1	2.8 : 1	2.3 : 1
PegSolRow5	15	5	5	4	4	4	3				1 : 5.7	1 : 5.1	1 : 5.1
SlidingTiles	20	10	10	10	0	0	0						
Tetris	20	5	5	5	0	0	0						
$\Sigma$	266	99	103	67	50	53	44				1 : 29.4	1 : 27.7	1 : 61.7
NoMystery (0.4)	5	2	2	3	3	5	4	509.2	737.7	598.8	1 : 23218.1	1 : 16176.2	1 : 13587.5
NoMystery (0.85)	5	0	0	2	2	2	2	3534.4	103531.7	9378.2			
NoMystery (0.99)	5	0	0	2	2	0	1						
NoMystery (0.999)	5	0	0	2	2	0	1						
Rovers (0.99)	20	4	7	12	12	3	7	85.7	3888.6	184.3	1 : 9107.5	1 : 211.6	1 : 4914.1
TPP (0.5)	15	9	10	14	14	12	9	416.3	413.7	819.9	1 : 79.9	1 : 77.1	1 : 40.7
TPP (0.99)	15	6	6	5	5	3	2	2399.4	4993.0	3411.4	1 : 11.6	1 : 5.6	1 : 8.2
$\Sigma$	70	21	25	40	40	25	26	504.3	1500.8	925.6	1 : 268.4	1 : 131.4	1 : 147.2
$\Sigma$ UIPC	336	120	128	107	90	78	70				1 : 69.5	1 : 50.7	1 : 86.5
Unsolvable Resource-Constrained Planning (RCP) Benchmarks (Nakhost et al., 2012)													
NoMystery (0.5)	30	14	25	30	30	30	30	216.1	390.8	379.8	1 : 25760.1	1 : 22387.8	1 : 23410.2
NoMystery (0.6)	30	2	15	30	30	30	28	516.9	1304.3	939.8	1 : 86197.9	1 : 25389.3	1 : 53497.6
NoMystery (0.7)	30	0	5	29	29	25	22	1180.6	2957.6	2804.8			
NoMystery (0.8)	30	0	0	26	26	21	10	2694.1	5648.3	4260.0			
NoMystery (0.9)	30	0	0	14	14	12	6	3262.3	12782.0	10929.4			
$\Sigma$ NoMystery	150	16	45	129	129	118	96	600.2	1362.9	1163.9	1 : 29958.2	1 : 22742.6	1 : 25958.0
Rovers (0.5)	30	1	3	30	30	27	28	105.1	121.7	90.9	1 : 596322.0	1 : 67191.2	1 : 280622.1
Rovers (0.6)	30	0	2	30	30	25	26	211.3	181.3	166.7			
Rovers (0.7)	30	0	0	30	30	22	26	406.1	800.3	427.4			
Rovers (0.8)	30	0	0	29	29	13	17	588.4	460.9	265.5			
Rovers (0.9)	30	0	0	24	24	8	9	963.1	502.9	755.5			
$\Sigma$ Rovers	150	1	5	143	143	95	106	257.0	281.6	207.2	1 : 596322.0	1 : 67191.2	1 : 280622.1
TPP (0.5)	5	2	4	5	5	5	5	1914.8	4023.9	2490.9	1 : 1855.8	1 : 249.8	1 : 680.2
TPP (0.6)	5	0	1	5	5	3	2	2742.0	3214.1	3270.9			
TPP (0.7)	5	0	0	2	3	2	0						
TPP (0.8)	5	0	0	1	1	1	0						
TPP (0.9)	5	0	0	0	0	0	0						
$\Sigma$ TPP	25	2	5	13	14	11	7	2121.7	3773.6	2692.5	1 : 1855.8	1 : 249.8	1 : 680.2
$\Sigma$ RCP	325	19	55	285	286	224	209	419.2	665.4	525.6	1 : 26165.4	1 : 14975.2	1 : 20053.9
$\Sigma$ Total	661	139	183	392	376	302	279				1 : 539.2	1 : 362.0	1 : 567.8

Table 6.9.: Unsolvability benchmarks. Results for generating unsolvability certificates  $C$ , comparing online learning with DFS and neighbors refinement to offline learning with path-cut respectively relaxed-plan refinement. Best results among these three strategies are highlighted in **bold**. (A) coverage, i.e., number of instances for which a certificate was generated. (B) Geometric mean of certificate size  $|C|$ . (C) Compactness relative to the (reachable) state space size  $|S|$ ; an entry “1 :  $x$ ” means that, in the geometric mean,  $|C|$  is  $x$  times smaller than  $|S|$ , an entry “ $x$  : 1” means that, in the geometric mean,  $|C|$  is  $x$  times larger than  $|S|$ . Abbreviations: “w/o Cer” early stopping option enabled, i.e., not generating certificates; “w/ Cer” early stopping option disabled. Other abbreviations as before.

producing unsolvability certificates. Neighbors refinement is clearly best overall, and it consistently dominates on the resource-constrained benchmarks. On the non-resource benchmarks though, the offline methods are competitive, path-cut refinement even better overall, with substantial advantages in BagTransport, Bottleneck, and Diagnosis.

Observe that, with early stopping, neighbors refinement “beats” the offline methods on the non-resource UIPC, while without early stopping it does not. The advantage of neighbors refinement without early stopping here mainly stems from the SlidingTiles and Tetris domains, which is simply due to the aforementioned fact that no learning takes place here (cf. Table 6.8). So, the superiority of DFS with early stopping here is one of search, not of learning. On the resource-constrained benchmarks, switching early stopping off does not have any adverse impact on coverage.

Consider now part (B) of Table 6.9, giving a view of absolute certificate size (while (C) is relative to state space size). In PegSolRow5 there is no data as, on the commonly covered instances,  $\mathcal{U}^1(I) = \infty$  so nothing needs to be done. Online learning with neighbors refinement is superior on the resource-constrained benchmarks, except for Nakhost et al. Rovers (and one case of UIPC TPP), where the picture is more mixed. On the non-resource UIPC benchmarks, the methods are complementary, with differing strengths depending on the domain. The small certificate sizes here, in the order of 100s or 1000s of conjunctions, are merely due to the size of the commonly solved instances. On the largest instances solved by DFS with neighbors refinement, the certificates have 10000s of conjunctions.

By definition, the relative performance of learning methods is the same in parts (C) and (B). What’s remarkable in (C) is that the certificates found often are extremely compact, several orders of magnitude smaller than the state space itself. This is especially pronounced in the resource-constrained benchmarks, but also happens in some of the other domains, most notably in Bottleneck, Diagnosis, and DocTransfer.

## 7. LP Heuristics Over Conjunctions: Compilation, Convergence & Conflict Refinement

Linear programming is a popular method to derive admissible cost-to-goal bounds in classical and probabilistic planning (e.g., van den Briel et al., 2007; Bonet, 2013; Bonet and van den Briel, 2014; Pommerening et al., 2014; Trevizan et al., 2017b). Underlying many LP-based heuristics is the so called *state equation* (Bonet, 2013), a system of linear inequalities over action execution counts necessarily satisfied by every plan. Another successful variant are *potential heuristics* (Pommerening et al., 2015), which estimate the cost-to-goal via a linear combination of real-valued *state-feature weights*. The computation of weights defining admissible potential heuristics can be cast as an LP. Proved by the success in the latest *Unsolvability International Planning Competition* (UIPC’16), LP heuristics also often constitute strong unsolvability detectors (Seipp et al., 2016). In the following, we explore the use of *LP heuristics over fact conjunction* as an alternative to the critical-path dead-end detector  $u^C$  in our conflict-driven learning procedure.

Fact conjunctions are a popular method to improve heuristic approximations. In the previous chapter, we have shown that fact conjunctions constitute an effective means for refining the critical-path heuristic towards recognizing dead ends. Haslum (2012) and Keyder et al. (2014) have considered conjunctions to improve delete relaxation heuristics (Hoffmann and Nebel, 2001). They showed that the compilation  $\Pi^C$ , rendering a given set of conjunctions  $C$  *explicit*, forces delete relaxed plans to *converge* to real plans in the limit, i.e., for suitable  $C$ , the perfect delete-relaxation heuristic  $h^+$  applied to  $\Pi^C$  renders equal to the perfect heuristic  $h^*$ . This leads to powerful heuristics that allow interpolate freely between computational cost and accuracy of the estimates.

Fact conjunctions have also been considered for enhancing LP heuristics. van den Briel et al. (2007) and Bonet and van den Briel (2014) considered (*partial*) *variable merges* to enhance the state-equation heuristic via constraints over conjunctions. Seipp et al. (2016) examined the size of conjunctions needed for a potential heuristic to find a plan without search. Pommerening et al. (2017) designed potential heuristics over arbitrary sets of conjunctions. In particular, they provided an efficient construction method for potential heuristics over fact pairs, but they also showed that the construction over fact triples is already computationally hard.

This chapter explores further the combination of LP heuristics and conjunctions, contributing new results pertaining to the use of the  $\Pi^C$  compilation, to convergence properties, and to the exploitation of convergence for conflict-driven learning. We show that for tasks in transition normal form (Pommerening and Helmert, 2015), partial variable merges are strictly dominated by the compilation  $\Pi^C$ , and that both render the state-equation heuristic equal to  $h^*$  for suitable  $C$ . We show that the dual state-equation LP on  $\Pi^C$  yields admissible conjunction potential heuristics. Note that this does not contradict Pommerening et al.’s (2017) hardness result, due to  $\Pi^C$ ’s exponential worst-case growth in  $|C|$ . We show that, together with a (trivial to compute) upper bound  $U^*$  on  $h^*(s)$  for solvable  $s$ , one can choose  $C$  so that the  $\Pi^C$  potential heuristic equals  $h^*$ . Finally, we exploit these properties for designing a conflict refinement method, choosing new conjunctions suitable for the state equation, and hence also the potential heuristic, to recognize

a previously unrecognized dead end.

This chapter is based on (Steinmetz and Hoffmann, 2018). Section 7.1 presents some basic notions, and introduces the  $\Pi^C$  compilation for FDR tasks. In Section 7.2, we begin our analysis with the state-equation heuristic, recall its definition from literature, compare its extensions to conjunctions via partial variable merges and via  $\Pi^C$ , as well as prove convergence of both to  $h^*$ . In Section 7.3, we consider potential heuristics. We show how to leverage the  $\Pi^C$  compilation to generate admissible potential heuristics over arbitrary conjunctions, how to use potential heuristics for unsolvability detection, and we provide the obligatory convergence result. Section 7.4 spells out the conflict analysis and refinement algorithm. In Section 7.5, we compare the LP heuristics applied to  $\Pi^C$  and the critical-path unsolvability detector  $u^C$  in terms of their *learning effectiveness*, i.e., comparing how much additional information they require to recognize the same dead-end state. Section 7.6 concludes this chapter with a brief experimental evaluation.

## 7.1. Preliminaries

Throughout this chapter, we assume classical planning tasks in FDR notation (cf. Definition 2.4). Let  $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  be such a task. We need the notion of FDR *regression*, defined in the standard way:

**Definition 7.1** (FDR Regression). *Let  $P$  be a variable assignment, and let  $a \in \mathcal{A}$  be an action. If  $\text{eff}(a) \cap P \neq \emptyset$ , and  $\text{eff}(a) \parallel P$ , and  $(P \setminus \text{eff}(a)) \parallel \text{pre}(a)$ , then the **regression** of  $P$  over  $a$  is defined as follows:*

$$\text{regress}(P, a) = (P \setminus \text{eff}(a)) \cup \text{pre}(a)$$

Otherwise, the regression is undefined, and we write  $\text{regress}(P, a) = \perp$ .

Note that the main difference between this and Definition 6.1 is the additional condition  $(P \setminus \text{eff}(a)) \parallel \text{pre}(a)$ , which is needed for  $\text{regress}(P, a)$  to be a proper variable assignment. The set of actions for which the regression is defined is denoted again by  $\mathcal{A}[P] = \{ a \in \mathcal{A} \mid \text{regress}(P, a) \neq \perp \}$ .

Sometimes, we will require  $\Pi$  to be in *transition normal form* (Pommerening and Helmert, 2015):

**Definition 7.2** (Transition Normal Form). *An FDR task  $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  is in **transition normal form**, short **TNF**, if  $\Pi$  satisfies*

$$\text{(TNF1)} \quad \text{vars}(\text{eff}(a)) \subseteq \text{vars}(\text{pre}(a)), \text{ for all actions } a \in \mathcal{A}, \text{ and}$$

$$\text{(TNF2)} \quad \text{vars}(\mathcal{G}) = \mathcal{V}.$$

The condition (TNF1) differs slightly from the requirement  $\text{vars}(\text{pre}(a)) = \text{vars}(\text{eff}(a))$  of the original definition (Pommerening and Helmert, 2015). We do so for simplicity only. All our results apply directly to the original version as well. Given any FDR task  $\Pi$ , it is possible to construct an “equivalent” TNF task  $\Pi^{\text{TNF}}$  in time polynomial in the size of  $\Pi$ , specifically  $\Pi^{\text{TNF}}$  such that the plans of  $\Pi$  and  $\Pi^{\text{TNF}}$  are polynomially transformable into one another.

We next introduce the  $\Pi^C$  compilation. We follow Haslum (2012), with small modifications suiting our context. As in Section 6.1, a *conjunction*  $c$  is a variable assignment (set of facts). To represent a set  $C$  of conjunctions explicitly in a given task  $\Pi$ , the  $\Pi^C$  compilation introduces a new Boolean variable  $\pi_c$  for each  $c \in C$ ; abusing notation, we identify  $\pi_c$  with the fact  $\pi_c \mapsto \top$ . For a variable assignment  $P$ ,

$$P^C = P \cup \{ \pi_c \mapsto \top \mid c \in C, c \subseteq P \} \cup \{ \pi_c \mapsto \perp \mid c \in C, c \not\subseteq P \}$$

augments  $P$  with the conjunctions it contains as well as the negation of the conjunctions it conflicts with. We say that a set of conjunctions  $C \subseteq \mathcal{C}$  is *consistent* if all  $c, c' \in C$  are pairwise consistent. With these notations in hand, we can define the compiled task:

**Definition 7.3** (FDR  $\Pi^C$  Compilation). *Let  $C$  be a set of conjunctions. The  $\Pi^C$  **compilation** is given by the FDR task  $\Pi^C = \langle \mathcal{V}^C, \mathcal{A}^C, \mathcal{I}^C, \mathcal{G}^C \rangle$ , where*

- $\mathcal{V}^C = \mathcal{V} \cup \{ \pi_c \mid c \in C \}$  with  $\mathcal{D}_{\pi_c} = \{ \top, \perp \}$ ,
- $\mathcal{A}^C$  contains an action  $a^C$  for every pair of  $a \in \mathcal{A}$  and consistent  $C \subseteq \mathcal{C}$  such that

(i)  $a \in \mathcal{A}[c]$ , for all  $c \in C$ , and

(ii) for every  $c' \in \mathcal{C} \setminus C$ , either  $a \notin \mathcal{A}[c']$  or  $\text{regress}(c', a) \not\subseteq \bigcup_{c \in C} \text{regress}(c, a)$ .

Overloading notation, denote by  $\text{regress}(C, a) = \bigcup_{c \in C} \text{regress}(c, a)$  the union of the regressions over the individual conjunctions. The action  $a^C$  is given by

$$(1) \text{ pre}(a^C) = [\text{regress}(C, a)]^C$$

$$(2) \text{ eff}(a^C) = \text{eff}(a) \cup \{ \pi_c \mapsto \top \mid c \in C \} \cup \{ \pi_{c'} \mapsto \perp \mid c' \in C, c' \not\parallel \text{regress}(C, a), c' \not\parallel \text{eff}(a) \}$$

$$(3) \text{ c}(a^C) = \text{c}(a)$$

Intuitively,  $a^C$  represents an *occurrence* of  $a$  that makes all  $c \in C$  true. For this to happen, the regression of each  $c$  over  $a$  must be true beforehand, as per (1), a conjunction  $c'$  potentially invalidated by  $a$  is false afterwards. Condition (ii) assures consistency: if an occurrence  $a^C$  always makes true a conjunction  $c$ , then  $\pi_c$  must be set to true necessarily.

With the possible  $C$  being subsets of  $\mathcal{C}$ ,  $|\mathcal{A}^C|$  may grow exponentially in  $|C|$ . This can be ameliorated (but not overcome entirely) with *mutex information* (Keyder et al., 2014). Compatibility of  $C$  as postulated here is a special case thereof.

Note that the singleton conjunction from  $C$  do not need to be represented explicitly, as  $\Pi^C$  takes over all variables from  $\Pi$ , i.e., denoting by  $C_{>1} \subseteq C$  the subset of non-singleton conjunctions, it suffices to construct  $\Pi^{C_{>1}}$  instead of  $\Pi^C$ . For the sake of simplicity, whenever we write  $\Pi^C$  in the following, we refer to  $\Pi^{C_{>1}}$ .

Following previous works on the  $\Pi^C$ -compilation, we interpret heuristics for  $\Pi^C$ , written  $h[\Pi^C]$ , as heuristics for  $\Pi$  by mapping every state  $s$  of  $\Pi$  to the state in  $\Pi^C$  obtained by augmenting  $s$  with the  $\pi_c$ -variable assignments according to whether or not  $c$  is satisfied in  $s$ , i.e., we define  $h[\Pi^C](s) = h[\Pi^C](s^C)$ .

Plan equivalence between  $\Pi$  and  $\Pi^C$  can be easily shown by adapting Haslum's (2012) proof to our slightly modified  $\Pi^C$  definition. An extension of this equivalence result to individual transitions will become handy later on:

**Lemma 7.1.** *Let  $h$  be a heuristic for  $\Pi^C$ . If  $h$  is consistent in  $\Pi^C$ , then  $h$  is consistent in  $\Pi$ .*

*Proof.* Let  $s$  be any state of  $\Pi$ , and  $a \in \mathcal{A}(s)$  be any action applicable in  $s$ . Define  $C = \{c \in \mathcal{C} \mid c \subseteq s[a], c \cap \text{eff}(a) \neq \emptyset\}$ . Obviously,  $C$  is consistent; for every  $c \in C$ , it holds that  $\text{regress}(c, a) \neq \perp$ ; and for every  $c' \in (\mathcal{C} \setminus C)$ , either  $\text{regress}(c', a) = \perp$  or  $\text{regress}(c', a) \not\subseteq s$  and thus  $\text{regress}(c', a) \not\subseteq \bigcup_{c \in C} \text{regress}(c, a)$ . Hence,  $a^C \in \mathcal{A}^C$ . Moreover, the selection of  $C$  ensures that  $a^C$  is applicable in  $s^C$  and that for every  $c \subseteq s[a]$ ,  $s^C[a^C][\pi_c] = \top$ . For every  $c \not\subseteq s[a]$ , it holds that  $c \notin C$  and either  $s^C[\pi_c] = \perp$ ; or  $c \subseteq s$  and  $c \not\parallel \text{eff}(a)$ , i.e.,  $\text{eff}(a^C)[\pi_c] = \perp$ . In both cases,  $s^C[a^C][\pi_c] = \perp$  follows.

We get  $h(s) = h(s^C) \leq h(s^C \llbracket a^C \rrbracket) + c(a^C) = h((s \llbracket a \rrbracket)^C) + c(a^C) = h(s \llbracket a \rrbracket) + c(a)$ , what shows the claim.  $\square$

## 7.2. The State-Equation Heuristic

We begin our analysis with the state-equation heuristic. Section 7.2.1 provides the formal definition. In Section 7.2.2, we then compare *partial variable merges* and the  $\Pi^C$  compilation. We show that the state-equation heuristic applied to  $\Pi^C$  dominates its extension by the corresponding partial variable merges, for tasks in TNF; and that the state-equation heuristic converges to  $h^*$  via appropriate partial variable merges, hence so via  $\Pi^C$ .

### 7.2.1. Definition

The *state equation* (SEQ) (Bonet, 2013) describes a relation between variable-value changes, the *net-changes*, that every plan must satisfy. A fact  $p = v \mapsto d$  is **produced** by an action  $a$  if  $\text{eff}(a)[v] = d$ ;  $p$  is **consumed** by  $a$  if  $\text{pre}(a)[v] = d$  and  $v \in \text{vars}(\text{eff}(a))$ . We denote by  $\text{PROD}(p)$  and  $\text{CONS}(p)$  the set of all actions that produce, respectively consume,  $p$ . Let  $s$  be any state,  $\pi$  be any plan for  $s$ , and  $p$  be any fact. Every consumption of  $p$  along  $\pi$  requires its production beforehand. If  $p$  is true in  $s$ , then  $p$  can be consumed once more than it is produced. If  $p$  must be true after the application of  $\pi$ , then  $p$  must be produced more often than it is consumed. Denote by  $\text{COUNT}_a^\pi$  the number of occurrences of  $a$  in  $\pi$ . Writing down this relation formally, one obtains

**Definition 7.4** (State Equation). *Let  $s$  be a state,  $\pi$  be a plan for  $s$ , and  $p$  be a fact. The **state equation**, short **SEQ**, for  $p$  is*

$$\sum_{a \in \text{PROD}(p)} \text{COUNT}_a^\pi - \sum_{a \in \text{CONS}(p)} \text{COUNT}_a^\pi \geq \Delta_p(s) \quad (7.1)$$

where

$$\Delta_p(s) = \begin{cases} 1, & \text{if } p \notin s \text{ and } p \in \mathcal{G} \\ -1, & \text{if } p \in s \text{ and } p \notin \mathcal{G} \\ 0, & \text{otherwise} \end{cases}$$

The *state-equation heuristic*  $h^{\text{seq}}$  is defined via an LP. The LP contains one variable  $\text{COUNT}_a \in \mathbb{R}_0^+$  for every action  $a$ . For every fact  $p$ , the LP contains the constraint given by Equation (7.1) for  $p$ , choosing the right hand side  $\Delta_p(s)$  according to the state  $s$  for which  $h^{\text{seq}}(s)$  is being computed. The objective function is to minimize  $\sum_{a \in \mathcal{A}} \text{COUNT}_a \cdot c(a)$ . We denote this LP as  $\text{SEQ}(s)$ . If  $\text{SEQ}(s)$  has an optimal solution, then  $h^{\text{seq}}(s)$  gives the respective the objective value. Otherwise,  $h^{\text{seq}}(s) = \infty$ . As every plan must satisfy Equation (7.1) for every  $p$ ,  $h^{\text{seq}}$  is admissible. In fact, it is straightforward to show that  $h^{\text{seq}}$  is goal-aware and consistent.  $h^{\text{seq}}$  is goal-aware, because  $\Delta_p(s_*) \leq 0$  for all facts  $p$  and goal states  $s_*$ , so setting every variable  $\text{COUNT}_a$  to 0 trivially satisfies the state-equation constraints, resulting in  $h^{\text{seq}}(s_*) = 0$  as desired. To show consistency, suppose  $s$  is an arbitrary state, and  $a \in \mathcal{A}(s)$  is an arbitrary action applicable in  $s$ . Observe that one can obtain from every optimal solution to  $\text{SEQ}(s \llbracket a \rrbracket)$ , a solution to  $\text{SEQ}(s)$  with objective value  $h^{\text{seq}}(s \llbracket a \rrbracket) + c(a)$  by simply incrementing  $\text{COUNT}_a$  by one. This solution upper bounds the optimal objective value of  $\text{SEQ}(s)$ , hence  $h^{\text{seq}}(s) \leq h^{\text{seq}}(s \llbracket a \rrbracket) + c(a)$ .

We denote the unsolvability detector variant of  $h^{\text{seq}}$  by  $u^{\text{seq}}$ , i.e.,  $u^{\text{seq}}(s) = \infty$  if  $\text{SEQ}(s)$  is infeasible, and  $u^{\text{seq}}(s) = 0$  otherwise. Given that  $h^{\text{seq}}$  is admissible,  $u^{\text{seq}}(s) = \infty$  implies that  $s$  is a dead end, i.e.,  $u^{\text{seq}}$



Figure 7.1.: Simple variants of our rovers running example. (a) the rover needs to move from  $A_2$  to  $B$ , but there is only one energy unit left. (b) the rover needs to collect the sample at  $A_2$  and bring it to  $B$ , without constraints on the energy consumption.

indeed is an unsolvability detector. Since  $h^{\text{seq}}$  is consistent,  $u^{\text{seq}}$  furthermore satisfies the transitivity property. As the main concern in the remainder of this section is the comparison of approaches for improving heuristic estimates in general, we will be referring to  $h^{\text{seq}}$  rather than  $u^{\text{seq}}$  most of the time.

**Example 7.1.** Consider the variant of our rovers running example depicted in Figure 7.1a. The FDR planning task encoding is similar to Example 2.2. The initial state is  $\mathcal{I} = \{\text{rov} \mapsto A_2, \text{bat} \mapsto 1\}$ . The goal is  $\mathcal{G} = \{\text{rov} \mapsto B\}$ . Consider  $\text{Seq}(\mathcal{I})$ . To satisfy the state equation of the goal fact, it must be  $\text{COUNT}_{\text{move}(A_1, B, 1)} \geq 1$ . Since  $\text{move}(A_1, B, 1)$  consumes the fact  $\text{rov} \mapsto A_1$ , but  $\Delta_{\text{rov} \mapsto A_1}(\mathcal{I}) = 0$ , this entails that  $\text{COUNT}_{\text{move}(A_2, A_1, 1)} \geq 1$ . However, both  $\text{move}$  actions also consume  $\text{bat} \mapsto 1$ , which has no producer, and is available initially only once, i.e.,  $-\text{COUNT}_{\text{move}(A_1, B, 1)} - \text{COUNT}_{\text{move}(A_2, A_1, 1)} < -1 = \Delta_{\text{bat} \mapsto 1}(\mathcal{I})$ . Hence,  $\text{Seq}(\mathcal{I})$  is infeasible, and  $u^{\text{seq}}(\mathcal{I}) = \infty$ . Note that, this is in contrast to the critical-path unsolvability detector  $u^1$ , which without additional conjunctions is not able recognize  $\mathcal{I}$  as dead end.

An important weakness of the state-equation heuristic is that *prevail* conditions of actions, i.e., preconditions  $\text{pre}(a)[v]$  where  $v \notin \text{vars}(\text{eff}(a))$ , are disregarded completely.

**Example 7.2.** Consider now the rover example from Figure 7.1b. The corresponding FDR task consists of just two variables: the rover position  $\text{rov}$ , with  $\mathcal{D}_{\text{rov}} = \{B, A_1, A_2\}$ , and the sample position  $\text{samp}$  with  $\mathcal{D}_{\text{samp}} = \{B, A_1, A_2, R\}$ . As before, the rover can move between the locations according to the depicted connections (yet, now, without conditions or effects on the remaining energy); it can collect the sample at the rover's current location; and it can drop the sample at the rover's current location. Suppose that all actions have cost 1. The initial state is given by  $\mathcal{I} = \{\text{rov} \mapsto A_1, \text{samp} \mapsto A_2\}$ . The goal is  $\mathcal{G} = \{\text{samp} \mapsto B\}$ . The  $h^{\text{seq}}$  value for this state is 2, accounting only for collecting and disposing the sample. Rover movements are not counted since collecting and disposing the sample prevail the rover's position.

### 7.2.2. The State Equation over Conjunctions

The weakness just discussed can be addressed by considering net-changes over conjunctions instead of single facts.

**Example 7.3.** Reconsider Example 7.2. Consider the set of conjunctions  $\mathcal{C} = \{c_1, c_2\}$  for  $c_1 = \{\text{rov} \mapsto A_2, \text{samp} \mapsto A_2\}$ , and  $c_2 = \{\text{rov} \mapsto B, \text{samp} \mapsto R\}$ . Collecting the sample at  $A_2$  now requires and consumes  $\pi_{c_1} \mapsto \top$ , and dropping the sample at  $B$  consumes  $\pi_{c_2} \mapsto \top$ . To produce  $\pi_{c_1} \mapsto \top$ , the rover has to move to location  $A_2$ . To produce  $\pi_{c_2} \mapsto \top$ , the rover has to move to  $B$ . Further, to satisfy Equation (7.1) for  $p = \text{rov} \mapsto A_1$ , the rover needs to move back from  $A_2$  to  $B$ . This results in the perfect heuristic value 5; indeed,  $h^{\text{seq}}[\Pi^{\mathcal{C}}](\mathcal{I}) = 5$ .

Bonet and van den Briel (2014) designed *partial variable merges* to augment  $h^{\text{seq}}$  by conjunctions. We now compare this technique to the computation of  $h^{\text{seq}}$  in  $\Pi^{\mathcal{C}}$ , and we show convergence of both to the perfect heuristic  $h^*$  for tasks  $\Pi$  in TNF. For the sake of readability, within this section, we only provide the principle ideas underlying the proofs of our results. The detailed proofs are available in Appendix B.3.

Like the  $\Pi^C$ -compilation, partial variable merges consider a set of conjunctions  $C$ . But conjunctions  $c, c' \in C$  are put into relation only when they instantiate the same variables,  $\text{vars}(c) = \text{vars}(c')$ . Hence the name “partial variable merges”: Bonet and van den Briel start from the simpler idea of pre-merging entire variable subsets  $X \subseteq \mathcal{V}$ , extending  $\Pi$  by a new variable representing this product; they improve over that idea by considering only particular value tuples within the product. As a result, their LP encoding grows polynomially in  $|C|$ , but might lose information relative to  $\Pi^C$ , because no constraints are included *across* (conjunctions over) different variable subsets  $X \neq X'$ .

Specifically, partial variable merges are based on notions of *potential* producers and consumers, i.e., actions whose applications *can* achieve respectively invalidate  $c$ :

$$\begin{aligned} \text{PPROD}(c) &= \{ a \in \mathcal{A} \mid \text{regress}(c, a) \neq \perp \} \\ \text{PCONS}(c) &= \{ a \in \mathcal{A} \mid c \not\parallel \text{eff}(a), c \parallel \text{pre}(a) \} \end{aligned}$$

This complication arises, because the impact of an action  $a$  on a conjunction  $c$  depends on the context in which  $a$  is applied: on the action occurrence.

While  $\Pi^C$  enumerates possible action occurrences, variable merges handle each subset of  $C$ , sharing the same variables  $X$  separately. Denote by  $\Pi|_X$  the projection of  $\Pi$  onto  $X$ . To represent those product states  $P$  of  $\Pi|_X$ , where  $P \notin C$ , an abstract state  $\mathfrak{s}_X$  is introduced. The transitions within  $\Pi|_X$  are abstracted by inserting  $\mathfrak{s}_X$  whenever the start or end state of a transition is not contained in  $C$ . Denote by  $\Theta_X^C$  the corresponding *abstract* transition system. Equation (7.1) for a conjunction  $c$  is then defined by summing over the in- and out-transitions of  $c$ , with a separate occurrence-counter variable  $\text{COUNT}_a^{x \rightarrow x'}$  for every state-changing transition  $\langle x, a, x' \rangle \in \mathcal{T}_{\text{vars}(c)}^C$ , i.e.,

$$\sum_{\langle x, a, c \rangle \in \mathcal{T}_{\text{vars}(c)}^C, x \neq c} \text{COUNT}_a^{x \rightarrow c} - \sum_{\langle c, a, x \rangle \in \mathcal{T}_{\text{vars}(c)}^C, c \neq x} \text{COUNT}_a^{c \rightarrow x} \geq \Delta_c(s) \quad (7.2)$$

$\Delta_c(s)$  is defined similarly to single facts:

$$\Delta_c(s) = \begin{cases} 1, & \text{if } c \not\subseteq s \text{ and } c \subseteq \mathcal{G} \\ -1, & \text{if } c \subseteq s \text{ and } c \not\subseteq \mathcal{G} \\ 0, & \text{otherwise} \end{cases}$$

For tasks in TNF, Equation (7.2) can be characterized exactly via the potential producers and consumers of  $c$ . Namely, notice that if (TNF1) is satisfied, then for every  $c \in C$  and  $a \in \mathcal{A}$ ,  $a$  labels at most one transition in  $\Theta_{\text{vars}(c)}^C$  going into  $c$ , and the set  $\text{PPROD}(c)$  lists exactly those actions for which a transition exists. Moreover, as usual for projections, for each  $c$  and  $a$ ,  $a$  can also label at most one transition in  $\Theta_{\text{vars}(c)}^C$  leaving  $c$ ; and those leaving transitions are exactly identified by  $\text{PCONS}(c)$ . Hence, (7.2) can be written equivalently as

$$\sum_{a \in \text{PPROD}(c)} \text{COUNT}_a^{* \rightarrow c} - \sum_{a \in \text{PCONS}(c)} \text{COUNT}_a^{c \rightarrow *} \geq \Delta_c(s) \quad (7.3)$$

where  $*$  is a placeholder for the corresponding transitions.

The separate transition counters are related back to the main action counters by introducing, for every action  $a$  and every variable set  $X$  glanced by  $C$ , the following *link constraint*:

$$\sum_{\langle x, a, x' \rangle \in \mathcal{T}_X^C, x \neq x'} \text{COUNT}_a^{x \rightarrow x'} \leq \text{COUNT}_a \quad (7.4)$$

For every  $X$  and  $a \in \mathcal{A}$ , for which the effect of  $a$  on the conjunctions  $\text{vars}(c) = X$  is determined uniquely, without additional context information, i.e., for  $a$  that label just a single transition in  $\Theta_X^C$ , the “ $\leq$ ” can be replaced by “ $=$ ” forcing that transition to be taken on every application of  $a$ .

We denote by  $h^{Cseq}$  the heuristic extending  $h^{seq}$  by Equation (7.2) and Equation (7.4) for the conjunctions  $C$ .

**Theorem 7.1.** *For every  $\Pi$  in TNF, every set of conjunctions  $C$ , and every state  $s$ , it holds that*

$$h^{seq}[\Pi^C](s) \geq h^{Cseq}(s)$$

*Proof sketch.* Let  $SEQ[\Pi^C]$  be the LP underlying  $h^{seq}[\Pi^C](s)$ , and  $CSEQ$  that underlying  $h^{Cseq}(s)$ . Every solution to  $SEQ[\Pi^C]$  can be transformed into a solution to  $CSEQ$ , with equal objective value. The proof is technical but straightforward.  $\square$

**Theorem 7.2.** *There exists families of  $\Pi$  and  $C$  s.t., to obtain  $h^{C' seq}(s) \geq h^{seq}[\Pi^C](s)$  for all states  $s$ ,  $C'$  must be exponentially larger than  $C$ .*

*Proof sketch.* This happens, e.g., in a transportation example where  $n$  packages must be transported from  $B$  to  $A$ , and truck-load capacity is 1. In  $h^{seq}[\Pi^C]$ , considering all conjunctions of size up to 3 makes visible that no two packages can be in the truck at the same time, yielding  $h^{seq}[\Pi^C] = h^*$ . The partial variable merges in  $h^{Cseq}$ , however, cannot account perfectly for the interactions across packages unless all of them are considered jointly in the same  $\Pi|_X$ .  $\square$

The proof of Theorem 7.2 exploits the fact that  $\Pi^C$ ’s action occurrences relate conjunctions over *arbitrary* variable sets. Yet, recall that exactly this property is what is also causing the size of  $\Pi^C$  to be exponential in  $|C|$ , and therewith the computation of  $h^{seq}[\Pi^C]$ . As this is in contrast to  $h^{Cseq}$ , which can always be computed in time polynomial in  $|C|$ , Theorems 7.1 and 7.2 may hence be not so surprising. We utilize the flexibility offered by  $\Pi^C$  in our conjunction-set refinement method below.

For general tasks  $\Pi$ , the relation between  $h^{seq}[\Pi^C]$  and  $h^{Cseq}$  is not so clear anymore. Complications stem from actions  $a$  affecting variables  $v$  without precondition on  $v$ .  $\Pi^C$  cannot relate the consumption of  $\pi_c$  of any conjunction  $c$  where  $v \in \text{vars}(c)$  with  $a$ ’s action occurrences. In contrast,  $h^{Cseq}$  can do so by enumerating the missing preconditions through different transitions.

**Example 7.4.** *Consider the task with variables  $v_1$  and  $v_2$  and domains  $\mathcal{D}_{v_1} = \mathcal{D}_{v_2} = \{0, 1, 2\}$ ; actions:  $a_1$  requires  $v_2 \mapsto 1$  and sets  $v_1 \mapsto 1$ ;  $a_2$  requires that  $v_1 \mapsto 1$  and sets  $v_2 \mapsto 1$ , and  $a_3$  with empty precondition and effect  $v_1 \mapsto 2$ ; initial state  $\mathcal{I} = \{v_1 \mapsto 0, v_2 \mapsto 0\}$ ; and goal  $\mathcal{G} = \{v_1 \mapsto 1, v_2 \mapsto 1\}$ . Assume that  $C$  contains all conjunctions.*

*Since the abstract states corresponding to the conjunctions  $\mathcal{I}$  and  $\mathcal{G}$  are not connected, it is not possible to satisfy Equation (7.2) for  $\mathcal{G}$  without violating the state-equation constraint of any other conjunction. Hence,  $h^{Cseq}(\mathcal{I}) = \infty$ .*

*However,  $h^{seq}[\Pi^C](\mathcal{I}) < \infty$ : let  $COUNT_{a_1\{\mathcal{G}\}} = 1$ ,  $COUNT_{a_2\{\mathcal{G}\}} = 1$ ,  $COUNT_{a_3^0} = 1$ . It is straightforward to verify that  $COUNT$  satisfies Equation (7.1) for all facts  $v \mapsto d$  and  $\pi_c$  value-assignments. The action occurrence  $a_3^0$  is required to satisfy the constraint for  $\pi_{\mathcal{I}} \mapsto \perp$ .*

Combining the task from Example 7.4 with the one from the proof of Theorem 7.2 shows the existence of non-TNF  $\Pi$  and  $C$  for which  $h^{\text{seq}}[\Pi^C]$  and  $h^{C\text{seq}}$  are incomparable.

We now turn to convergence properties. It is easy to show that partial variable merges can force  $h^{C\text{seq}}$  to converge.

**Theorem 7.3.** *For every planning task  $\Pi$  with  $\text{vars}(\mathcal{G}) = \mathcal{V}$ , there exists a set of conjunctions  $C$  s.t.  $h^{C\text{seq}} = h^*$ .*

*Proof.* A suitable  $C$  is  $C = \mathcal{S}^\Pi$ , i.e., the set of all states in the given task  $\Pi$ . The LP underlying  $h^{C\text{seq}}$  then boils down to an LP encoding of shortest paths in the state space graph, essentially a special case of the min-cost flow problem. The variables encode state-transition weights. The constraints ensure that the difference between the incoming and outgoing flow at every state is  $\geq 0$ , respectively  $\geq -1$  ( $\geq 1$ ) at initial (goal) state. The objective to minimize action (and thus state-transition) weights entails that, in any optimal solution, equality will hold in all these three cases. The condition  $\text{vars}(\mathcal{G}) = \mathcal{V}$  is required because  $\Delta_c(s) = 1$  may only hold if  $c \subseteq \mathcal{G}$ . Thus, in order for the state space graph, encoded as the variable merge over variables  $\mathcal{V}$ , to actually contain a *sink* state, it must hold  $\text{vars}(\mathcal{G}) = \mathcal{V}$ .  $\square$

From Theorem 7.3 and Theorem 7.1 together, we immediately get convergence of  $h^{\text{seq}}[\Pi^C]$ :

**Corollary 7.1.** *For every planning task  $\Pi$  in TNF, there exists a set of conjunctions  $C$  s.t.  $h^{\text{seq}}[\Pi^C] = h^*$ .*

### 7.3. Potential Heuristics

Notice that Equation (7.1) depends on the state considered. Therefore, heuristics based on that equation need to solve the LP anew for every state encountered during search. *Potential heuristics* require solving just a single LP. The LP solution is used to compute *weights* (potentials) that, when combined in a linear fashion, define an admissible heuristic. Following Pommerening et al. (2017), formally:

**Definition 7.5** (Potential Heuristic). *Let  $C$  be a set of conjunctions. Let  $w: C \mapsto \mathbb{R}$  be a weight function. The **potential heuristic** induced by  $C$  and  $w$  is the function*

$$h_{C,w}^{\text{pot}}(s) = \sum_{c \in C, c \subseteq s} w(c)$$

Given that  $h_{C,w}^{\text{pot}}(s) < \infty$  holds for all states by definition, potential heuristics at first glance do not seem suitable for unsolvability detection. We come back to this issue in our convergence analysis below.

Note that the weights can take the full range of reals. Assigning negative weights to some conjunctions can yield larger admissible heuristic values for some states, by compensating for the assignment of larger weights to other conjunctions. However, this also means the heuristic values of some states might be negative. To obtain non-negative estimates while preserving admissibility, one can simply take the maximum of the heuristic and 0.

Clearly, not every weight function  $w$  yields an admissible potential heuristic  $h_{C,w}^{\text{pot}}$ . This raises the question of how to find suitable  $w$  for a given  $C$ . This has been addressed by Pommerening et al. (2015) for singleton conjunctions, through an LP encoding of  $w$ , guaranteeing goal-awareness and consistency, and thus admissibility. Pommerening et al. (2017) extended this LP to general conjunctions. For pairs of facts, the size of their LP encoding is still polynomially bounded in the size of  $\Pi$ . For arbitrary conjunctions, however,

the LP representation may require an exponential number of variables. In fact, Pommerening et al. (2017) have shown that for conjunctions of size larger than two, the construction of desired potential heuristics is computationally hard. Here, we explore this direction further.

In Section 7.3.1, we revisit Pommerening et al.’s (2015) weight computation for singleton conjunctions. In Section 7.3.2, we show that this computation applied to  $\Pi^C$  yields  $w$  suitable for defining admissible potential heuristics over arbitrary  $C$ . In Section 7.3.3, we analyze the convergence properties of this approach. In Section 7.3.4, we briefly discuss the relation to the state-equation heuristic. That relation will serve as basis for our conjunction refinement method, presented in Section 7.4.

### 7.3.1. Admissible Fact Potential Heuristics

Plugged into the definition of consistency (Definition 2.7),  $h_{C,w}^{\text{pot}}$  is consistent if and only if it holds for all states  $s \in \mathcal{S}^\Pi$  and actions  $a \in \mathcal{A}(s)$  applicable in  $s$  that

$$\sum_{c \in C, c \subseteq s} w(c) \leq \sum_{c \in C, c \subseteq s[a]} w(c) + c(a)$$

When subtracting the  $w(c)$  terms from the right-hand side, this becomes

$$\sum_{c \in C, c \subseteq s, c \not\subseteq s[a]} w(c) - \sum_{c \in C, c \not\subseteq s, c \subseteq s[a]} w(c) \leq c(a) \quad (7.5)$$

Suppose that  $\Pi$  is in TNF, and let  $\mathcal{F}$  be its set of facts. To simplify notation, in the following, we treat facts and singleton conjunctions interchangeably. Notice that, given (TNF1), the action descriptions alone fully determine the change of every fact  $p \in \mathcal{F}$  from  $s$  to  $s[a]$ , i.e.,  $p \in s$  and  $p \notin s[a]$  hold iff  $a \in \text{Cons}(p)$ , respectively  $p \notin s$  and  $p \in s[a]$  hold iff  $a \in \text{Prod}(p)$ . Therefore, for  $C = \mathcal{F}$ , one can drop the state  $s$  from Equation (7.5), obtaining an inequation that solely depends on the action  $a$ :

$$\sum_{p \in \mathcal{F} : a \in \text{Cons}(p)} w(p) - \sum_{p \in \mathcal{F} : a \in \text{Prod}(p)} w(p) \leq c(a) \quad (7.6)$$

Moreover, by (TNF2), there is only a single goal state  $s_* = \mathcal{G}$ . So, the weights  $w$  ensure goal-awareness if

$$\sum_{p \in \mathcal{G}} w(p) \leq 0 \quad (7.7)$$

Equations (7.6) and (7.7) together define an LP, denoted  $\text{Pot}[\Pi]$ , with  $|\mathcal{A}| + 1$  constraints, and  $|\mathcal{F}|$  variables representing the weights  $w$ , i.e., an LP whose size is linear in the size of  $\Pi$ . Every feasible solution to this LP yields an admissible potential heuristic. The objective function in  $\text{Pot}[\Pi]$  can be freely chosen. Seipp et al. (2015) have explored various alternatives. We specify our choices below.

**Example 7.5.** *Reconsider Example 7.2. Consider the weight function  $w(\text{samp} \mapsto x) = 2$  for  $x \in \{A_1, A_2\}$ ,  $w(\text{samp} \mapsto R) = 1$ , and  $w(p) = 0$  for the remaining facts. Equation (7.7) is trivially satisfied. Further notice that *collect* and *drop* are the only actions with non-0 terms on the left-hand side of Equation (7.6). *collect*( $x$ ) consumes  $\text{samp} \mapsto x$  and produces  $\text{samp} \mapsto R$ , hence  $w(\text{samp} \mapsto x) - w(\text{samp} \mapsto R) \leq 1$ , i.e., Equation (7.6) is satisfied. Vice versa, *drop*( $x$ ) consumes  $\text{samp} \mapsto R$  and produces  $\text{samp} \mapsto x$ , hence again  $w(\text{samp} \mapsto R) - w(\text{samp} \mapsto x) \leq 1$ . In conclusion,  $h_{C,w}^{\text{pot}}$  is consistent and goal-aware. It achieves an initial state value  $h_{C,w}^{\text{pot}}(I) = w(\text{samp} \mapsto A_2) = 2 = h^{\text{seq}}(I)$ .*

### 7.3.2. Potential Heuristics Over Arbitrary Conjunctions

Let  $C$  now be an arbitrary set of conjunctions. We next show that the approach from Section 7.3.1, applied to  $\Pi^C$ , yields  $w$  defining an admissible potential heuristic  $h_{C,w}^{\text{pot}}$ . This is straightforward, in principle, but subtleties arise from the  $\Pi^C$  construction.

#### TNF Transformation

First, as discussed, the POT-based weight computation requires tasks in TNF. However, even under the assumption that the base task  $\Pi$  is in TNF, this still does not necessarily hold for  $\Pi^C$ . For instance, assume that  $\Pi$  contains an action  $a$  that changes the value of a single variable  $v$  from  $d$  to  $d'$ , with no other preconditions. Then, the action's occurrence  $a^0$  in  $\Pi^C$  sets  $\pi_c \mapsto \perp$  for all conjunctions  $c \in C$  with  $v \mapsto d \in c$ , but  $a^0$  has no precondition on  $\pi_c$  unless  $c = \{v \mapsto d\}$ . Therefore,  $\Pi^C$  violates (TNF1).

Before we can apply POT to  $\Pi^C$ , we hence must first bring it into TNF. To this end, we can simply leverage the standard transformation method (Pommerening and Helmert, 2015). When starting from  $\Pi$  already in TNF, this boils down to the following steps

1. add an auxiliary value  $*$  to the domain of every  $\pi_c$  variable;
2. for each  $\pi_c$ , create 0-cost actions  $\text{unset}_{c,d}$  for  $d \in \{\top, \perp\}$  with precondition  $\{\pi_c \mapsto d\}$  and effect  $\{\pi_c \mapsto *\}$ ;
3. for every action  $a^C \in \mathcal{A}^C$  and effect  $\pi_c \mapsto \perp \in \text{eff}(a^C)$  where  $\pi_c \notin \text{vars}(\text{pre}(a^C))$ , add  $\pi_c \mapsto *$  to the precondition.

Provided that the base action  $a$  satisfies (TNF1), for all effects  $\pi_c \mapsto \top \in \text{eff}(a^C)$ , it holds that  $\text{vars}(c) \subseteq \text{vars}(\text{regress}(c, a))$ , and thus  $c \not\models \text{regress}(c, a)$ . So, by definition,  $\pi_c \mapsto \perp \in \text{pre}(a^C)$ , and there is no need for adding the auxiliary precondition  $\pi_c \mapsto *$ . Moreover, given that the original goal  $\mathcal{G}$  is defined for all variables, it holds for all  $c \in C$  that one of  $c \subseteq \mathcal{G}$  or  $c \not\models \mathcal{G}$  is satisfied, i.e.,  $\mathcal{G}^C$  already satisfies (TNF2). In conclusion, the task  $\Pi_{\text{TNF}}^C$  resulting after the above steps is in TNF. Pommerening and Helmert (2015) have shown that this TNF transformation does not affect consistent and goal-aware fact potential heuristics, i.e., a fact potential heuristic is consistent and goal-aware for  $\Pi^C$  if and only if it is for  $\Pi_{\text{TNF}}^C$ .

#### From $\text{Pot}[\Pi_{\text{TNF}}^C]$ to Conjunction Potentials

Denote by  $\mathcal{F}^C$  the set of facts from  $\Pi_{\text{TNF}}^C$ . Let  $\hat{w}: \mathcal{F}^C \rightarrow \mathbb{R}$  be any weight function satisfying  $\text{Pot}[\Pi_{\text{TNF}}^C]$ , and let  $h_{\mathcal{F}^C, \hat{w}}^{\text{pot}}[\Pi_{\text{TNF}}^C]$  be the corresponding potential heuristic over the  $\Pi_{\text{TNF}}^C$  compilation. We want to find conjunction weights  $w: C \rightarrow \mathbb{R}$  such that  $h_{C,w}^{\text{pot}}(s) = h_{\mathcal{F}^C, \hat{w}}^{\text{pot}}[\Pi_{\text{TNF}}^C](s^C)$  holds for all states  $s \in \mathcal{S}^\Pi$ . The  $\pi_c \mapsto \perp$  facts however cause complications.

We assume that  $C$  contains all singleton conjunctions. Let  $\hat{v} \in \mathcal{V}$  be any variable of the original task  $\Pi$ , and let

$$W = \sum_{c \in C, |c| > 1} \hat{w}(\pi_c \mapsto \perp)$$

be the sum of all  $\pi_c \mapsto \perp$  fact weights (recall that singleton conjunctions are not represented explicitly in

$\Pi_{\text{TNF}}^C$ ). Consider the following  $w$ :

$$w(c) = \begin{cases} \hat{w}(\hat{v} \mapsto d) + W, & \text{if } c = \{ \hat{v} \mapsto d \} \\ \hat{w}(v \mapsto d), & \text{if } c = \{ v \mapsto d \} \text{ and } v \neq \hat{v} \\ \hat{w}(\pi_c \mapsto \top) - \hat{w}(\pi_c \mapsto \perp), & \text{otherwise} \end{cases}$$

In words,  $W$  is added to the weights of all  $\hat{v}$ -facts, and the weight of a conjunction is set to the difference between its  $\pi_c \mapsto \top$  and  $\pi_c \mapsto \perp$  fact weights. The former ensures that  $W$  is included in  $h_{C,w}^{\text{pot}}(s)$  for every state  $s$ . The latter removes from  $W$  the  $\pi_c \mapsto \perp$  fact weights of the conjunctions satisfied in  $s$ . In conclusion

**Lemma 7.2.** *Let  $C$  be an arbitrary set of conjunctions that contains at least all singleton conjunctions, and  $\hat{w}$  be a feasible solution to  $\text{Pot}[\Pi_{\text{TNF}}^C]$ . Then,  $w$  as defined above satisfies*

$$h_{C,w}^{\text{pot}}(s) = h_{\mathcal{F}^C, \hat{w}}^{\text{pot}}[\Pi_{\text{TNF}}^C](s^C)$$

for all states  $s \in \mathcal{S}^\Pi$ .

*Proof.*

$$\begin{aligned} h_{C,w}^{\text{pot}}(s) &= \sum_{v \in \mathcal{V}} w(v \mapsto s[v]) + \sum_{c \in C, |c| > 1, c \subseteq s} w(c) \\ &= W + \sum_{v \in \mathcal{V}} \hat{w}(v \mapsto s[v]) + \sum_{c \in C, |c| > 1, c \subseteq s} (\hat{w}(\pi_c \mapsto \top) - \hat{w}(\pi_c \mapsto \perp)) \\ &= \sum_{v \in \mathcal{V}} \hat{w}(v \mapsto s[v]) + \sum_{c \in C, |c| > 1, c \subseteq s} \hat{w}(\pi_c \mapsto \top) + \sum_{c \in C, |c| > 1, c \not\subseteq s} \hat{w}(\pi_c \mapsto \perp) \\ &= h_{\mathcal{F}^C, \hat{w}}^{\text{pot}}[\Pi_{\text{TNF}}^C](s^C) \end{aligned}$$

□

Lemma 7.1 leads to the desired result:

**Theorem 7.4.** *Let  $\Pi$  be any task, and  $C$  be any set of conjunctions containing at least all singleton conjunctions. Let  $\hat{w}$  be any solution to  $\text{Pot}[\Pi_{\text{TNF}}^C]$ , and  $w$  be the corresponding conjunction weight function, as defined above. Then,  $h_{C,w}^{\text{pot}}$  is consistent and goal-aware in  $\Pi$ .*

### POT Objective Function

It now only remains to specify POT's objective function. We employ the following two variants presented by Seipp et al. (2015). We specify them in terms of the fact weights  $\hat{w}$  in  $\Pi_{\text{TNF}}^C$ , along the lines of the previous discussion.

(O1) Maximizing the heuristic value of an individual state  $s$ :

$$\max \sum_{v \in \mathcal{V}} \hat{w}(v \mapsto s[v]) + \sum_{c \in C, |c| > 1, c \subseteq s} \hat{w}(\pi_c \mapsto \top) + \sum_{c \in C, |c| > 1, c \not\subseteq s} \hat{w}(\pi_c \mapsto \perp)$$

(O2) Maximizing the average heuristic value over all states  $\mathcal{S}^\Pi$ . This requires to normalize the weight associated with a conjunction  $c$  by the *frequency* of  $c$ , i.e., by the fraction of states in which  $c$  is satisfied. The frequency of  $c$  is syntactically defined by

$$\lambda(c) = \left( \prod_{v \mapsto d \in c} |\mathcal{D}_v| \right)^{-1}$$

The optimization objective becomes

$$\begin{aligned} \max \quad & \sum_{v \in \mathcal{V}, d \in \mathcal{D}_v} \lambda(v \mapsto d) \cdot \hat{w}(v \mapsto d) \\ & + \sum_{c \in \mathcal{C}, |c| > 1} \lambda(c) \cdot \hat{w}(\pi_c \mapsto \top) \\ & + \sum_{c \in \mathcal{C}, |c| > 1} (1 - \lambda(c)) \cdot \hat{w}(\pi_c \mapsto \perp) \end{aligned}$$

We use the first objective function, (O1), to establish the connection to state-equation heuristics. We use (O2) to encode the perfect heuristic as a potential heuristic. Note that, in general, both functions necessitate a weight upper-bound in the presence of dead ends to prevent divergence. We will discuss this as part of our convergence study in Section 7.3.3.

### Summary

In summary, one can compute admissible potential heuristics for any conjunction set  $\mathcal{C}$  via a detour to  $\Pi^{\mathcal{C}}$ . Pommerening et al.'s (2017) hardness result is reflected in the worst-case growth of  $\Pi^{\mathcal{C}}$ . But, for cases where  $\Pi^{\mathcal{C}}$  grows polynomially in  $|\mathcal{C}|$ , Theorem 7.4 shows that a desired potential heuristic can be computed in polynomial time. In this sense, Theorem 7.4 identifies a sufficient criterion for the efficient construction of potential heuristics.

It should be noted though that not every admissible potential heuristic over conjunctions  $\mathcal{C}$  can be constructed from  $\text{Pot}[\Pi_{\text{TNF}}^{\mathcal{C}}]$ . This is the case because Equation (7.6) in  $\text{Pot}[\Pi_{\text{TNF}}^{\mathcal{C}}]$  does no longer form a necessary condition for the consistency in  $\Pi$ :  $\text{Pot}[\Pi_{\text{TNF}}^{\mathcal{C}}]$  enforces consistency over occurrences  $a^{\mathcal{C}}$  where  $\mathcal{C}$  does not fully specify the action application context, while this context is always completely defined when considering  $\Pi$ 's transitions.

**Example 7.6.** Consider a task with binary variables  $v_1$  and  $v_2$  and the following components

- initial state  $\mathcal{I} = \{v_1 \mapsto 0, v_2 \mapsto 0\}$ ,
- goal  $\mathcal{G} = \{v_1 \mapsto 1, v_2 \mapsto 1\}$ ,
- two unit-cost actions:  $a_1$  changing the value of  $v_1$  from 0 to 1; and  $a_2$  changing the value of  $v_2$  from 0 to 1.

Suppose that  $\mathcal{C}$  contains all singleton conjunctions, together with

- $c_1 = \{v_1 \mapsto 0, v_2 \mapsto 0\}$ , and
- $c_2 = \{v_1 \mapsto 0, v_2 \mapsto 1\}$ , and
- $c_3 = \{v_1 \mapsto 1, v_2 \mapsto 0\}$ .

Consider the potential heuristic with weights  $w(v_1 \mapsto 0) = w(v_2 \mapsto 0) = 2$ ,  $w(c_1) = -2$ ,  $w(c_2) = w(c_3) = -1$ , and  $w(v_1 \mapsto 1) = w(v_2 \mapsto 1) = 0$ . Note that  $h_{C,w}^{\text{pot}} = h^*$ , and thus it is consistent and goal-aware. However, there is no  $\hat{w}$  that agrees with  $w$  on the singletons, while satisfying the constraints (7.6) in  $\text{Pot}[\Pi_{\text{TNF}}^{C>1}]$ . To see this, just consider the occurrence  $a_1^0$ . Equation (7.6) becomes

$$\begin{aligned} & \hat{w}(\pi_{c_1} \mapsto *) + \hat{w}(\pi_{c_2} \mapsto *) + \hat{w}(v_1 \mapsto 0) \\ & - \hat{w}(\pi_{c_1} \mapsto \perp) - \hat{w}(\pi_{c_2} \mapsto \perp) - \hat{w}(v_1 \mapsto 1) \leq 1 \end{aligned} \quad (7.8)$$

As per the 0-cost  $\text{unset}_{c,\perp}$  actions, it must be

$$\hat{w}(\pi_c \mapsto \perp) \leq \hat{w}(\pi_c \mapsto *)$$

for all conjunctions  $c$ . This together with (7.8) yields

$$\hat{w}(v_1 \mapsto 0) - \hat{w}(v_1 \mapsto 1) \leq 1$$

which is clearly violated if  $\hat{w}(v_1 \mapsto 0) = 2$  and  $\hat{w}(v_1 \mapsto 1) = 0$ .

### 7.3.3. Convergence

We next turn our attention to the convergence property, i.e., does there always exist  $C$  for which  $h_{C,w}^{\text{pot}}$  obtained from  $\text{Pot}[\Pi_{\text{TNF}}^C]$  is perfect? The answer is “yes”, under objective (O2) maximizing the average heuristic value. However, the presence of dead-end states causes complications.

Obviously, the value  $h^*(s) = \infty$  for a dead end  $s$  cannot be produced as part of the solution to an LP. Instead, the weights in the LP may diverge:  $\text{Pot}[\Pi]$  is not guaranteed to have an optimal solution. To see this, consider that no transition path starting from a dead end ever reaches the goal; so, for conjunctions  $c$  true only in dead ends, the weight can be made arbitrarily high while still satisfying consistency. Intuitively, the LP encoding imposes constraints on solution paths over conjunctions, and diverges where such a path does not exist. For that reason, Seipp et al. (2015) introduce a modified LP with the additional constraints

$$w(p) \leq U \quad (7.9)$$

for all  $p \in \mathcal{F}$ , where  $U \in \mathbb{R}$  is a user-defined parameter. We denote the modified LP by  $\text{Pot}[\Pi, U]$ .

Intuitively,  $U$  is a cut-off value on the cost of solutions considered in the LP. Convergence is achieved below  $U$ :

**Theorem 7.5.** *Let  $\Pi$  be any task in TNF, and  $U \in \mathbb{R}_0^+$ . Then there exists a set  $C$  of conjunctions s.t., with  $w$  obtained from any solution to  $\text{Pot}[\Pi_{\text{TNF}}^C, U]$  optimal for (O2),  $h_{C,w}^{\text{pot}}(s) = h^*(s)$  for all states  $s$  with  $h^*(s) \leq U$ .*

*Proof sketch.* A set of conjunctions satisfying the claim is again  $C = \mathcal{S}^\Pi$ , the set of all states in the task.  $\text{Pot}[\Pi_{\text{TNF}}^C, U]$  then boils down to an LP encoding of paths in the state space of  $\Pi$ , with Equation (7.6) bounding the value of a state by its successor states. Objective (O2) makes sure that, up to  $U$ , the exact shortest path length is returned.  $\square$

Appendix B.3.3 contains the full proof.

A simple trick now suffices to obtain  $h^*$  globally. We pessimistically interpret the cut-off  $U$  as a dead-end indicator, defining  $h_{C,w,U}^{\text{pot}}(s) = h_{C,w}^{\text{pot}}(s)$  if  $h_{C,w}^{\text{pot}}(s) < U$  and  $h_{C,w,U}^{\text{pot}}(s) = \infty$  otherwise. We then need to

choose a cut-off that will never apply on solvable states,  $U > h^*(s)$  for all  $s$  with  $h^*(s) < \infty$ . This is the case for

$$U^* = \left( \prod_{v \in \mathcal{V}} |\mathcal{D}_v| \cdot \max_{a \in \mathcal{A}} c(a) \right) + 1$$

**Corollary 7.2.** *Let  $\Pi$  be any task in TNF. Then there exists a set  $C$  of conjunctions s.t., with  $w$  obtained from a solution to  $\text{Pot}[\Pi_{\text{TNF}}^C, U^*]$  optimal for (O2),  $h_{C,w,U^*}^{\text{pot}} = h^*$ .*

For the simpler purpose of detecting all dead-end states, it is not necessary to use the exceedingly large constant  $U^*$ . We instead consider the task  $\Pi_0$  identical to  $\Pi$  except that all actions are assigned cost 0. Clearly,  $h^*[\Pi_0](s) = \infty$  iff  $h^*(s) = \infty$ , i.e.,  $h^*[\Pi_0]$  detects all dead ends in  $\Pi$ . But, all solvable states  $s$  have  $h^*[\Pi_0](s) = 0$ , so setting  $U^* = \epsilon$  to any  $\epsilon > 0$  results in  $h_{C,w,U^*}^{\text{pot}}$  that converges to  $h^*[\Pi_0]$  as per Corollary 7.2. We will denote the *unsolvability potential heuristics* constructed in this way as  $u_{C,w}^{\text{pot}}$ . As per the admissibility and consistency of  $h_{C,w,U^*}^{\text{pot}}$ ,  $u_{C,w}^{\text{pot}}$  is a valid, transitive, unsolvability detector.

**Example 7.7.** *Reconsider Example 7.1. To match the (TNF2) requirement, suppose for simplicity that  $\mathcal{G} = \{ \text{rov} \mapsto B, \text{bat} \mapsto 0 \}$ ; (TNF1) is already satisfied by all actions. Assume that all actions have cost 0. Consider the singleton conjunctions, and weights  $w(\text{bat} \mapsto 0) = 1$ ,  $w(\text{rov} \mapsto A_2) = 1$ ,  $w(\text{rov} \mapsto B) = -1$ , and  $w(p) = 0$  for the remaining facts.  $h_{C,w}^{\text{pot}}$  is goal-aware since  $h_{C,w}^{\text{pot}}(\mathcal{G}) = -1 + 1 = 0$ .  $\text{move}(A_2, A_1, 1)$  and  $\text{move}(A_1, B, 1)$  are the only actions whose left-hand side of Equation (7.6) could be positive;  $\text{move}(A_2, A_1, 1)$  because it consumes  $\text{rov} \mapsto A_2$ , and  $\text{move}(A_1, B, 1)$  because it produces  $\text{rov} \mapsto B$ . Since both actions produce  $\text{bat} \mapsto 0$ , the weights cancel out, showing that  $h_{C,w}^{\text{pot}}$  is consistent. Since  $h_{C,w}^{\text{pot}}(\mathcal{I}) = 1$ ,  $u_{C,w}^{\text{pot}}(\mathcal{I}) = \infty$ ; and indeed  $\mathcal{I}$  is a dead end.*

### 7.3.4. Relation to the State Equation

Pommerening et al. (2015) have shown that, for TNF tasks  $\Pi$ ,  $\text{Pot}[\Pi]$  under objective (O1) for a state  $s$  is the dual of the state-equation LP for  $s$ . By the strong duality theorem for linear programs, the two heuristics therefore have identical values on  $s$ .

However, beyond individual states, the heuristics differ: on states  $s'$  other than  $s$ , the potential heuristic merely gives a lower bound on  $h^{\text{seq}}(s')$ . In fact, there exist tasks and conjunction sets where *no* potential heuristic  $h_{C,w}^{\text{pot}}$  equals  $h^{\text{seq}}[\Pi^C]$  on all states.

**Example 7.8.** *Consider a task with variables  $v_1$  and  $v_2$ , domains  $\mathcal{D}_{v_1} = \mathcal{D}_{v_2} = \{0, 1, 2\}$ , and the following components*

- goal  $\mathcal{G} = \{ v_1 \mapsto 2, v_2 \mapsto 2 \}$
- two 0-cost actions:  $a_1$  changing the values of both  $v_1$  and  $v_2$  from 0 to 2; and  $a_2$  changing the values of  $v_1$  and  $v_2$  from 1 to 2.

*The initial state is not important. Every state which does not assign both  $v_1$  and  $v_2$  to the same value is a dead end. Consider the set of all singleton conjunctions  $C$ . In order to satisfy the state equation constraints for  $v_1 \mapsto 2$  and  $v_2 \mapsto 2$  in a non-goal state, one of  $a_1$  and  $a_2$  must be executed at least once. However, since  $a_1$  consumes  $v_i \mapsto 0$  for  $i \in \{1, 2\}$ ; and  $a_2$  consumes  $v_i \mapsto 1$  for  $i \in \{1, 2\}$ ; and none of them is produced by any action, there exists a feasible solution only for the solvable states, i.e.,  $h^{\text{seq}}$  recognizes all dead ends.*

*Consider the dead ends  $s_1 = \{ v_1 \mapsto 0, v_2 \mapsto 1 \}$  and  $s_2 = \{ v_1 \mapsto 1, v_2 \mapsto 0 \}$ . Observe that there is no consistent and goal-aware unsolvability potential heuristic  $u_{C,w}^{\text{pot}}$  which recognizes both  $s_1$  and  $s_2$ . Consistency*

requires that  $w(v_1 \mapsto 0) + w(v_2 \mapsto 0) \leq 0$ , as per action  $a_1$ , and  $w(v_1 \mapsto 1) + w(v_2 \mapsto 1) \leq 0$ , as per  $a_2$ . In order that  $u_{C,w}^{\text{pot}}$  recognizes both  $s_1$  and  $s_2$ , it must hold that  $w(v_1 \mapsto 0) + w(v_2 \mapsto 1) > 0$  and  $w(v_1 \mapsto 1) + w(v_2 \mapsto 0) > 0$ . There obviously does not exist  $w$  that satisfies all 4 inequations. The example can be easily extended to potential heuristics  $h_{C,w}^{\text{pot}}$  in general.

## 7.4. Conflict Analysis: Refining the State Equation

We have shown that the state-equation heuristic and potential heuristics converge to  $h^*$  for suitable conjunctions  $C$ . Thus, both are capable of representing arbitrary sets of dead ends, and therewith fulfill a necessary prerequisite of the conflict-driven learning search framework. What is missing is a concrete *refinement* procedure, i.e., given a conflict identified by Algorithm 5.1 or Algorithm 5.2, a set of dead-end states  $S$ , how to compute the conjunctions  $C$  to recognize all those states? The set of conjunctions used to proof Corollaries 7.1 and 7.2 could be used in principle, but are impractical.

In the following, we introduce a counterexample-guided refinement procedure selecting conjunctions  $X$  suitable to make  $u^{\text{seq}}[\Pi^{C \cup X}](s) = \infty$  for any dead end  $s$ . To recognize all dead ends of the conflict component  $S$ , it suffices to apply this method to the root state  $\hat{s}$  of  $S$ , as per the transitivity property of  $u^{\text{seq}}$ . To refine potential heuristics, we exploit the relation to the state-equation heuristic, and use the exact same procedure for updating  $C$ . More precisely, given that a single potential heuristic is in general insufficient to cover the same dead ends as  $u^{\text{seq}}$ , we keep an entire collection of such heuristics. To test whether a state  $s$  is a dead end, we consult every potential heuristic in the collection, returning  $\infty$  if  $u_{C,w}^{\text{pot}}(s) = \infty$  holds for at least one of them. To refine the collection on a conflicts  $S$ , we first compute  $X$  so that  $u^{\text{seq}}[\Pi^{C \cup X}](s) = \infty$  for all  $s \in S$ . Afterwards, we add a new unsolvability potential heuristic  $u_{C \cup X,w}^{\text{pot}}$  to the collection, so that  $u_{C \cup X,w}^{\text{pot}}(\hat{s}) = \infty$  for the conflict root state  $\hat{s}$ . The weights  $w$  suiting these needs can be obtained from any solution to  $\text{Pot}[\Pi_{\text{TNF}}^{C \cup X}, \epsilon]$  optimal for (O1) and that state, as per  $u^{\text{seq}}[\Pi^{C \cup X}](\hat{s}) = \infty$  and the duality relation. Due to transitivity, this is enough to recognize all conflict states  $s \in S$ .

We next spell out the details of the state-equation refinement method, computing the desired  $X$ . We assume  $\Pi$  to be in TNF. Suppose  $C$  is a set of conjunctions, and  $s$  is dead end where  $u^{\text{seq}}[\Pi^C](s) \neq \infty$ . We need to compute  $X$  such that  $u^{\text{seq}}[\Pi^{C \cup X}](s) = \infty$ . We do so by iteratively finding a conjunction  $x \notin C$  whose SEQ constraint is not satisfied by some solution to the LP  $\text{SEQ}[\Pi^C](s)$  underlying  $u^{\text{seq}}[\Pi^C](s)$ . We replace  $C$  by  $C \cup \{x\}$ , and repeat the process until  $u^{\text{seq}}[\Pi^C](s) = \infty$  is satisfied. Since  $C$  is extended in every iteration, Corollary 7.1 guarantees that the termination condition holds eventually.

It remains to show how to choose  $x$ . The refinement is based on a concrete solution  $\text{COUNT}$  to  $\text{SEQ}[\Pi^C](s)$ . Consider (1) the  $\Pi^C$  actions *selected* by  $\text{COUNT}$ , i.e., those  $a^C \in \mathcal{A}^C$  where  $\text{COUNT}_{a^C} > 0$ . Additionally consider (2) two auxiliary actions  $a_s^{C_s}$  and  $a_{\mathcal{G}}^{C_{\mathcal{G}}}$ , representing the current state and the goal, i.e.,  $\text{pre}(a_s) = \emptyset$ ,  $\text{eff}(a_s) = s$ ;  $\text{pre}(a_{\mathcal{G}}) = \mathcal{G}$ , and  $\text{eff}(a_{\mathcal{G}}) = \emptyset$ . Denote by  $\mathcal{A}_{\text{COUNT}}^C$  the actions from (1) and (2). Our key observation is that we can find an action  $a_{i_z}^{C_{i_z}} \in \mathcal{A}_{\text{COUNT}}^C$  whose precondition is not supported by  $\mathcal{A}_{\text{COUNT}}^C$ , i.e.,  $a_{i_z}^{C_{i_z}}$  such that

$$(\dagger) \text{ for all } a^C \in \mathcal{A}_{\text{COUNT}}^C, \text{regress}(\text{regress}(C_{i_z}, a_{i_z}), a) \not\subseteq \text{regress}(C, a)$$

Recall that the actions  $a^C$  in  $\Pi^C$  enumerate action occurrences, and that the precondition  $\text{pre}(a^C)$  carries the context  $\text{regress}(C, a)$  so to satisfy the conjunctions from  $C$ .

To show the existence of an action  $(\dagger)$ , consider the digraph with nodes  $\mathcal{A}_{\text{COUNT}}^C$ , and edges  $a_1^{C_1} \rightarrow a_2^{C_2}$ , for every  $a_1^{C_1}, a_2^{C_2} \in \mathcal{A}_{\text{COUNT}}^C$  such that  $\text{regress}(\text{regress}(C_2, a_2), a_1) \subseteq \text{regress}(C_1, a_1)$ , i.e., where the action occurrence  $a_1^{C_1}$  makes true the preconditions of  $a_2^{C_2}$ . Abusing notion, and using  $\text{regress}(C_{\mathcal{G}}, a_{\mathcal{G}}) = \mathcal{G}$ , observe that every path in this graph from  $a_s^{C_s}$  to  $a_{\mathcal{G}}^{C_{\mathcal{G}}}$  corresponds to a plan for  $s$ . Since  $s$  is a dead end, such a path cannot exist. Therefore, there exists at least one node which is not connected to  $a_s^{C_s}$ . If  $(\dagger)$  is not satisfied for any  $a_{\frac{1}{2}}^{C_{\frac{1}{2}}} \in \mathcal{A}_{\text{COUNT}}^C$ , then every node must have an incoming edge. But then, those actions in  $\mathcal{A}_{\text{COUNT}}^C$  that are disconnected from  $a_s^{C_s}$  must form at least one cycle. We construct a cycle  $a_1^{C_1} \rightarrow a_2^{C_2} \dots a_{n-1}^{C_{n-1}} \rightarrow a_n^{C_n} \rightarrow a_1^{C_1}$  such that

$$\text{vars}(\text{regress}(C_i, a_i)) = \mathcal{V} \quad (7.10)$$

for all  $1 \leq i \leq n$ . Recall that, by the construction of the graph,

$$\text{regress}(\text{regress}(C_j, a_j), a_i) \subseteq \text{regress}(C_i, a_i) \quad (7.11)$$

for all  $1 \leq i < n$  and  $j = i + 1$ , and for  $i = n$  and  $j = 1$ .

Such a cycle exists because of the TNF assumption. To see this, note that by (TNF1), it holds that  $\text{vars}(P) \subseteq \text{vars}(\text{regress}(P, a))$  for all variable assignments  $P$  and actions  $a \in \mathcal{A}[P]$ . For the auxiliary node  $a_{\mathcal{G}}^{C_{\mathcal{G}}}$ , we have  $\text{regress}(C_{\mathcal{G}}, a_{\mathcal{G}}) = \mathcal{G}$ , so by (TNF2),  $\text{vars}(\text{regress}(C_{\mathcal{G}}, a_{\mathcal{G}})) = \mathcal{V}$ . Let  $a^C$  be a node with an edge going into  $a_{\mathcal{G}}^{C_{\mathcal{G}}}$ . Then,  $a^C \neq a_s^{C_s}$ , by assumption, and  $\text{regress}(\mathcal{G}, a) \subseteq \text{regress}(C, a)$ , by the definition of the graph, i.e.,  $\text{vars}(\text{regress}(C, a)) = \mathcal{V}$ . Iterating this procedure will exhaust the nodes from  $\mathcal{A}_{\text{COUNT}}^C \setminus \{a_s^{C_s}\}$ , and thus we must find the desired cycle eventually.

Having found such a cycle, notice that as per Equations (7.10) and (7.11), every fact of  $\Pi^C$  (including the  $\pi_c$  facts) is produced along  $a_1^{C_1}, \dots, a_n^{C_n}$  as often as it is consumed, and vice versa. Therefore, we can obtain another feasible solution  $\text{COUNT}'$  to  $\text{SEQ}[\Pi^C]$  such that  $a_i^{C_i} \notin \mathcal{A}_{\text{COUNT}'}^C$ , for at least one  $1 \leq i \leq n$ , by reducing all  $\text{COUNT}_{a_j^{C_j}}$  values by  $\text{COUNT}_{a_i^{C_i}}$ , for the  $i$  with minimal count. Repeatedly applying this step will eventually remove all cycles, leaving us with the desired action  $a_{\frac{1}{2}}^{C_{\frac{1}{2}}}$ .

Finally, let

$$P = \text{regress}(C_{\frac{1}{2}}, a_{\frac{1}{2}})$$

denote the context-augmented precondition of  $a_{\frac{1}{2}}$ . Consider  $\Pi^{C'}$  for  $C' = C \cup \{P\}$ , and the  $\pi_P$ -variable corresponding to  $P$ . Since  $\Pi$  is in TNF,  $\pi_P \mapsto \top$  is consumed by  $a_{\frac{1}{2}}^{C_{\frac{1}{2}}}$ . However, by  $(\dagger)$ ,  $\text{COUNT}$  does not include any action that produces  $\pi_P \mapsto \top$ . In other words,  $\text{COUNT}$  violates the SEQ constraint corresponding to  $\pi_P \mapsto \top$  in  $\text{SEQ}[\Pi^{C'}]$ . Hence,  $P \notin C$ , and we can set  $x = P$ .

We employ two optimizations:

- (R1) We minimize  $P$ : starting with  $x := P$ , we iteratively remove facts  $p$  from  $x$  so long as the necessary properties are preserved.
- (R2) We consider not a single  $a_{\frac{1}{2}}^{C_{\frac{1}{2}}}$ , but all actions with that profile, and add a conjunction  $x$  for each. This results in fewer refinement iterations.

## 7.5. Learning Effectiveness: State Equation vs. Critical-Path Heuristics

We have established LP-based heuristics as an alternative to the critical-path unsolvability detector  $\mathcal{U}^C$  for the conflict-driven learning search framework. Both kinds of heuristics share the principal ability to represent any dead end via appropriate sets of conjunctions  $C$ . Yet, they may differ substantially in the concrete sets  $C$  suitable for this purpose. For the same set of conjunctions, the heuristics are generally incomparable, i.e., either one may recognize dead ends the other one does not. Examples 7.1 and 7.7 have already shown instances in favor of the LP heuristics, and it is straightforward to construct examples showing the opposite direction. However, this still leaves the question regarding the relation of the *amount* of additional information needed by either approach to recognize the same dead ends. Besides of theoretical interest, such comparison may give hints for the automatic selection of the unsolvability detector best suited for the particular planning task at hand. In the following, we provide two examples, showing that if  $\mathcal{U}^C(s) = \infty = \mathcal{U}^{\text{seq}}[\Pi^C](s)$ , for some dead end  $s$ , then

- $|C|$  must be exponential in  $|\Pi^C|$  (Proposition 7.1), and
- $|\Pi^C|$  must be exponential in  $|C|$  (Proposition 7.2).

We relate  $|C|$  and  $|\Pi^C|$  (instead of  $|C|$  and  $|C'|$ ) to account for the fact  $\mathcal{U}^C(s) = \infty$  can be tested in time polynomial in  $|C|$ , while computing  $\mathcal{U}^{\text{seq}}[\Pi^C](s)$  is only polynomial in  $|\Pi^C|$ , which in turn might be exponential in  $|C'|$ . The exploitation of these results remains a topic for future work. We next spell out the intuitions behind our constructions. The detailed proofs are available in Appendix B.3.2.

**Proposition 7.1.** *There exist planning tasks  $\Pi$ , dead-end states  $s$ , and sets of conjunctions  $C$  such that  $\mathcal{U}^{\text{seq}}[\Pi^C](s) = \infty$ , but in order that  $\mathcal{U}^C(s) = \infty$ ,  $|C'|$  must be exponential in  $|\Pi^C|$ .*

*Proof sketch.* This happens for example in a task with  $n$  binary variables  $v_i$  whose values must be flipped from 0 to 1. Each  $v_i$  is flipped individually, but each flip decrements a counter variable  $u$  by one. Assigning this counter variable  $u$  to  $n - 1$  makes the task unsolvable. The state-equation heuristic recognizes this, because every flip consumes some  $u \mapsto k$  fact, but each one of those can be produced just a single time, what limits the production of the  $v_i \mapsto 1$  facts to at most  $n - 1$  different  $v_i$  variables. However,  $\mathcal{U}^C$  needs to enumerate in  $C$ , for each  $1 \leq k < n$ , all possible combinations of  $k$   $v_i \mapsto 1$  assignments and  $u \mapsto n - k$ . Omitting just a single such combination allows to construct a regression trace from the goal that can be split into individually solvable parts.  $\square$

**Proposition 7.2.** *There exist planning tasks  $\Pi$ , sets of conjunctions  $C$ , and dead-end states  $s$ , where  $\mathcal{U}^C(s) = \infty$ , but in order that  $\mathcal{U}^{\text{seq}}[\Pi^C](s) = \infty$ ,  $|\Pi^C|$  must be exponential in  $|C|$ .*

*Proof sketch.* Such a planning task can be constructed from  $n$  ternary variables  $v_i$ . The task is designed such that none of them can be assigned to 1 when starting from the initial state. The goal is to have  $v_i \mapsto 1$  for all  $i$ . The  $v_i$  variables can freely change their values between 0 and 2, creating exponentially many reachable states. But setting any  $v_i$  to 1 is possible only via an unreachable prevail condition, requiring  $v_j \mapsto 1$  for some other  $j$ .  $\mathcal{U}^1$  recognizes the initial state as dead end, as due to the cyclic dependency between the  $v_i \mapsto 1$  and  $v_j \mapsto 1$  facts, none of them is reachable even under  $\mathcal{U}^1$ 's relaxing assumptions. On the other hand, in order that  $\mathcal{U}^{\text{seq}}[\Pi^C](\mathcal{I}) = \infty$ ,  $C$  must contain for each reachable state and  $i \neq j$  a conjunction  $c$  where either (i)  $\{v_i \mapsto 1, v_j \mapsto 0\} \subseteq c$ , or (ii)  $\{v_i \mapsto s[v_i], v_j \mapsto 0\} \subseteq c$  to cover the prevail conditions.  $\Pi^C$  then contains an action occurrence of the action setting  $v_j$  to 0, for all subset-combinations of those

conjunctions. Hence,  $|\mathcal{A}^C|$  is exponential. Note though that  $|C|$  itself does not need to be exponential, as it suffices that  $C$  contains just the aforementioned pairs.  $\square$

## 7.6. Experimental Evaluation

We first describe the general experiment setup (Section 7.6.1). Section 7.6.2 addresses a particularity in our conjunction-set refinement method that turned out to have a crucial impact on performance. Section 7.6.3 discusses our main empirical findings.

### 7.6.1. Experiment Setup

The implementation is based on our conflict-driven learning extension of FAST DOWNWARD (FD) (Helmert, 2006). The source code is publicly available.<sup>1</sup> We consider a similar setup as in Chapter 6. We focus on proving unsolvability, where dead-end detection has (naturally) the largest impact. We use the UIPC'16 benchmarks, as well as the unsolvable resource-constrained (RCP) benchmarks (Nakhost et al., 2012; Hoffmann et al., 2014). All experiments were run on machines equipped with Intel Xeon E5-2660 CPUs, using runtime (memory) limits of 30 minutes (4 GB).

In all our configurations, we initialize  $C$  to the singleton conjunctions, yet explicitly represent in  $\Pi^C$  only the non-singletons. We experiment with the following three variants:

**$C_I$ SEQ:** We apply refinement, *offline*, before search, adding new conjunctions into  $C$  until either hitting a size limit (see below) or proving the task unsolvable,  $\mathcal{U}^{\text{seq}}[\Pi^C](\mathcal{I}) = \infty$ . The subsequent search uses  $\mathcal{U}^{\text{seq}}[\Pi^C]$  for dead-end pruning, without generating new conjunctions.

**$C_S$ SEQ:** Conjunctions are learned *online*, during search, running DFS as per Algorithm 5.2 equipped with  $\mathcal{U}^{\text{seq}}[\Pi^C]$  for unsolvability detection and conflict refinement.

**$C_S$ POT:** Similar to  $C_S$ SEQ, but using potential heuristics. We maintain and refine a collection of such heuristics, as described in Section 7.4.

Similar to earlier works on the  $\Pi^C$ -compilation (Keyder et al., 2014), to cope with the worst-case explosion, we impose a size limit  $\alpha$  on the ratio  $|\mathcal{A}^C|/|\mathcal{A}|$ . Once  $\Pi^C$  reaches the limit  $\alpha$ , we disable the generation of new conjunctions. We experimented with  $\alpha \in \{2, 4, 8, \dots, 1024, \infty\}$ , where for  $\alpha = \infty$  the size of  $\Pi^C$  is not limited.

We compare these variants to the state-equation heuristic SEQ over just the singleton conjunctions; to an unsolvability potential heuristic POT over singleton conjunctions, whose weights are obtained from a solution to the POT LP optimal for (O1) and the initial state; to the dead-end PDB heuristic component of UIPC'16 winning Aidos planning system (Pommerening and Seipp, 2016; Seipp et al., 2016); and to dead-end learning DFS via  $\mathcal{U}^C$  using neighbors refinement.

Given the issue of floating point precision errors, to experimentally verify that the unsolvability potential heuristics do not wrongly flag states as dead ends, we executed separate test runs, checking whether  $\mathcal{U}^{\text{seq}}[\Pi^C](s) = \infty$  whenever  $\mathcal{U}_{C,w}^{\text{pot}}(s) = \infty$  for some state  $s$ . In all our runs, this was always the case. In the actual experiments below, this test has been disabled.

<sup>1</sup><https://doi.org/10.5281/zenodo.6992688>

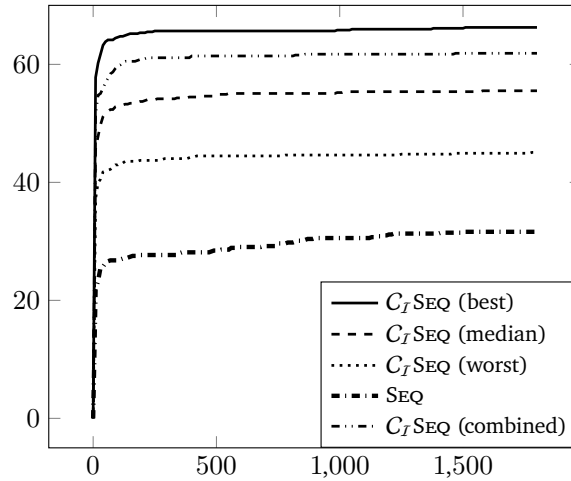


Figure 7.2.: Coverage (number of instances proved unsolvable), in %, as a function of time in seconds for different variable orders in conjunction generation, see text.

### 7.6.2. Conjunction Set Refinement: On the Order In Which To Remove Facts

Figure 7.2 sheds light on an implementation detail that turns out to be important. The optimization (R1) in our refinement method (Section 7.4) leaves open the order in which to remove facts  $p$  from  $x$ . We make this choice by fixing a variable order a priori. That order has a large impact on performance. Figure 7.2 compares the results for  $C_I$ SEQ and  $\alpha = \infty$ , for five randomly generated orders, picking the per-instance best, median, and worst variable order. The variance in coverage is large.

To counteract this brittleness, all our configurations in what follows *combine* the five variable orders, maintaining for each a separate conjunction set. Refinement works on all these sets, interleaving the individual refinement steps and stopping as soon as any of them succeeds. As can be seen in Figure 7.2 (“seeds combined”), this performs almost as well as the hypothetical per-instance best configuration. Considering fewer orders negatively affects coverage. Coverage remains stable for up to 10 orders, but starts to drop off eventually due to the additional overhead introduced with every order.

### 7.6.3. Discussion

Table 7.1 shows our main coverage results. Consider first the comparison of our algorithms to the baselines, SEQ and POT, that use the same heuristics but over the singleton conjunctions only, without any refinement. On the UIPC benchmarks, SEQ and POT dominate in the overall, but are outperformed by our techniques in DocTransfer, NoMystery, and Rovers. On the RCP benchmarks, our techniques are vastly better. These observations hold regardless of our configuration, with the single exception of  $C_I$ SEQ in UIPC Rovers. We remark that the bad performance of our methods in the BagGripper and BagTransport domains is only due to the overhead of maintaining five different conjunction sets (cf. above); when maintaining a single set  $C$ , we get the same coverage here as SEQ respectively POT.

Considering  $\mathcal{U}^{\text{seq}}$  on the initial state only, without vs. with learning (the rightmost two columns), shows that the learned larger conjunctions yield a dramatic increase in unsolvability-detection power, despite the quick-or-not-at-all performance profile observed in Figure 7.2. Indeed, the number of conjunctions needed to prove  $\mathcal{I}$  unsolvable here is typically small. The maximal conjunction-fact ratio  $|C|/|\mathcal{F}^\Pi|$  required is

								$C_S$ SEQ			$C_S$ POT			$C_I$ SEQ			
Domain	#	Blind	$u^1$	PDB	SEQ	POT	$u^C$	128	256	$\infty$	128	256	$\infty$	128	256	$\infty$	$I$ SEQ
Unsolvability IPC (UIPC) 2016 Benchmarks																	
BagBarman	20	<b>12</b>	8	<b>12</b>	4	<b>12</b>	0	0	0	0	0	0	0	0	0	0	0
BagGripper	25	4	3	3	<b>14</b>	8	2	2	2	2	2	2	2	2	2	2	<b>14</b>
BagTransport	29	7	6	7	<b>22</b>	<b>22</b>	6	19	19	19	19	19	19	19	19	19	<b>22</b>
Bottleneck	25	10	21	19	<b>25</b>	<b>25</b>	9	<b>25</b>	<b>25</b>	<b>25</b>	<b>25</b>	<b>25</b>	<b>25</b>	<b>25</b>	<b>25</b>	<b>25</b>	<b>25</b>
CaveDiving	25	7	7	7	<b>8</b>	<b>8</b>	<b>8</b>	7	7	2	4	3	2	7	7	5	1
ChessBoard	23	5	5	5	<b>23</b>	<b>23</b>	2	<b>23</b>	<b>23</b>	<b>23</b>	<b>23</b>	<b>23</b>	<b>23</b>	<b>23</b>	<b>23</b>	<b>23</b>	<b>23</b>
Diagnosis	11	4	5	5	4	4	<b>9</b>	4	4	2	3	3	2	4	4	2	0
DocTransfer	20	5	7	<b>12</b>	6	5	5	9	9	7	9	9	7	9	9	7	0
NoMystery	20	2	2	11	1	2	11	<b>12</b>	<b>12</b>	<b>12</b>	11	11	11	11	11	11	0
PegSol	24	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	14	20	20	4	16	14	4	22	22	4	0
PegSolRow5	15	5	5	5	<b>15</b>	<b>15</b>	4	<b>15</b>	<b>15</b>	<b>15</b>	<b>15</b>	<b>15</b>	<b>15</b>	<b>15</b>	<b>15</b>	<b>15</b>	<b>15</b>
Rovers	20	7	7	<b>12</b>	6	7	<b>12</b>	10	8	8	10	10	10	4	4	4	0
SlidingTiles	20	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	0	0
Tetris	20	10	5	10	<b>20</b>	<b>20</b>	5	<b>20</b>	<b>20</b>	<b>20</b>	<b>20</b>	<b>20</b>	<b>20</b>	<b>20</b>	<b>20</b>	<b>20</b>	<b>20</b>
TPP	30	17	16	<b>24</b>	11	17	19	17	17	17	17	16	16	16	16	16	2
$\Sigma$ UIPC	327	129	131	166	193	<b>202</b>	116	193	191	166	184	180	166	187	187	153	122
Unsolvable Resource-Constrained Planning (RCP) Benchmarks																	
NoMystery	150	27	53	<b>149</b>	15	27	130	137	131	131	140	132	131	135	137	137	0
Rovers	150	3	7	93	1	3	<b>142</b>	117	118	117	120	121	121	102	108	110	0
TPP	25	6	5	<b>20</b>	0	5	13	9	9	9	8	8	8	9	9	9	0
$\Sigma$ RCP	325	36	65	262	16	35	<b>285</b>	263	258	257	268	261	260	246	254	256	0
$\Sigma$ Total	652	165	196	428	209	237	401	<b>456</b>	449	423	452	441	426	433	441	409	122

Table 7.1.: Coverage results, i.e., number of instances proved unsolvable. Best results in **bold**. “blind”: exhaustive search without dead-end pruning; “ $u^1$ ”: DFS with dead-end pruning via  $u^1$ ; “ $u^C$ ”: DFS with conflict-driven learning using  $u^C$ ; “PDB”: dead-end PDBs of AIDOS; “SEQ” and “POT” the state-equation unsolvability detector  $u^{\text{seq}}$ , respectively the potential heuristic of AIDOS, both using the singleton conjunctions only. “ $C_S$ SEQ” and “ $C_S$ POT”: DFS conflict-driven learning using the state-equation and potential-heuristic unsolvability detectors, for different  $\Pi^C$  size limits  $\alpha$ . “Cert.”: state-equation *unsolvability certificates*, i.e., number of instances proved unsolvable by “SEQ”:  $u^{\text{seq}}(I) = \infty$ , for  $u^{\text{seq}}$  over the singleton conjunctions; and “ $C_I$ SEQ” where  $u^{\text{seq}}[\Pi^C](I) = \infty$  after the  $C_I$ SEQ refinement.

1.66.

Comparing to the state of the art, UIPC NoMystery is the only domain where the coverage of (the best of) our new methods is strictly higher (by the smallest margin, +1) than that of any competitor. The main advantage of our methods is that they combine both, the strength of LP heuristics on the UIPC benchmarks, and that of conjunction-learning on RCP benchmarks: they are the only configurations with near-top performance in both benchmark categories. The “ $\Sigma$  Total” row of Table 7.1 illustrates this (but should be taken with a grain of salt given the different numbers of instances per domain).

Comparing our configurations against each other, the targeted conjunction computations, based on conflicts identified during search, is more effective than offline refinement. This is in line with our observations from Section 6.4. Proving the initial state unsolvable without search ( $C_I$ SEQ with  $\alpha = \infty$ ) is competitive in RCP NoMystery, but otherwise lacks considerably behind the search variants. This is because, on the

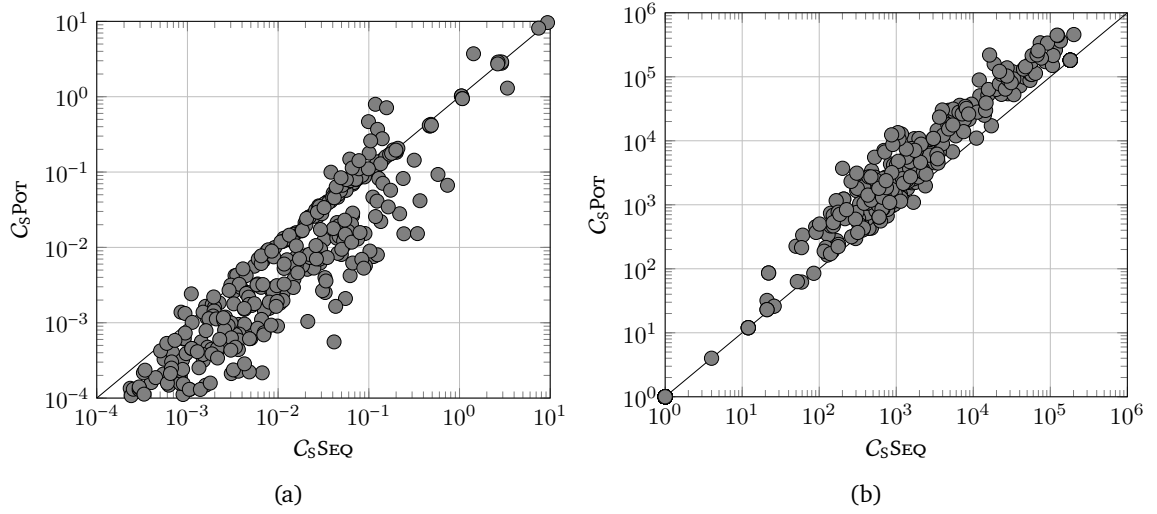


Figure 7.3.: Per-instance comparison between  $C_{SSEQ}$  (x-axes) vs.  $C_{SPOT}$  (y-axes) of (a) runtime (in seconds) and (b) search space size, i.e., the number of visited states.

one hand, the combination of state-space exploration with conjunction learning makes the search variants less affected in situations where conjunction learning is not effective (e.g., PegSol and SlidingTiles). On the other hand, when conjunction learning is effective, the online learning variants tend to be superior. The larger limits  $\alpha = 256$  and  $\alpha = \infty$  (the same applies to  $\alpha = 512$  and  $\alpha = 1024$ ) are almost consistently worse than  $\alpha = 128$ . It is almost always the case that either the refinements add only few conjunctions, or that conjunction learning does not work at all. The size limits  $\alpha = 64$  and smaller start to become too restrictive, cutting off the refinements before generalization happens at sufficient scale. This holds especially for the RCP domains, where coverage of  $C_{SSEQ}$  drops to 253 for  $\alpha = 64$  and to 242 for  $\alpha = 32$ ;  $C_{TSEQ}$  is even more affected, dropping in coverage to 232 and 207, respectively. Somewhat surprisingly, potential heuristics hardly ever improve over the state equation. Figure 7.3 elucidates the latter.  $C_{SSEQ}$  has the edge in search space size, due to its higher pruning power; while potential heuristics are faster. Yet, the former effect tends to be larger than the latter one.



## 8. On Unsolvability Detectors, the Traps They Set, and Trap Learning

A *dead-end trap* (Lipovetzky et al., 2016) is a set  $T$  of non-goal states that is invariant under transitions, i.e., where from any state  $s \in T$ , all states  $s'$  reachable from  $s$  are also contained in  $T$ . Represented compactly, dead-end traps become effective unsolvability detectors, but how to find suitable representations? Lipovetzky et al. (2016) considered state sets  $T^\Gamma$  represented as disjunctions of fact (variable-value) conjunctions  $\Gamma$ . They presented a method that applied once *offline*, before search, yields  $\Gamma$  suitable for dead-end detection and pruning during search. Here, we extend Lipovetzky et al.’s (2016) concepts in two ways:

- (i) We observe that dead-end traps can be combined for synergistic effect with arbitrary other unsolvability detectors  $\mathcal{U}$ .

We define the generalized concept of  *$\mathcal{U}$ -traps*. This captures dead-end traps relative to a given unsolvability detector  $\mathcal{U}$ , where  $T$  can be escaped, but only into dead-end states recognized by  $\mathcal{U}$ .

- (ii) We observe that  *$\mathcal{U}$ -traps* can be incrementally constructed *online*, during search, via the conflicts identified in that search.

We start with  $\Gamma$  representing the empty state set  $T^\Gamma = \emptyset$ . Every time search identifies a conflict, i.e., a set of dead ends  $S$  such that  $S \not\subseteq T^\Gamma$  and  $\mathcal{U}(s) \neq \infty$  for some  $s \in S$ , we refine  $\Gamma$  to  $\Gamma'$ , adding new conjunctions, so that the represented state set  $T^{\Gamma'}$  now includes  $S$  (and potentially other states that are dead ends for similar reasons), while preserving the  *$\mathcal{U}$ -trap* properties.

By (i), the trap  $\Gamma$  extends the reach of  $\mathcal{U}$ , avoiding “the traps set for the search by  $\mathcal{U}$ ”. By (ii), this is done dynamically from information that becomes available during search. Notably, our technique can also be run without any other unsolvability detector  $\mathcal{U}$ . In this case, (i) is mute, and (ii) turns our technique into an online-learning variant of the original dead-end traps proposal.

In the ability to exploit synergy with another unsolvability detector  $\mathcal{U}$ , our technique differs from the unsolvability detectors seen so far in that: *if  $s$  is a state all of whose successor states  $s'$  are recognized as dead ends by  $\mathcal{U}$ , then we can learn to recognize  $s$  without having to recognize also the states  $s'$* . This is in contrast to the previous unsolvability detectors, which, when learning to recognize  $s$ , necessarily – and redundantly with the given  $\mathcal{U}$  – also learn to recognize all  $s'$ . This has in particular been an issue in Chapter 6, where we found that the combination of  $\mathcal{U}^C$  dead-end learning with other unsolvability detectors  $\mathcal{U}$  suffers from having to subsume  $\mathcal{U}$  in the refinement of  $\mathcal{U}^C$ . The issue goes back to the transitivity property of the unsolvability detectors. Intuitively, our notion of  *$\mathcal{U}$ -traps* gets around this by enforcing transitivity only relative to  $\mathcal{U}$ .

This chapter is based on (Steinmetz and Hoffmann, 2017b). In Section 8.1, we revisit Lipovetzky et al.’s (2016) dead-end trap definition, and their characterization of dead-end traps via sets of fact conjunctions. In Section 8.2, we introduce our notion of  *$\mathcal{U}$ -traps*. Section 8.3 presents our variant of Lipovetzky et al.’s (2016) offline construction procedure, followed by our online conflict-based  *$\mathcal{U}$ -trap* refinement method in

Section 8.4. In Section 8.5, we compare  $\mathcal{U}$ -traps to the previously considered unsolvability detectors  $\mathcal{U}^C$  and  $\mathcal{U}^{\text{seq}}[\Pi^C]$ . Section 8.6 concludes the chapter with an experimental evaluation.

### 8.1. Preliminaries

We assume FDR planning. Let  $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  be an FDR planning task.

We denote the naive unsolvability detector as  $\mathcal{U}^0$ , where  $\mathcal{U}^0(s) = 0$  for all states  $s \in \mathcal{S}^\Pi$ , i.e.,  $\mathcal{U}^0$  does not recognize any dead end. To consider combinations of two or more unsolvability detectors  $\mathcal{U}_i$ , we use addition, which *dominates* each individual  $\mathcal{U}_i$ , returning  $\infty$  whenever  $\mathcal{U}_i$  does.

Dead-end traps (Lipovetzky et al., 2016) are formally defined as follows:

**Definition 8.1** (Dead-End Trap). *Let  $T \subseteq \mathcal{S}^\Pi$  be a set of states. Then,  $T$  is a **dead-end trap** if*

- (i) *for every state  $s \in T$ ,  $\mathcal{G} \not\subseteq s$ , and*
- (ii) *for every state  $s \in T$ , and action  $a \in \mathcal{A}(s)$  applicable in  $s$ ,  $s[a] \in T$ .*

Intuitively, a dead-end trap is a set  $T$  of non-goal states that is invariant, i.e., once we are in  $T$  we can never leave it again. In other words, a dead-end trap  $T$  forms a proof in its own that every  $s \in T$  is a dead end. The idea is to identify a compact representation  $\Gamma$  of such a  $T^\Gamma$  offline, prior to search, and to use  $\Gamma$  to detect and prune dead ends during search. Such a compact representation can be determined from fact conjunctions of size up to  $k$ , where  $k$  is a parameter. The states induced by such  $\Gamma$  are given by

$$T^\Gamma = \{ s \in \mathcal{S} \mid \exists c \in \Gamma : c \subseteq s \}$$

Verifying whether  $T^\Gamma$  is a dead-end trap can be done equivalently on  $\Gamma$  through the *progression* over variable assignments:

**Definition 8.2** (FDR Progression). *Let  $P$  be a variable assignment, and let  $a \in \mathcal{A}$  be an action. If  $P \parallel \text{pre}(a)$ , then the **progression** of  $P$  over  $a$  is*

$$\text{progress}(P, a) = (P \cup \text{pre}(a)) \circ \text{eff}(a)$$

*Otherwise, the progression is undefined, and we write  $\text{progress}(P, a) = \perp$ . We overload the applicable actions operator, and denote as  $\mathcal{A}(P) \subseteq \mathcal{A}$  the set of all actions  $a$  where  $\text{progress}(P, a) \neq \perp$ .*

In words, the progression of  $P$  over  $a$  extends  $P$  by the precondition  $\text{pre}(a)$ , and the resulting variable assignment is overwritten by  $\text{eff}(a)$ . Note that, by definition,  $\text{progress}(P, a) \subseteq s[a]$  for the application of  $a$  in any state  $s$  where  $a$  is applicable and  $P \subseteq s$ . It is hence straightforward to show that:

**Lemma 8.1.** *Let  $\Gamma$  be any set of conjunctions. Then,  $T^\Gamma$  constitutes a dead-end trap if*

- (i) *for every  $c \in \Gamma$ ,  $c \not\parallel \mathcal{G}$ , and*
- (ii) *for every  $c \in \Gamma$ , and action  $a \in \mathcal{A}(c)$ , there exists some  $c' \in \Gamma$  such that  $c' \subseteq \text{progress}(c, a)$ .*

In words,  $\Gamma$  induces a dead-end trap  $T^\Gamma$  if (i) every  $c \in \Gamma$  disagrees with the goal on some variable, and (ii)  $\Gamma$  is closed under progression. We denote the unsolvability detector associated with  $\Gamma$  as  $\mathcal{U}^\Gamma$ , where  $\mathcal{U}^\Gamma(s) = \infty$  if  $\exists c \in \Gamma$  such that  $c \subseteq s$ , and  $\mathcal{U}^\Gamma(s) = 0$  otherwise.

**Example 8.1.** Reconsider the rover running example from Example 5.1. As a reminder, there are actions  $\text{move}(x, y, k)$  for moving the rover between  $A_3$  and the other three locations ( $A_1, A_2, B$ ). Each move consumes one battery unit. Moreover, there are actions for collecting  $\text{collect}(\text{samp}_i, x)$  from, respectively dropping  $\text{drop}(\text{samp}_i, x)$  samples at the rover's current location. The initial state is  $\mathcal{I} = \{\text{rov} \mapsto A_3, \text{bat} \mapsto 2, \text{samp}_1 \mapsto A_1, \text{samp}_2 \mapsto A_2\}$ , the goal is  $\mathcal{G} = \{\text{samp}_1 \mapsto B, \text{samp}_2 \mapsto B\}$ .

Consider  $\Gamma = \{c_1, c_2, c_3, c_4\}$ , for the conjunctions  $c_1 = \{\text{bat} \mapsto 0, \text{samp}_1 \mapsto A_1\}$ ;  $c_2 = \{\text{rov} \mapsto A_1, \text{bat} \mapsto 0, \text{samp}_1 \mapsto R\}$ ; and, symmetrically,  $c_3 = \{\text{bat} \mapsto 0, \text{samp}_2 \mapsto A_2\}$ ; and  $c_4 = \{\text{rov} \mapsto A_2, \text{bat} \mapsto 0, \text{samp}_2 \mapsto R\}$ .  $\Gamma$  satisfies (i) of Lemma 8.1 because all four conjunctions assign a value to either one of the  $\text{samp}_i$  variables different from the goal.

To see that  $\Gamma$  also satisfies (ii) of Lemma 8.1, consider first  $c_1$ .  $\text{bat} \mapsto 0$  makes the precondition of every move action inconsistent with  $c_1$ ; and  $\text{samp}_1 \mapsto A_1$  makes the preconditions of all  $\text{drop}(\text{samp}_1, x)$  actions and that of all  $\text{collect}(\text{samp}_1, y)$  actions where  $y \neq A_1$  inconsistent with  $c_1$ . Of the remaining actions, those affecting the sample  $\text{samp}_2$  have no effect on  $c_1$ , i.e.,  $c_1 \subseteq \text{progress}(c_1, a)$ . Finally, the progression of  $c_1$  over  $\text{collect}(\text{samp}_1, A_1)$  makes true  $c_2$ , concluding that  $c_1$  satisfies (ii). Similarly,  $c_2$  satisfies (ii), the only action  $a \in \mathcal{A}(c_2)$  where  $c_2$  is not invariant being  $a = \text{drop}(\text{samp}_1, A_1)$ , for which it holds that  $c_1 \subseteq \text{progress}(c_2, \text{drop}(\text{samp}_1, A_1))$ . For symmetric reasons, (ii) is also satisfied for  $c_3$  and  $c_4$ . Therefore,  $T^\Gamma$  is a dead-end trap, and in particular  $s_5, s_6 \in T^\Gamma$  for the states  $s_5$  and  $s_6$  from Figure 5.1, where the rover is at  $A_3$  with no energy left, and either of the samples is still at its initial location.

## 8.2. Unsolvability Detectors and the Traps they Set

We show that dead-end traps can be combined with arbitrary unsolvability detectors.

**Definition 8.3** ( $\mathcal{U}$ -Trap). Let  $\mathcal{U}$  be an unsolvability detector, and  $T \subseteq \mathcal{S}^\Pi$  be a set of states. Then,  $T$  is a  $\mathcal{U}$ -trap if

- (i) for every state  $s \in T$ ,  $\mathcal{G} \not\subseteq s$ , and
- (ii) for every state  $s \in T$ , and action  $a \in \mathcal{A}(s)$  applicable in  $s$ , either  $s \llbracket a \rrbracket \in T$ , or  $\mathcal{U}(s \llbracket a \rrbracket) = \infty$ .

In other words, a  $\mathcal{U}$ -trap is a set of non-goal states whose only escape routes lead into dead ends recognized by  $\mathcal{U}$ . Intuitively, such  $T$  is a “trap set for the search by  $\mathcal{U}$ ”, in that, starting from  $T$ ,  $\mathcal{U}$  will eventually prune every search path; yet  $\mathcal{U}$  does not explicitly indicate this, so we will have to search through the entirety of  $T$  before finding out.

For the naive unsolvability detector  $\mathcal{U} = \mathcal{U}^0$ , the additional condition  $\mathcal{U}^0(s \llbracket a \rrbracket) = \infty$  is never satisfied. Thus,  $\mathcal{U}$ -traps generalize the original dead-end traps (the special case of  $\mathcal{U}^0$ -traps). An important difference between  $\mathcal{U}$ -traps and original dead-end traps, as we move away from  $\mathcal{U}^0$  and use more informed  $\mathcal{U}$ , is *transitivity*. While  $\mathcal{U}^0$ -traps  $T$ , by definition, have the property that for every  $s \in T$  and every transition  $\langle s, a, s' \rangle$ , it must be  $s' \in T$ , this is no longer so for  $\mathcal{U}$ -traps in general: those  $s'$  where  $\mathcal{U}(s') = \infty$  no longer need to be contained in  $T$ . As previously discussed, this is key to synergy, as it allows  $T$  to be complementary to  $\mathcal{U}$ , instead of forcing  $T$  to subsume  $\mathcal{U}$ .

To make use of  $\mathcal{U}$ -traps in search, the idea is to identify compact representations  $\Gamma$ , whose computation does not require the enumeration of  $T^\Gamma$ . For the characterization of such  $\Gamma$ , we require the unsolvability detector  $\mathcal{U}$  to be *partial-state compatible*. This is, it must be possible to evaluate  $\mathcal{U}$  on partial variable assignments  $P$  efficiently, where  $\mathcal{U}(P) = \infty$  if and only if  $\mathcal{U}(s) = \infty$  for all states  $s \in \mathcal{S}$  so that  $P \subseteq s$ . Note that, in principle, every dead-end detector can be evaluated on partial variable assignments, since  $\mathcal{U}(P)$

can be computed trivially by enumerating the states  $s$  that satisfy  $P$ , and evaluating  $\mathcal{U}$  for every one of them. But this is not in general efficient as the number of states  $s$  is exponential in the number of variables where  $P$  is undefined. Fortunately, many unsolvability detectors natively support the evaluation on partial variable assignments, and for those that do not, there usually exist sufficient conditions for  $\mathcal{U}(P) = \infty$  that can be tested efficiently. We give more details on that in the experiments section.

The original dead-end trap conditions are easily generalized:

**Theorem 8.1.** *Let  $\mathcal{U}$  be an unsolvability detector, and  $T^\Gamma$  be any set of conjunctions. Then,  $T^\Gamma$  is a  $\mathcal{U}$ -trap if*

(T1) *for every  $c \in \Gamma$ ,  $c \not\models \mathcal{G}$ , and*

(T2) *for every  $c \in \Gamma$ , and every action  $a \in \mathcal{A}(c)$ , there either exists some  $c' \in \Gamma$  such that  $c' \subseteq \text{progress}(c, a)$ , or  $\mathcal{U}(\text{progress}(c, a)) = \infty$ .*

*Proof.* Assume the contrary. It follows immediately from (T1) that  $T^\Gamma$  cannot contain a goal state. Hence, there must be a state  $s \in T^\Gamma$  and an action  $a \in \mathcal{A}(s)$  that is applicable in  $s$  so that  $s \not\models a$  and  $\mathcal{U}(s \not\models a) < \infty$ . Let  $c \in \Gamma$  be so that  $c \subseteq s$ . Note that because  $\text{pre}(a) \subseteq s$ , it immediately follows that  $c \parallel \text{pre}(a)$ , and thus the progression  $\text{progress}(c, a)$  is defined. Now, due to (T2), either there exists  $c' \in \Gamma$  with  $c' \subseteq \text{progress}(c, a)$ , or  $\mathcal{U}(\text{progress}(c, a)) = \infty$ . Since  $\text{progress}(c, a) \subseteq s \not\models a$ , both cases yield a contradiction.  $\square$

**Example 8.2.** *Consider the example from before. Notice that the set of conjunctions  $\Gamma = \{c_1, c_2\}$ , where  $c_1 = \{bat \mapsto 1, samp_1 \mapsto A_1\}$  and  $c_2 = \{rov \mapsto A_1, bat \mapsto 1, samp_1 \mapsto R\}$ , yields a  $\mathcal{U}^1$ -trap. Clearly, (T1) is satisfied, as both conjunctions assign  $samp_1$  to a value different from the goal (which is  $B$ ). It remains to show that both conjunctions satisfy (T2).*

*Consider  $c_1$ . Notice that  $c_1$  is invariant under  $\text{collect}(samp_2, x)$  and  $\text{drop}(samp_2, x)$  (no matter of  $x$ ), because neither action affects any of  $c_1$ 's variables. Moreover, notice that  $c_2 \subseteq \text{progress}(c_1, \text{collect}(samp_1, A_1))$ . The remaining actions in  $\mathcal{A}(c_1)$  are  $\text{move}(x, y, 1)$ , for all possible values  $x$  and  $y$ . The progression of  $c_1$  over any of these yields  $\text{progress}(c_1, \text{move}(x, y, 1)) = \{samp_1 \mapsto A_1, bat \mapsto 0, rov \mapsto y\} =: P$ , which due to the  $bat \mapsto 0$  assignment, is not covered by  $\Gamma$ . However, note that  $\mathcal{U}^1(P) = \infty$ , because no matter of the value of  $samp_2$ , and regardless of the rover location  $y$ , there is at least one move necessary in order to get the rover to both  $A_1$  (needed to collect  $samp_1$ ) and to  $B$  (needed to achieve the goal of  $samp_1$ ). With no energy left, this is not possible even under  $\mathcal{U}^1$ 's relaxing assumptions. Therefore,  $c_1$  satisfies (T2) for  $\mathcal{U}^1$ .*

*Consider  $c_2$ . Again  $c_2$  is invariant under actions affecting the sample  $samp_2$ , and symmetrically to above, the progression of  $c_2$  over  $\text{drop}(samp_1, A_1)$  is covered by  $c_1$ . There is only one other action whose precondition is consistent with  $c_2$ :  $\text{progress}(c_2, \text{move}(A_1, A_3, 1)) = \{samp_1 \mapsto R, bat \mapsto 0, rov \mapsto A_3\} =: P'$ . As before, it holds that  $\mathcal{U}^1(P') = \infty$ , as there is no energy left for the rover to move to  $B$ , which is needed to achieve the goal for  $samp_1$ .*

Hence,  $T^\Gamma$  constitutes a  $\mathcal{U}^1$ -trap. Finally, reconsider the states  $s_1, s_2, s_3$  from Example 5.1, where the rover is at  $A_1, A_2$ , respectively  $B$ , with 1 battery unit left, and the samples being at their initial locations. Notice that all three satisfy  $c_1$ , i.e., are recognized by  $\mathcal{U}^\Gamma$  as dead ends while neither of them is recognized by  $\mathcal{U}^1$ . Further notice that  $T^\Gamma$  does not constitute a dead-end trap as per the original definition, since, e.g., the successor state  $s_5$  of  $s_1$ , where the rover is at  $A_3$  with no energy left, does not satisfy any of the conjunctions.

**Algorithm 8.1:** Offline  $\mathcal{U}$ -trap construction.

---

**Input:** Set of conjunctions  $C$ ,  
Partial-state compatible unsolvability detector  $\mathcal{U}$

**Output:** Maximal subset  $\Gamma \subseteq C$  such that  $\Gamma$  satisfies (T1) and (T2) of Theorem 8.1.

```

/* Construct AND/OR graph  $\langle N_{\text{and}}, N_{\text{or}}, E \rangle$ : */
1  $N_{\text{and}} \leftarrow \{ a^c \mid c \in C, a \in \mathcal{A}(c), \mathcal{U}(\text{progress}(c, a)) \neq \infty \};$ 
2  $N_{\text{or}} \leftarrow C;$ 
3  $E \leftarrow \{ \langle c, a^c \rangle \mid c \in N_{\text{or}}, a \in \mathcal{A}(c) \}$ 
    $\{ \langle a^c, c' \rangle \mid a^c \in N_{\text{and}}, c' \subseteq \text{progress}(c, a) \};$ 
/* Propagate markings */
4  $\text{marked} \leftarrow \{ c \in N_{\text{or}} \mid c \parallel \mathcal{G} \};$ 
5 while  $\text{marked}$  changes do
6   foreach  $a^c \in N_{\text{and}} \setminus \text{marked}$  do
7     if  $c' \in \text{marked}$  for all  $\langle a^c, c' \rangle \in E$  then
8        $\text{marked} \leftarrow \text{marked} \cup \{ a^c \};$ 
9   foreach  $c \in N_{\text{or}} \setminus \text{marked}$  do
10    if  $a^{c'} \in \text{marked}$  for some  $\langle c, a^{c'} \rangle \in E$  then
11       $\text{marked} \leftarrow \text{marked} \cup \{ c \};$ 
/* Return conjunctions that have not been marked */
12 return  $N_{\text{or}} \setminus \text{marked};$ 

```

---

**8.3. Offline Construction**

The algorithm for computing  $\mathcal{U}^0$ -traps proposed by Lipovetzky et al. (2016) can be easily adapted to support the generation of  $\mathcal{U}$ -traps for arbitrary partial-state compatible unsolvability detectors  $\mathcal{U}$ . Algorithm 8.1 shows the general procedure. It identifies a subset of the given conjunction *candidates*  $C$ , guaranteeing that the result satisfies the conditions of Theorem 8.1. In Lipovetzky et al.'s (2016) original proposal, all conjunctions of size up to  $k$  were considered in  $C$ , where  $k$  was a parameter. However, this is not required for the correctness of the algorithm;  $C$  may be chosen arbitrarily.

Finding the desired subset of conjunctions corresponds to propagating markings through an AND/OR graph whose AND-nodes correspond to actions, whose OR-nodes correspond to the selection of conjunctions in consideration, and whose edges correspond to progression over those conjunctions. The conjunctions not marked during this procedure give the resulting  $\Gamma$ . The propagation starts with the OR-nodes, so conjunctions, that do not disagree with the goal on any variable, i.e., those violating (T1). An AND-node is marked when all its successors are marked, and thus the corresponding progression would not be covered by the resulting trap. An OR-node is marked when at least one of its successors is marked, i.e., when there is an action whose progression would lead out of the trap, violating (T2).

The main difference to Lipovetzky et al.'s (2016) algorithm is that an action  $a \in \mathcal{A}(c)$  is ignored, if the progression of  $c$  over  $a$  is already recognized by  $\mathcal{U}$  as dead end. The resulting AND/OR-graph may hence contain fewer edges, so fewer conjunctions may be touched during marking propagation, and hence removed from  $C$ , resulting in larger traps  $T^\Gamma$ .

The procedure guarantees to find a maximal trap:

**Theorem 8.2.** *Let  $\Gamma$  be the result of Algorithm 8.1. It holds*

- (i)  $\Gamma$  satisfies (T1) and (T2) of Theorem 8.1, and
- (ii) for every  $\Gamma' \subseteq C$  satisfying (T1) and (T2), it holds that  $\Gamma' \subseteq \Gamma$ .

*Proof.* Claim (i) is a simple extension of the argument by Lipovetzky et al. (2016). Claim (ii) holds because the procedure does not unnecessarily remove any conjunction.  $\square$

#### 8.4. Trap Learning: Conflict Analysis & Refinement

One major drawback of the algorithm from the previous section is that the computation of  $\Gamma$  requires up-front a choice of conjunctions candidates  $C$ . The only known method is the enumeration of all conjunctions of size up to  $k$ . This is feasible only for small  $k$ . On the other hand, many of those conjunctions might actually not be relevant for the resulting  $\mathcal{U}$ -trap representation, and by imposing a size bound on the conjunctions, we might be missing the ones that actually matter.

We next show how to integrate  $\mathcal{U}$ -traps into the conflict-driven learning search framework, Algorithms 5.1 and 5.2, by providing the necessary trap-refinement procedure. In effect, we obtain a  $\mathcal{U}$ -trap construction method that chooses the conjunctions dynamically, during search, making use of the knowledge that becomes available because of the search. However, instead of choosing for this refinement the candidate conjunctions  $C$  to feed into Algorithm 8.1, we update the  $\mathcal{U}$ -trap representation  $\Gamma$  directly.

##### 8.4.1. Overall Refinement Procedure

The general idea is to run search with the combined unsolvability detector  $\mathcal{U} + \mathcal{U}^\Gamma$ , pruning all dead-end states  $s$  recognized by any of  $\mathcal{U}$  or  $\mathcal{U}^\Gamma$ , i.e.,  $\mathcal{U}(s) = \infty$  or  $\mathcal{U}^\Gamma(s) = \infty$ . We start with the trivial  $\mathcal{U}$ -trap initialization  $\Gamma := \emptyset$ . When search identifies a conflict  $S \subseteq \mathcal{S}$ , i.e., a set of dead-end states  $S$  such that  $\mathcal{U}(s) \neq \infty$  and  $\mathcal{U}^\Gamma(s) \neq \infty$  for some  $s \in S$ , we compute a *generalization*  $\Gamma'$  of  $\Gamma$ , guaranteeing that  $\Gamma'$  still satisfies (T1) and (T2) of Theorem 8.1, and, additionally,

(T3)  $\Gamma'$  is *weaker* than  $\Gamma$ , i.e., every state represented by  $\Gamma$  is also represented by  $\Gamma'$ ,  $T^\Gamma \subseteq T^{\Gamma'}$ , and

(T4) the states in  $S$  are all covered by  $\Gamma'$ , i.e.,  $S \subseteq T^{\Gamma'}$ .

Conditions (T1) and (T2) ensure that  $T^{\Gamma'}$  still constitutes a  $\mathcal{U}$ -trap; (T3) and (T4) ensure progress in that  $\Gamma'$  recognizes strictly more dead ends than  $\Gamma$ .

Note that such a refinement is always possible, as setting  $\Gamma' = \{s \in \mathcal{S}^\Pi \mid s \text{ is a dead end, } \mathcal{U}(s) \neq \infty\}$  trivially satisfies (T1) to (T4). Yet, this refinement would obviously not be very useful. We instead attempt to extend  $\Gamma$  by *small* conjunctions  $\mathcal{X}$ , setting  $\Gamma' = \Gamma \cup \mathcal{X}$ . The conjunctions  $\mathcal{X}$  have the potential to *generalize* to states outside of  $S$ , and in particular to states not visited by the search so far.

For the computation of  $\mathcal{X}$ , we leverage once again the *recognized-neighbors property* (Definition 6.3), i.e., that upon the refinement call on  $S$ , all neighbor states  $t \in \text{Succ}(S) \setminus S$  are already recognized by  $\mathcal{U} + \mathcal{U}^\Gamma$ ,  $(\mathcal{U} + \mathcal{U}^\Gamma)(t) = \infty$ . Recall from Proposition 6.1 that this property is necessarily guaranteed if search prunes only dead ends recognized by  $\mathcal{U} + \mathcal{U}^\Gamma$ .

With the recognized-neighbors property at hand, a trivial refinement satisfying (T1) – (T4) is given by  $\mathcal{X} = S$ . Let  $\Gamma' := \Gamma \cup S$ . That every conjunction in  $\Gamma'$  disagrees with the goal on some variable follows from

the invariant that  $\Gamma$  satisfies (T1), and the assumption that  $S$  does not contain a goal state.  $\Gamma'$  satisfies (T2), due to the invariant that  $\Gamma$  satisfies (T2), and, given the recognized-neighbors property, since every transition going out of the states in  $S$  ends in a dead end recognized by  $\mathcal{U}$ , or a state that is represented by  $\Gamma$ . (T3) and (T4) are satisfied by construction. However, in this computation of  $\Gamma'$ ,  $T^{\Gamma'}$  would merely be an extension of the previous trap  $T^{\Gamma}$  to the states in  $S$ . In particular,  $\Gamma'$  does not generalize to states that have not yet been visited by search so far.

#### 8.4.2. Generalizing $\mathcal{U}$ -Traps

---

**Algorithm 8.2:** Online  $\mathcal{U}$ -trap refinement.

---

**Input:** Partial-state compatible unsolvability detector  $\mathcal{U}$ ,  
 Current  $\mathcal{U}$ -trap representation  $\Gamma$ ,  
 Set of dead-end states  $S$ , satisfying the  $\mathcal{U} + \mathcal{U}^{\Gamma}$  recognized-neighbors property  
**Output:** Set of conjunctions  $\mathcal{X}$  such that  $\Gamma' = \Gamma \cup \mathcal{X}$  satisfies (T1) – (T4).

```

1  $c_s \leftarrow \emptyset$ , for each  $s \in S$ ;
  /* (T1) */
2 foreach  $s \in S$  do
3   | let  $v \in \text{vars}(\mathcal{G})$  be such that  $s[v] \neq \mathcal{G}[v]$ ;
4   |  $c_s \leftarrow \{v \mapsto s[v]\}$ ;
  /* (T2) */
5 while there is some  $s \in S$  such that  $c_s$  violates (T2) do
6   | let  $v \in \mathcal{V} \setminus \text{vars}(c_s)$ ;
7   |  $c_s \leftarrow c_s \cup \{v \mapsto s[v]\}$ ;
8 return  $\{c_s \mid s \in S\}$ ;
```

---

Let  $S \subseteq \mathcal{S}^{\Pi}$  be a set of states satisfying the  $\mathcal{U} + \mathcal{U}^{\Gamma}$  recognized-neighbors property. Towards obtaining a refined  $\mathcal{U}$ -trap representation  $\Gamma'$  that potentially generalizes to other states than  $S$ , the idea is compute conjunctions from  $S$  by removing variable assignments from the states not relevant w.r.t. (T1) and (T2). The pseudocode of this *generalization* procedure is shown in Algorithm 8.2. For each state  $s \in S$ , a conjunction  $c_s \subseteq s$  is computed so that the extension of  $\Gamma$  by all those  $c_s$  still satisfies (T1) and (T2). The smaller the subsets are, the more states are represented by the resulting set of conjunctions.

To ensure that  $\Gamma'$  satisfies condition (T1), for every state  $s \in S$ , the corresponding  $c_s$  is initialized to one of state's facts disagreeing with the goal. Such a fact must exist because  $S$  does not contain a goal state by assumption. Condition (T2) is ensured by iteratively re-adding facts from the states to their conjunctions. As argued above,  $\Gamma \cup S$  satisfies (T2), and hence this happens when  $c_s = s$  for all  $s \in S$  at the latest. Note that regardless which state  $s$  is selected inside the while loop, there must be always a variable  $v \in \mathcal{V}$  that is not assigned by the corresponding conjunction,  $v \notin \text{vars}(c_s)$ . Namely, for every state  $s \in S$  where  $c_s = s$ , and for every action  $a \in \mathcal{A}(s) = \mathcal{A}(c_s)$ , it holds that  $s[a] = \text{progress}(c_s, a)$ , and one of  $s[a] \in S$  or  $(\mathcal{U} + \mathcal{U}^{\Gamma})(s[a]) = \infty$  is true as per the recognized-neighbors requirement. Given that  $c_{s'} \subseteq s'$ , for every state  $s' \in S$ , by construction, this however means that  $s$  cannot be the state violating (T2), i.e., such states  $s$  cannot be chosen. Hence, the execution of Algorithm 8.2 is well-defined, and terminates with  $\mathcal{X}$  such that  $\Gamma' = \Gamma \cup \mathcal{X}$  satisfies (T1) and (T2). Moreover,  $\Gamma'$  satisfies (T3), because it is a superset of  $\Gamma$ .  $\Gamma'$  satisfies (T4), because it contains a conjunction  $c_s \subseteq s$ , for every state  $s \in S$ . Finally,  $\Gamma'$  may generalize to states outside of  $S$  as soon as at least some  $c_s$  is a strict subset of  $s$ .

**Theorem 8.3.** *Let  $S$  be a set of dead ends satisfying the  $\mathcal{U} + \mathcal{U}^\Gamma$  recognized-neighbors property. The execution of Algorithm 8.2 is well-defined, and terminates with  $\mathcal{X}$  such that  $\Gamma \cup \mathcal{X}$  satisfies (T1) – (T4).*

Note that when extending  $\Gamma$  by new conjunctions, it may become possible to also minimize the existing conjunctions in  $\Gamma$ . Thus, it could make sense to apply Algorithm 8.2 to all elements in  $\Gamma \cup S$ , instead of just  $S$ . Yet in our experiments this turned out to be detrimental, causing always a prohibitive overhead while improving generalization only in rare cases.

**Example 8.3.** *Consider once more the conflict-driven learning example, Example 5.1. Suppose we combine  $\mathcal{U}^1$  dead-end pruning with  $\mathcal{U}^1$ -trap learning. As argued in Chapter 6,  $\mathcal{U}^1$  recognizes exactly the dead-end states pruned in the search from Example 5.1. Let  $\Gamma := \emptyset$ . Consider the first conflict refinement requested by that search, i.e.,  $S = \{s_4, s_1\}$ , where  $s_4 = \{rov \mapsto A_1, bat \mapsto 1, samp_1 \mapsto R, samp_2 \mapsto A_2\}$  and  $s_1$  is the same but with  $samp_1 \mapsto A_1$ . Their successor states  $s_5$  and  $s_6$  are recognized as dead ends by  $\mathcal{U}^1$ , i.e.,  $S$  satisfies the  $\mathcal{U}^1 + \mathcal{U}^\Gamma$  recognized-neighbors property. Suppose we use Algorithm 8.2 to compute a refinement of  $\Gamma$ .*

*We initialize  $c_{s_4} := \emptyset$  and  $c_{s_1} := \emptyset$ . To satisfy (T1), suppose we choose for both states  $samp_1$ , updating the conjunctions to  $c_{s_1} := \{samp_1 \mapsto A_1\}$  and  $c_{s_4} := \{samp_1 \mapsto R\}$ .*

*Afterwards, we find that  $c_{s_4}$  violates (T2) due to  $progress(c_{s_4}, drop(samp_1, B)) = \{rov \mapsto B, samp_1 \mapsto B\}$  which is neither covered by the chosen conjunctions, nor by  $\mathcal{U}^1$ . Suppose we extend  $c_{s_4}$  by the position of the rover, i.e.,  $c_{s_4} = \{rov \mapsto A_1, samp_1 \mapsto R\}$ .*

*$c_{s_4}$  still violates (T2), because of  $progress(c_{s_4}, move(A_1, A_3, 2)) = \{rov \mapsto A_3, bat \mapsto 1, samp_1 \mapsto R\} =: P$ .  $P$  is not covered by the chosen conjunctions, and with the state  $t = P \cup \{samp_2 \mapsto B\}$  being solvable,  $P$  cannot possibly be recognized as dead end by  $\mathcal{U}^1$ . Suppose we include  $bat$  into  $c_{s_4}$ , resulting in  $c_{s_4} := \{rov \mapsto A_1, bat \mapsto 1, samp_1 \mapsto R\}$ .*

*There remains only a single move action whose precondition is consistent with  $c_{s_4}$ ,  $move(A_1, A_3, 1)$ , for which  $\mathcal{U}^1(progress(c_{s_4}, move(A_1, A_3, 1))) = \infty$  (the progression sets  $bat \mapsto 0$ , making the goal  $samp_1 \mapsto B$  unreachable even under  $\mathcal{U}^1$ ). Moreover, with  $c_{s_1} \subseteq progress(c_{s_4}, drop(samp_1, A_1))$ ,  $c_{s_4}$  satisfies (T2).*

*$\Gamma \cup \mathcal{X}$  does still not satisfy (T2),  $c_{s_1}$  violating that condition via, e.g.,  $move(A_1, A_3, 2)$ . Extending  $c_{s_1}$  by the battery value,  $\{bat \mapsto 1, samp_1 \mapsto A_1\}$ , resolves this last issue, and the refinement terminates.*

*Consider  $\Gamma' = \{c_{s_1}, c_{s_4}\}$ . With  $\Gamma = \emptyset \subseteq \Gamma'$ , the requirement (T3) is trivially satisfied. It holds that  $s_1, s_4 \in T^\Gamma$ , by construction, fulfilling (T4). Finally, notice that  $\Gamma'$  generalizes beyond the input conflict component, recognizing, e.g., the dead ends  $s_2, s_3$ , where the rover is at  $A_2$  respectively  $B$  with one battery unit left and  $samp_1$  being at its initial location.*

## 8.5. Learning Effectiveness: Comparison to the State-Equation and Critical-Path Heuristics

As already mentioned, a distinguishing feature of  $\mathcal{U}$ -traps compared to the unsolvability detectors from Chapters 6 and 7 is the ability to incorporate information of arbitrary other (partial-state compatible) unsolvability detectors, without the need of duplicating that information. Yet, even leaving this feature aside, the structure of dead-end traps can be advantageous in representing dead-end states more compactly than possible by the other approaches. Continuing our previous discussions, here we attempt to shed light on the individual strengths of the different techniques.

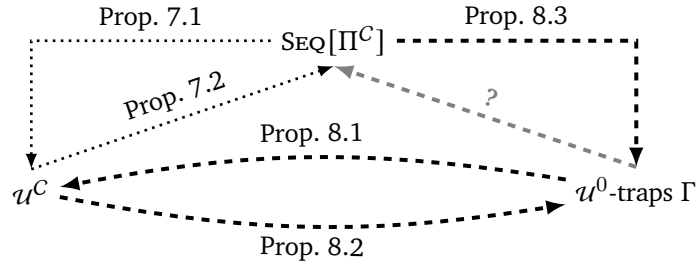


Figure 8.1.: Comparison between the families of unsolvability detectors considered so far in terms of their *learning effectiveness*. An edge  $A \rightarrow B$  refers to the example showing that  $A$  can be more effective than  $B$ . The relations represented by the dotted edges were already given in Section 7.5.

In Section 7.5, we have compared the critical-path and the conjunctions-augmented state-equation unsolvability detectors, showing for both directions cases where  $u^C(s) = \infty = u^{\text{seq}}[\Pi^C](s)$  entails that  $|C|$  and  $|\Pi^C|$  relate to each other by an exponential factor. In the following, we extend the analysis to  $u^0$ -traps  $\Gamma$ , i.e., dead-end traps not taking into account any other source of unsolvability information.

Next, we provide examples, similar to the previous ones, comparing  $u^C$  and  $\Gamma$  in either direction. Moreover, we show that there are cases where  $u^{\text{seq}}[\Pi^C](s) = \infty = u^\Gamma(s)$  implies  $|\Gamma|$  being exponential in  $|\Pi^C|$ . The opposite direction we have to leave open. We could neither find an affirmative example, nor prove that  $u^{\text{seq}}[\Pi^C]$  can simulate any  $u^0$ -trap  $\Gamma$  with only polynomial overhead. We conjecture that instantiating  $C = \Gamma$  is generally sufficient for  $u^{\text{seq}}[\Pi^C]$  recognizing the dead ends as represented by  $\Gamma$ . Yet, considering the  $\Pi^C$  construction, there are likely cases where  $|\Pi^C|$  grows exponentially, but where the cross-context information encoded in  $\Pi^C$ 's actions is not necessary to represent a dead end, yielding the advantage for the dead-end traps  $\Gamma$ . Figure 8.1 provides an overview of our known results.

**Proposition 8.1.** *There exist planning tasks  $\Pi$ , dead-end states  $s$ , and sets of conjunctions  $\Gamma$  satisfying (i) and (ii) of Lemma 8.1, where  $s \in T^\Gamma$  but in order that  $u^C(s) = \infty$ ,  $|C|$  must be exponential in  $|\Gamma|$ .*

*Proof sketch.* Such an example is given by task from our worst-case  $u^C$  refinement result, Proposition 6.2. The task consisted in flipping the values of  $n$  variables  $p_i$  from 0 (initial value) to 1 (goal). Each flip action requires an additional precondition  $r \mapsto 1$ , which to achieve, necessitates changing the value of any one of the  $p_i$  variables to 2 (i.e., deleting  $p_i \mapsto 0$ ), preventing that variable from ever reaching its goal value afterwards.  $u^C$  requires exponentially (in  $n$ ) many conjunctions to prove this task unsolvable, whereas the set  $\Gamma = \{ \{ p_i \mapsto 2 \} \mid 1 \leq i \leq n \} \cup \{ r \mapsto 0 \}$  satisfies the dead-end trap conditions, and with  $\mathcal{I}[r] = 0$ , represents the initial state.  $\square$

**Proposition 8.2.** *There exist planning tasks  $\Pi$ , dead-end states  $s$ , and sets of conjunctions  $C$ , where  $u^C(s) = \infty$ , but for every  $\Gamma$  satisfying (i) and (ii) of Lemma 8.1 with  $s \in T^\Gamma$ ,  $|\Gamma|$  is exponential in  $|C|$ .*

*Proof sketch.* Such an example is given by the task from Proposition 7.2. The task consisted in changing the values of  $n$  variables  $v_i$  from 0 (initial state) to 1 (goal). The variables can change their values freely between 0 and 2, but changing from 0 to 1 necessitates some other variable  $v_j$  to already have the value of 1.  $u^1$  recognizes the initial state as dead end, as due the cyclic dependencies between the facts  $v_i \mapsto 1$  and  $v_j \mapsto 1$ , none of them are reachable even under  $u^1$ 's relaxing assumptions. In contrast, in order that  $\mathcal{I} \in T^\Gamma$ ,  $\Gamma$  must enumerate all reachable states. Leaving only a single variable unassigned in any  $c \in \Gamma$  yields a  $T^\Gamma$  containing solvable states.  $\square$

**Proposition 8.3.** *There exist planning tasks  $\Pi$ , dead-end states  $s$ , and sets of conjunctions  $C$  such that  $\mathcal{U}^{\text{seq}}[\Pi^C](s) = \infty$ , but for every  $\Gamma$  satisfying (i) and (ii) of Lemma 8.1 with  $s \in T^\Gamma$ ,  $|\Gamma|$  is exponential in  $|\Pi^C|$ .*

*Proof sketch.* Such an example is given by the task from Proposition 7.1. The task again consisted in changing the value of  $n$  variables  $v_i$  from 0 to 1, while now each such change decrements a counter variable  $u$ . With  $\mathcal{I}[u] = n - 1$ , the task is unsolvable. The linear program underlying  $\mathcal{U}^{\text{seq}}$  (i.e., using the singleton conjunctions only) sees that the counter’s facts need to be “consumed”  $n$  times to achieve all goal facts, while it is possible to “produce” those facts only  $n - 1$  times. Hence,  $\mathcal{U}^{\text{seq}}$  recognizes  $\mathcal{I}$  as dead end. In order that  $\mathcal{I} \in T^\Gamma$ ,  $\Gamma$  must contain a conjunction  $c_s$  for every reachable state of the form  $s = \{v_1 \mapsto 0, \dots, v_k \mapsto 0, v_{k+1} \mapsto 1, \dots, v_n \mapsto 1, u \mapsto k - 1\}$  such that  $\{v_1, \dots, v_k, u\} \subseteq \text{vars}(c_s)$ . Leaving only a single of these variables unassigned results in  $T^\Gamma$  containing solvable states.  $\square$

Detailed proofs are available in Appendix B.4.1.

## 8.6. Experimental Evaluation

Section 8.6.1 lays down the general experiment setup. In Section 8.6.2, we list the unsolvability detectors that we consider for generating  $\mathcal{U}$ -traps. For each of them, we provide a brief description on how to compute or approximate  $\mathcal{U}(P)$  for a (partial) variable assignment  $P$ . Section 8.6.3 presents and discusses the results.

### 8.6.1. Experiment Setup

We implemented the described techniques on top of our conflict-driven learning extension of FAST DOWNWARD (FD) (Helmert, 2006). The source code is publicly available.<sup>1</sup> We consider a similar setup as in Chapters 6 and 7. We focus on proving unsolvability, where dead-end detection has (naturally) the largest impact. We use the UIPC’16 benchmarks, as well as the unsolvable resource-constrained (RCP) benchmarks (Nakhost et al., 2012; Hoffmann et al., 2014). All experiments were run on machines equipped with Intel Xeon E5-2650v3 CPUs, with runtime (memory) limits of 30 minutes (4 GB).

We experiment with  $\mathcal{U}$ -traps for seven unsolvability detectors  $\mathcal{U}$ , mostly taken from the UIPC’16 participants. The details are given below. We evaluate the benefits of each  $\mathcal{U}$  for two different purposes:

- (1) Offline generation of  $\mathcal{U}$ -traps  $\Gamma$ , as per Algorithm 8.1, executed once before search. The conjunction candidates  $C$  are chosen to all conjunctions of size up to  $k = 1$ , respectively  $k = 2$ . We use the generated  $\Gamma$  in two ways. First, we use in the subsequent search *only*  $\mathcal{U}^\Gamma$  for dead-end detection. This serves the purpose of analyzing the impact on the additional  $\mathcal{U}$  information on the traps themselves. Secondly, we combine  $\mathcal{U}^\Gamma$  with  $\mathcal{U}$ , to see the impact of  $\mathcal{U}^\Gamma$  on top of the search with  $\mathcal{U}$ .
- (2) Online learning of a  $\mathcal{U}$ -trap  $\Gamma$ , during search. We compare search with dead-end detection by  $\mathcal{U}$  alone, versus by the combination  $\mathcal{U} + \mathcal{U}^\Gamma$ , enabling  $\mathcal{U}$ -trap learning.

Furthermore, we experiment with a variant of the UIPC’16 winning AIDOS portfolio (Seipp et al., 2016), in which we added  $\mathcal{U}$ -trap online learning to each of its components. In this context, we disable two of AIDOS’ techniques, partial-order reduction and resource variable detection, neither of which is compatible with the  $\mathcal{U}$ -trap learning algorithm. We consider  $\mathcal{U}^C$  with neighbors refinement as a baseline representing

<sup>1</sup><https://doi.org/10.5281/zenodo.6992688>

our other conflict-driven learning variants. In all configurations, apart from the original AIDOS version, we run DFS as per Algorithm 5.2.

### 8.6.2. Unsolvability Detectors

We adapted seven unsolvability detectors  $\mathcal{U}$  towards supporting the  $\mathcal{U}$ -trap construction:

**Baseline** We include results for the naive unsolvability detector  $\mathcal{U}^0$  as a baseline, showing the impact of unsolvability detection by traps alone.

**Critical-Path Heuristics** We experiment with the critical-path unsolvability detectors  $\mathcal{U}^m$  for  $m = 1$  and  $m = 2$  (Haslum and Geffner, 2000). For partial variable assignments  $P$ , we approximate the value of  $\mathcal{U}^m(P)$  through  $\mathcal{U}^m(P^+)$ , where

$$P^+ = P \cup \{v \mapsto d \mid v \notin \text{vars}(P), d \in \mathcal{D}_v\}$$

augments  $P$  by all facts corresponding to the state variables unspecified in  $P$ . Obviously,  $\mathcal{U}^m(P^+) = \infty$  implies  $\mathcal{U}^m(s) = \infty$  for every  $s \supseteq P$ , so this can be used as a sufficient condition.

**Merge-and-Shrink** We experiment with the two most competitive unsolvability merge-and-shrink (M&S) abstractions (Hoffmann et al., 2014; Torralba et al., 2016):  $\text{MSp}$  which computes the perfect unsolvability detector  $\mathcal{U}^*$ , recognizing all dead ends; and  $\text{MSa}$  which imposes a bound on the abstraction size, and hence approximates  $\mathcal{U}^*$ . We include  $\text{MSp}$  for reference only, and do not use it for computing  $\mathcal{U}$ -traps (which would be pointless). For a variable assignment  $P$ ,  $\mathcal{U}^{\text{MSa}}(P)$  is computed by finding all abstract states of the states represented by  $P$  (this can be done effectively given the cascading tables representation of  $\text{MSa}$ ). Then,  $\mathcal{U}^{\text{MSa}}(P) = \infty$  iff every such abstract state is a dead end in the abstract state space.

**PDB** The dead-end PDB heuristic from AIDOS (Pommerening and Seipp, 2016; Seipp et al., 2016). We approximate  $\mathcal{U}^{\text{PDB}}(P)$  by checking whether the PDB contains a pattern  $V$  so that  $V \subseteq \text{vars}(P)$  and  $P$  projected onto  $V$  is recognized as dead end in the respective abstraction.

**State Equation** The state-equation heuristic  $\text{SEQ}$  (Bonet, 2013). Recall from Chapter 7 that for a state  $s$ ,  $\mathcal{U}^{\text{seq}}(s) = \infty$  if the corresponding LP has not optimal solution. To approximate  $\mathcal{U}^{\text{seq}}(P)$ , we set the lower- and upper-bounds of the state-equation constraints so that these constitute lower-, respectively upper-bounds for every state that is represented by  $P$ . Hence, if this LP does not have a solution, the LP corresponding to every  $s$  with  $P \subseteq s$  cannot have one either, i.e.,  $\mathcal{U}^{\text{seq}}(P) = \infty$ .

**Potential Heuristic** The unsolvability potential heuristic  $\text{POT}$  from AIDOS (Seipp et al., 2016). This heuristic is implemented as an extension of the state-equation heuristic, refining that by additional constraints. We approximate  $\mathcal{U}^{\text{pot}}(P)$  following the same idea as for  $\mathcal{U}^{\text{seq}}(P)$ .

### 8.6.3. Discussion

Table 8.2 provides the main coverage results. The modified AIDOS configuration is indicated by “†”. The version that participated in UIPC’16 is shown on the right-hand side of the table, together with  $\text{MSp}$  and dead-end learning via  $\mathcal{U}^C$ . The results for  $\mathcal{U}$ -trap online learning are shown in the middle part of the table (“–” shows the results for  $\mathcal{U}$  alone, “T” for  $\mathcal{U} + \mathcal{U}^\Gamma$ ). The left part shows the results for dead-end detection by  $\mathcal{U}^\Gamma$ , for offline computed  $\mathcal{U}^0$ -traps  $\Gamma$ . Remember that  $\mathcal{U}^0$ -traps correspond exactly to the traps as originally proposed by Lipovetzky et al. (2016), and that offline  $\mathcal{U}^0$ -traps are computed according to exactly that

	$u^0$	$u^1$	$u^2$	MSa	PDB	SEQ	POT	$\Sigma$ UIPC	$\Sigma$ RCP		+	-	$\Delta$ UIPC	$\Delta$ RCP
$u^0$	-	2	9	10	0	3	5	138	37	$u^0$	4	1	+7	+3
$u^1$	2	-	8	10	2	4	6	135	54	$u^1$	4	2	-1	+3
$u^2$	2	1	-	7	2	2	2	99	48	$u^2$	1	7	-28	-28
MSa	6	6	9	-	6	7	7	69	<b>228</b>	MSa	0	15	-65	-30
PDB	2	3	9	10	-	5	5	<b>139</b>	39	PDB	0	2	-3	0
SEQ	1	1	9	10	1	-	5	134	36	SEQ	4	3	-36	+3
POT	2	1	7	8	2	2	-	133	36	POT	1	10	-34	-23

(a)
(b)

Table 8.1.: (a) Comparison of different  $u$  for  $u$ -trap offline construction with  $k=2$ .  $u$  is used only for the trap construction; not for pruning dead ends during search. The entry in the table at row “ $u_1$ ” and column “ $u_2$ ” shows the number of domains on which search with the offline  $u_1$ -trap achieves higher coverage than with the  $u_2$ -trap. “ $\Sigma$  UIPC” and “ $\Sigma$  RCP” provide the total coverage results for the two benchmark sets. (b) Comparison of search using  $u$  alone versus the combination  $u + u^\Gamma$ , for offline constructed  $u^\Gamma$ ,  $k = 2$ . “+” counts the number of domains on which  $u + u^\Gamma$  achieves higher coverage than  $u$ ; vice versa “-” counts the number domains on which  $u + u^\Gamma$  performs worse than  $u$ ; “ $\Delta$  UIPC” and “ $\Delta$  RCP” shows the difference between the total number of instances solved by  $u + u^\Gamma$  and that by  $u$ , over the two benchmark sets.

proposal. Table 8.1 provides additional data for the offline trap configurations, using  $k = 2$ . The results for  $k = 1$  look similar. More detailed coverage results are available in Appendix A.1.

Consider first offline construction. As reflected by the results in Table 8.2, the conjunctions of size  $k = 1$  are not enough to compute a non-empty  $u^0$ -trap in any domain but DocTransfer, effectively turning the respective configuration into blind search. The computed  $u^0$ -traps for  $k = 2$  mainly help in Bottleneck, DocTransfer, and RCP TPP. As indicated by the results in Table 8.1a, the different unsolvability detectors  $u$  have complementary effects on trap generation. Every  $u$  helps in some domains (cf. the “ $u^0$ ” column), yet is also detrimental in other domains (cf. the “ $u^0$ ” row). The latter is due to the additional overhead resulting from the evaluation of  $u$  during trap construction. An extreme example for that is MSa, which vastly increases coverage in the resource-constrained domains, but worsens the results on all other domains considerably. Overall, dead-end detection through an offline computed  $u$ -trap alone cannot compete with current state-of-the-art methods. Considering the combination of  $u$  and  $u^\Gamma$  for dead-end detection, overall, the additional overhead associated with the  $\Gamma$  construction outweighs the gains in terms of search reduction. As shown in Table 8.1b, while the combination can improve coverage over  $u$  alone in at least some domains, for every  $u$  but MSa and PDB, coverage decreases in many others. With respect to the total number of instances solved, the  $u$ -traps turn out to be more detrimental than helpful.

The potential of  $u$ -traps really becomes alive, however, in the online conflict-driven learning setting. State-of-the-art performance is achieved in many domains even for  $\Gamma$  using  $u^0$ , i.e., without any additional unsolvability detector. This vanilla configuration outperforms, in particular, conflict-driven learning via  $u^C$  in its prime discipline, the resource-constrained domains. Coverage in the latter domains can be increased even further through combination with the dead-end PDB heuristic, pushing the respective  $u$ -trap learning configuration to perfect coverage in NoMystery and Rovers. In TPP,  $u$ -trap learning is inferior only to AIDOS’ resource-variable identification component.

$u$ -trap learning improves overall coverage on the resource-constrained domains for every tested  $u$  except for MSa. On the other domains, the overall picture is not as consistently good. The number of domains

		Offline										Online													
		$u^0$		$u^0$		$u^1$		$u^2$		MSa		PDB		SEQ		PoT		AIDoS <sup>†</sup>							
Domain	#	k=1	k=2	-	Γ	-	Γ	-	Γ	-	Γ	-	Γ	-	Γ	-	Γ	-	Γ	-	Γ	AIDoS	MSP	$u^C$	
Unsolvability IPC (UIPC) 2016 Benchmarks																									
BagBarman	20	12	12	12	0	8	0	0	0	4	0	12	0	4	0	4	0	12	0	12	0	0			
BagGripper	25	5	1	6	5	3	3	0	0	3	3	3	3	23	23	3	3	25	23	5	3	3			
BagTransport	29	7	7	7	7	6	7	16	16	6	6	7	7	29	29	24	24	29	29	22	1	7			
Bottleneck	25	10	15	10	8	20	16	21	19	10	4	19	18	25	25	25	25	25	25	25	5	9			
CaveDiving	25	7	7	7	5	7	5	6	5	7	4	7	5	8	6	8	7	9	7	8	3	8			
ChessBoard	23	5	5	5	3	5	3	4	3	5	2	5	3	23	23	23	23	23	23	23	2	2			
Diagnosis	11	4	5	4	6	6	9	5	6	4	4	5	9	4	9	4	8	5	9	5	5	8			
DocTransfer	20	6	11	5	5	7	11	7	7	10	6	12	16	6	10	7	7	15	16	13	5	5			
NoMystery	20	2	2	2	11	2	11	2	8	8	9	11	13	2	11	5	6	11	13	11	11	11			
PegSol	24	24	24	24	18	24	18	21	16	24	16	24	16	24	18	22	20	24	16	24	24	14			
PegSolRow5	15	5	5	5	4	5	4	4	4	5	4	5	4	15	15	15	15	15	15	15	3	4			
Rovers	20	7	7	7	13	7	13	7	12	9	10	12	17	6	12	6	12	12	16	14	15	12			
SlidingTiles	20	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10			
Tetris	20	10	10	10	10	5	5	5	5	5	5	10	10	20	20	20	20	20	20	20	5	5			
TPP	30	17	17	17	24	17	23	15	20	24	23	24	24	14	14	19	19	24	24	29	24	20			
Σ UIPC	327	131	138	131	129	132	138	123	131	134	106	166	155	213	225	195	199	259	246	236	116	118			
Unsolvable Resource-Constrained Planning (RCP) Benchmarks																									
NoMystery	150	26	26	26	143	52	142	83	125	130	134	149	150	16	136	68	79	149	150	149	150	130			
Rovers	150	3	3	3	142	7	139	67	120	111	111	93	150	1	125	1	125	93	150	109	129	144			
TPP	25	5	8	5	19	7	21	8	9	17	12	20	21	1	1	11	11	19	21	25	16	14			
Σ RCP	325	34	37	34	304	66	302	158	254	258	257	262	321	18	262	80	215	261	321	283	295	288			
Σ Total	652	165	175	165	433	198	440	281	385	392	363	428	476	231	487	275	414	520	567	519	411	406			

Table 8.2.: Coverage results (number of instances proved unsolvable). Best results in **bold**. “Offline” shows the results for search with an offline generated  $u^0$ -trap, choosing the conjunctions candidates  $C$  to all conjunctions of size  $k = 1$ , respectively  $k = 2$ . “Online” shows the results for “-”: DFS using only  $u$  for dead-end pruning (no learning); versus “+ $\Gamma$ ”: using  $u + u^\Gamma$ , and enabling  $u$ -trap learning, for different  $u$ ; “ $u^0$ ” naive unsolvability detector; “ $u^1$ ” and “ $u^2$ ”: critical-path unsolvability detectors; “MSa”: unsolvability merge-and-shrink abstraction with size bound; “PDB”: dead-end PDBs of Aidos; “SEQ” unsolvability detection via the state equation; “POT”: potential heuristic of Aidos; “Aidos $^\dagger$ ”: variant of Aidos as described in the text, without “-” versus with “+ $\Gamma$ ”  $u$ -trap learning enabled in all components; “Aidos”: Aidos as in UIPC’16; “MSP” perfect merge-and-shrink unsolvability detector; and “ $u^C$ ” DFS with conflict-driven learning using  $u^C$ .

where  $u$ -trap learning positively (negatively) affects coverage are 9 (6) for  $u^1$ ; 7 (4) for  $u^2$ ; 3 (9) for MSa; 7 (6) for PDB; 6 (3) for SEQ; 5 (3) for POT; and 7 (4) for Aidos $^\dagger$ . Synergistic effects, where the combination of  $u$  with  $u$ -trap learning solves instances not solved by either of the component techniques alone, occur for PDBs in the resource-constrained domains, as well as for  $u^1$ , SEQ, and POT in Diagnosis and DocTransfer.

Figure 8.2 provides additional per-benchmark-instance statistics on the impact of trap-based dead-end learning on the  $u^0$ -baseline: Figure 8.2a compares search space sizes, measured in terms of states visited in search; Figure 8.2b compares the search-space and runtime reduction factors that result from enabling

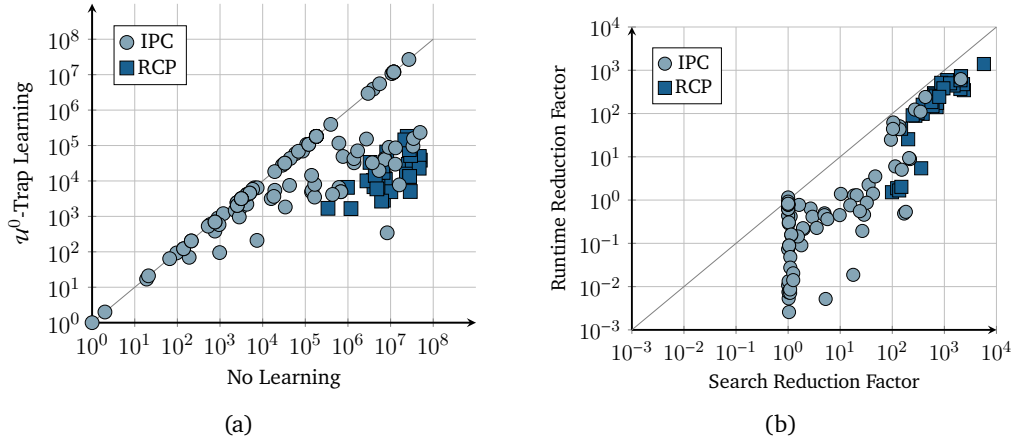


Figure 8.2.: (a) Per-instance based comparison of the search space sizes (number of visited states) between the  $u^0$ -baseline on the  $x$ -axis versus  $u^0$ -trap learning on the  $y$ -axis. (b) Per-instance based comparison of performance-impact measures of  $u$ -trap online learning on the  $u^0$ -baseline: search-space reduction factors on the  $x$ -axis vs. runtime reduction factors on the  $y$ -axis.

trap online learning in the  $u^0$ -baseline. The results for other  $u$  are qualitatively similar. The difference between search and runtime reduction corresponds exactly to the overhead induced by trap evaluation and refinement. In the instances where runtime could not be reduced (points below  $10^0$ ), trap refinement does not generalize at sufficient scale, and thus the overhead outweighs the benefits of trap learning. Extreme examples are the two PegSol domains, for which generalization does not happen at all. In contrast, search effort is reduced by several orders of magnitude in, e.g., DocTransfer, Diagnosis, and the resource-constrained domains.

Regarding AIDOS, our modified variant performs better on the UIPC domains because partial-order reduction and resource variable detection have small positive effects, yet have a large negative impact in BagGripper. Extending AIDOS<sup>†</sup> by  $u$ -trap online learning increases overall coverage, turning it into the overall best configuration in this comparison. The difference emerges from the synergies of  $u$ -trap learning with AIDOS' components as already pointed out. Over the UIPC domains,  $u$ -trap learning is slightly worse overall, but its overall disadvantage is due primarily to BagBarman, where the  $u$ -trap generalization algorithm struggles due to the prohibitive number of (ground) actions in the tasks, and often runs out of time.

## 9. Discussion

We close this part with a brief summary of the presented material. Moreover, we discuss related and possible future work.

### 9.1. Summary

Our work pioneers conflict-driven learning, of sound generalizable knowledge, from conflicts – dead-end states – in forward state-space search. This is founded upon refineable goal-reachability approximations  $\mathcal{U}$  that can be rendered perfect in the limit. We presented search methods for identifying conflicts, and we have shown how to refine different unsolvability detectors to recognize those states, doing so in hopes to avoiding similar conflicts in the remainder of the search. Wrapped in a depth-first search, this reaches the elegance of the conflict-driven learning approach for constraint satisfaction problems, including the ability to refute explored search subtrees, and to immediately backjump, across levels, to the latest decision point responsible for a conflict.

We considered three complementary families of unsolvability detectors: critical-path heuristics  $\mathcal{U}^C$ , LP-based state-equation  $\mathcal{U}^{\text{seq}}$  and potential  $\mathcal{U}^{\text{pot}}$  heuristics, and  $\mathcal{U}$ -traps. We showed how to, given a set of conflict states, refine  $C$  so that  $\mathcal{U}^C$  recognizes these states. Towards curbing  $\mathcal{U}^C$ 's computation overhead as  $C$  keeps growing, we investigated alternative characterizations of  $\mathcal{U}^C$ 's recognized dead-end states, NoGood formulae. We showed that one can actually represent  $\mathcal{U}^C$  dead-end detection exactly through a, worst-case exponentially large, NoGood formula, and presented two practical approaches generating NoGoods dynamically during search, reducing evaluation calls to  $\mathcal{U}^C$  while not getting rid of them entirely.

While the state-equation and potential heuristics do not natively support the necessary refinement operation, we showed that both can be enhanced via fact conjunctions, leveraging well-known  $\Pi^C$  compilation. We showed that these heuristics applied to  $\Pi^C$  converge to the perfect heuristic for suitable  $C$ . Moreover, we developed a refinement algorithm for computing  $C$  suitable for  $\mathcal{U}^{\text{seq}}$  and  $\mathcal{U}^{\text{pot}}$  to recognize previously unrecognized dead-end states.

The setups so far share the limitation that, when taking into account additional sources of goal-reachability information, the learning will eventually, and redundantly, have to re-represent that information. To address this shortcoming, we introduced  $\mathcal{U}$ -traps, an extension of the dead-end trap formalism wrapping around and generalizing an arbitrary other unsolvability detector  $\mathcal{U}$ . We presented an  $\mathcal{U}$ -trap refinement procedure that synergistically combines learning with  $\mathcal{U}$ 's embedded dead-end information.

Our experiments showcased that the learning can lead to tremendous savings in search, provided that three requirements are in place: (1) conflicts are identified quickly, (2) the learned knowledge generalizes, (3) the learning is effective, i.e., the scale of generalization outweighs the learning-related overhead. Such a problem structure is typical for resource-constrained planning, as far as represented by the current benchmarks, but unfortunately, it exists only rarely in other domains, as far as reflected by the competition benchmarks. Thus, beauty contests aside, from a pragmatical point of view our techniques certainly do

not, as they stand, deliver an empirical breakthrough. However, we consider our work to be merely a first foray into forward search conflict-learning techniques, and lots more remains to be explored. We hope and expect our work to be the beginning of the story, not its end.

## 9.2. Related Work

We group the related work into two categories: learning control knowledge during search, and dead-end detection respectively proving unsolvability in classical planning.

### 9.2.1. Learning Search Control Knowledge

Learning search control knowledge from experience has long history in planning. The approaches can be distinguished along two broad types of learning: *inductive* versus *deductive*. Inductive learning uses statistical methods to generalize from a large set of training examples, whereas deductive learning obtains generalizing knowledge by *explaining* individual training examples according to some *domain theory*. Zimmerman and Kambhampati (2003) and Celorrio et al. (2012) provide a, not exactly up-to-date, overview. Inductive learning typically comes without formal guarantees. We focus on the deductive approaches in the following.

Learning search control knowledge by explaining individual examples goes back to early works on the STRIPS planning system (Fikes et al., 1972), which learns *macro-actions* representing entire action sequences that are gleaned from successful searches. These macro-actions may help subsequent searches on related planning tasks by introducing shortcuts in the state spaces. Iba (1989) and Coles and Smith (2007) adapt this approach to learning macro-actions during forward state-space search so to speed up that search.

Minton et al. (1989) drove the idea of explanation-based learning for search further forward as part of their work on the PRODIGY/EBL planning system. PRODIGY/EBL conducts *backward* state-space search purely guided via a database of learned action preference rules. New rules are generated after search by inspecting the generated search space for instances of first-order logic characterizations of different learning *concepts*. Besides featuring concepts to learn from successful searches, like the approaches before it, PRODIGY/EBL is also able to learn from search failures, i.e., search branches that could not be completed to a solution (without including a cycle). The corresponding failure concepts serve as the basis to derive sound action pruning rules, i.e., ones guaranteeing not to prune any solution.

FAILSAFE (Mostow and Bhatnagar, 1987; Bhatnagar and Mostow, 1994) learns action pruning rules during forward state-space search. To foster learning, new rules are generated not only upon search failures, but whenever search is deemed to be *under-constrained*. To foster pruning, FAILSAFE learns overly generic rules, at the price of loosing soundness guarantees. FAILSAFE features a mechanism to relax some of its learned rules so to recover from situations where all solution paths were pruned.

Kambhampati et al. (1996) implement a form of conflict-driven learning in partial-order planning, i.e., in search in the space of plans. A depth limit ensures that the search space remains finite. Conflicts are identified on a local per search node basis via different failure conditions, such as inconsistencies in the constructed partial plans, or once exceeding the depth limit. Sound pruning rules are learned from failures by isolating the individual failure conditions and regressing those through the search decisions. The failure explanation of nodes crossing the depth limit requires additional domain knowledge. If the explanation

is not possible, learning remains sound but becomes incomplete. Kambhampati (1998) later unifies and relates the overall procedure to conflict-driven learning on constraint satisfaction problems.

Conflict-driven learning in state-space search has been studied in a length-bounded setting. GRAPHPLAN's *memoization* technique (Blum and Furst, 1997; Long and Fox, 1999), storing subgoals unreachable at different layers, can be seen as a simple form of NoGood learning. Kambhampati (2000) later extends this by a generalization procedure. Specifically, whenever GRAPHPLAN backtracks from its layer  $L_k$  to layer  $L_{k+1}$  with subgoal  $G_k$ , proving that  $G_k$  is not reachable at layer  $L_k$ , instead of memoizing  $G_k$ , one attempts to extract a small reason  $r_k \subseteq G_k$  sufficient for the unreachability.  $r_k$  is stored, and used to refute other subgoals  $G'_k \neq G_k$ , namely if  $r_k \subseteq G'_k$ , that may be encountered in later iterations. As observed by Suda (2014), this closely resembles PDR (Bradley, 2011; Suda, 2014).

Given the relation between length-bounded reachability in symbolic transition systems and propositional satisfiability (Kautz and Selman, 1999; Biere et al., 1999; Rintanen et al., 2006), via the appropriate compilations, the conflict-driven learning techniques developed in this area (e.g., Dechter, 1986; Dechter, 1990; Prosser, 1993; Silva and Sakallah, 1996; Moskewicz et al., 2001; Beame et al., 2004) apply unmodified.

SIXTHSENSE (Kolobov et al., 2010b) learns to detect dead ends during probabilistic forward state-space search, yet incorporates classical planning as a sub-procedure. Moreover, SIXTHSENSE relies on  $h^2$  (Haslum and Geffner, 2000) for generalizing its pruning rules, and as such cannot learn recognizing dead ends other than the ones recognized by  $h^2$ .

PRP (Muisse et al., 2012b) learns to detect dead ends in FOND planning, but again relies on a classical-planning sub-procedure, and its generalization capabilities are bounded by  $h^1$  (Haslum and Geffner, 2000).

Online heuristic refinement, improving the cost-to-goal estimates during heuristic forward search, has also received some attention. For example, Fickert and Hoffmann (2017) refine the critical-path based  $h^{CFF}$  heuristic during an enforced hill-climbing search to escape local minima. Eifler and Fickert (2018) refine Cartesian abstraction heuristics during A\* search based on inconsistencies in the Bellman equations. In principle, the refinements can result in recognizing new dead ends, but they are not geared at this purpose.

Finally, search using value function refinements via Bellman updates (e.g., Korf, 1990; Reinefeld and Marsland, 1994; Barto et al., 1995; Bonet and Geffner, 2006) will eventually learn that a state is a dead end, yet it does not generalize that knowledge.

In difference to all these approaches, our presented techniques are unique in the sense of (1) learning sound and generalizable state-pruning rules, (2) during forward state-space search, (3) without length bound, and (4) providing the ability to learn to recognize any dead-end state.

### 9.2.2. Dead-End Detection and Proving Unsolvability

Both, dead-end detection and proving unsolvability, have been traditionally neglected in classical planning; the latter has even been completely ignored, the focus being exclusively on solvable problems. First works designing techniques dedicated to dead-end detection appeared in 2013 and 2014 (Bäckström et al., 2013; Hoffmann et al., 2014); proving unsolvability became the center of attention for the first time in 2016, with the inaugural *Unsolvable International Planning Competition (UIPC'16)*.

Dead-end detection has been traditionally treated as byproduct of heuristic functions (e.g., Haslum and Geffner, 2000; Hoffmann and Nebel, 2001; Edelkamp, 2002; Helmert and Domshlak, 2009; Richter and

Westphal, 2010; Helmert et al., 2014). Hoffmann et al. (2014) were the first to break with this tradition. They designed unsolvability detectors based on merge-and-shrink abstractions (Dräger et al., 2009; Helmert et al., 2014; Sievers and Helmert, 2021). In UIPC’16, apart from merge-and-shrink abstractions (Torralba et al., 2016), new unsolvability detectors participated based on *potential heuristics* (Pommerening et al., 2015; Seipp et al., 2016), *pattern databases* (Edelkamp, 2002; Pommerening and Seipp, 2016), and *critical-path heuristics* (Haslum and Geffner, 2000; Haslum, 2016). But, in all those techniques, the unsolvability detector is constructed before search, and remains static during search.

Apart from the mentioned unsolvability detectors, UIPC’16 featured many other approaches tailored for proving unsolvability, such as symbolic search via BDDs (Edelkamp et al., 2015; Torralba, 2016), partial delete-relaxation (Domshlak et al., 2015; Gnad et al., 2016b), and sound pruning techniques (Wehrle and Helmert, 2014; Torralba and Hoffmann, 2015; Torralba and Kissmann, 2015). Yet, none of these incorporate any kind of learning.

Eriksson et al. (2017) have presented a generic proof system that can be plugged into many planning techniques proving unsolvability so to generate unsolvability certificates. Our techniques can provide unsolvability certificates without further modifications. Eriksson et al.’s (2017) proof system yields an *inductive* state set that separates initial and goal states. The structure of our certificates depends on the underlying unsolvability detector, specifying an instantiation that recognizes the initial state as dead end.

### 9.3. Future Work

Regarding future work, lots more remains to be explored. First and foremost, one thing we would particularly like to see is the export of this (kind of) technique to game-playing and model checking, where dead-end detection is at least as, probably even more, important than in classical planning. This works out of the box modulo the applicability of the underlying unsolvability detector. Specifically, the critical-path heuristic and the  $\Pi^C$  compilation rely on a conjunctive subgoal structure. Though, supporting richer formalisms (such as also disjunctions, or even integer arithmetic) should be doable, and there already exists some work in that direction (e.g., Löhr, 2014). Porting  $\mathcal{U}$ -trap learning per-se requires only a suitable adaption of the syntactic progression operation, but questions remain pertaining to how to then exactly integrate additional unsolvability detector  $\mathcal{U}$ .

On the algorithmic side of things, a major open point is the *learning utility problem* (e.g., Minton et al., 1989): the relation between learning overhead and impact on search. Learning can only be effective if the extent of the latter dominates the former. One possibility to tackle this problem could be a *safety belt* mechanism as already used for other search pruning techniques; if we could predict, during searching, what the learning utility will be, then we could enable/disable learning accordingly. This poses the question of how to measure the impact on search without knowing exactly how large the search space would have been if we had not pruned a state. Moreover, as search is progressing and the learned information keeps growing, the unsolvability detector refinements and evaluations become continuously more expensive, while not all information that was ever learned remains useful for the remaining search. Periodically cleaning up the learned knowledge has shown to be highly critical for the efficiency of clause learning SAT solvers (e.g., Audemard and Simon, 2009). Challenging questions in our setting pertain to deciding when to start a cleanup, and more importantly, how to even decide what information to forget.

We have covered a range of different unsolvability detectors, and seen that these methods can perform complementary on different domains. An interesting question remaining is how to effectively combine these methods? Moreover, an exciting line of research is the design of different refinement methods, be it

for considered, or for other dead-end detectors. In particular, can we refine merge-and-shrink unsolvability heuristics on the fly? For the critical-path heuristics, open questions pertain to different strategies for selecting the conjunctions resolving a flaw. Regarding the state-equation and potential heuristics, there remains the quest for alternative ways to incorporate conjunctions (or other information) without inducing an exponential blow-up like  $\Pi^C$ , while also not constraining the interactions between the individual elements like partial variable merges do. Regarding  $\mathcal{U}$ -traps, it could be worthwhile to investigate more expressive state-set representations, such as using Cartesian products over subsets of variable domains (e.g., Seipp and Helmert, 2018) instead of conjunctions.

Another point on the agenda is the integration of other search-space reduction techniques. For safe reduction techniques (ones guaranteeing to preserve at least one solution path, if one exists), this is principally possible already, although one needs to pay attention to particularities arising from some of the presented unsolvability-detector refinement algorithms (specifically, the recognized-neighbors property). Yet, when using a transitive unsolvability detector for learning, there again emerges the problem of duplicating knowledge embedded in the reduction technique already. This brings up the question whether there exist formalisms that, similar to  $\mathcal{U}$ -traps, allow to synergistically combine learning with other search-space reduction techniques.

Interesting questions also appear in the context of unsafe search-space reduction methods, such as in *width-based search* (Lipovetzky and Geffner, 2012), in *under-approximation refinement* (Heusner et al., 2014), or in *partial grounding* approaches (Gnad et al., 2019). For instance, width-based search uses a predefined set of state formulae, considering only states that satisfy some formula not satisfied by any state seen before it. Conflicts – states for which all goal paths have been pruned – abound, the more confined the chosen state formulae. On the one hand, this enlarges the potential for conflict-driven learning. Open questions pertain to unsolvability predictors that reason about the still available state formulae, i.e., given a state, how to derive sufficient conditions to whether there exists a non-pruned goal path, and can we refine these predictions upon finding unrecognized conflicts? On the other hand, conflict-driven learning could be an effective means to make width-based search complete. Namely, when the current set of formulae is too aggressive in the sense of pruning all solution paths, there is the question of how to refine the set of formulae before reiterating search. The analysis of conflicts could direct this refinement.

Last but not least, a promising direction for future work is lifting the presented techniques to weaker forms of conflicts. A particularly appealing variant is given by *loop-back states*, i.e., states  $s$  all whose solution paths need to pass through some ancestor of  $s$  in the search space. While loop-back states need not necessarily be dead ends, their exploration in search is equally wasteful. Loop-back states can be expected to appear much more frequently than dead-end states, addressing one of the main bottlenecks of the conflict-driven learning approach in its current form. Open questions pertain to the development of unsolvability-detector equivalents for recognizing loop-back states, alongside with corresponding refinement procedures.



## State-Space Search Methods for Goal-Probability Analysis in Probabilistic Planning

When acting under uncertainty, it may not be possible to achieve the goal while avoiding dead-end states under all possible circumstances. An important objective in such settings is *MaxProb*, determining the maximal probability with which the goal can be reached, and identifying a policy achieving that probability. This part of the thesis is concerned with MDP probabilistic planning methods solving variants of this objective optimally.

While this setup and objective certainly is relevant, there has been little work towards developing solvers, the main effort being made by Kolobov et al. (2011). Hou et al. (2014) consider several variants of topological value iteration, solving MaxProb but necessitating to build the entire reachable state space. Other works addressing goal-probability maximization do not aim at guaranteeing optimality (e.g., Teichteil-Königsbuch et al., 2010; Camacho et al., 2016). MDP heuristic search has the potential to find an optimal solution without building the entire state space, but cannot be applied to MaxProb directly due to the possibility of having 0-reward cycles. To remedy this limitation, Kolobov et al. (2011) devised the FRET framework, which admits heuristic search, yet requires several iterations of complete searches. Kolobov et al.’s contribution is mainly theoretical – considering not only MaxProb but a much larger class of MDPs – and their empirical evaluation serves merely as a proof of concept. Given this

- (i) *What is actually the empirical state of the art in MDP heuristic search for MaxProb?*
- (ii) *What about simpler yet still relevant special cases, and weaker objectives, that may be easier to solve?*

Regarding (ii), a relevant special case is *acyclic planning*, covering, e.g., *limited-budget planning*, or *penetration testing* models, where FRET is not needed. Moreover, as alternatives to MaxProb, it might be reasonable to ask for policies achieving some minimal goal-probability threshold (*AtLeastProb*) or accuracy with respect to MaxProb (*ApproxProb*). These weaker objectives enable what we will refer to as *early termination*.

Regarding (i), we develop a large MDP state-space search tool box composed of (a) *search algorithms*: variants of known and new heuristic search algorithms, equipped with early-termination criteria, tie-breaking strategies fostering early termination, a new FRET variant, and heuristic search algorithms without FRET dependency; (b) *state-space reduction methods*: we consider probabilistic bisimulation, and dead-end pruning via classical-planning heuristic functions; (c) *goal-probability heuristics*: we develop MaxProb variants of the occupation-measure and probabilistic operator-counting heuristics (Trevizan et al., 2017b). We systematically explore the behavior of the resulting design space on a large benchmark suite we design for that purpose, clarifying the state of the art, and demonstrating significant benefits of our new algorithm variants.

This part is based on the material presented in (Steinmetz et al., 2016a).



# 10. Probabilistic Planning

This chapter establishes the formal basis of goal-probability analysis in MDP-based probabilistic planning. The structure broadly follows that of Chapter 2. We start with the syntactic representation of probabilistic planning tasks, then define their semantics by a means of MDPs. For the syntactic specification, we build upon the classical planning formalism presented in Section 2.1.2, extending it by the support of non-deterministic action outcomes. We introduce a limited cost budget variant, which plays an important role in our experiments. We revisit the central concepts in the realm of MDPs, and define our goal-probability objectives in this context. For an in-depth discussion of the theory of MDPs, we refer to standard textbooks (e.g., Puterman, 1994).

## 10.1. Probabilistic FDR Planning Tasks

This part of the thesis is entirely based on the FDR formalism. Probabilistic planning tasks differ from the classical ones (Definition 2.4) in only the action definition. We provide the full definition for completeness:

**Definition 10.1** (Probabilistic FDR Task). *A **probabilistic FDR task** is a tuple*

$$\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$$

*with components*

- $\mathcal{V}$  is a finite set of variables, each  $v \in \mathcal{V}$  has a finite domain  $\mathcal{D}_v$ .
- $\mathcal{A}$  is a finite set of actions. Each action  $a \in \mathcal{A}$  has a precondition  $\text{pre}(a)$ , a variable assignment, and a non-empty finite set of **outcomes**  $\text{out}(a)$ . Each outcome  $o \in \text{out}(a)$  is associated with a non-zero **outcome probability**  $\text{prob}(o) \in (0, 1]$ , and the **outcome effect**  $\text{eff}(o)$ , which is a variable assignment. We require that the outcome probabilities sum up to 1:  $\sum_{o \in \text{out}(a)} \text{prob}(o) = 1$ .
- $\mathcal{I}$  is the initial state, a complete variable assignment.
- $\mathcal{G}$  is the goal, a variable assignment.

We sometimes refer to  $\Pi$ 's individual components by  $\mathcal{V}^\Pi$ ,  $\mathcal{A}^\Pi$ ,  $\mathcal{I}^\Pi$ , and  $\mathcal{G}^\Pi$ . Action costs are omitted, as these are not relevant for goal-probability analysis. We say that an action is **probabilistic** if it has more than one outcome, and we call it **deterministic** otherwise. Facts  $\mathcal{F}^\Pi$ , states  $\mathcal{S}^\Pi$ , and the notion of action applicability are defined just as for classical FDR tasks. For a state  $s \in \mathcal{S}^\Pi$ , applicable action  $a \in \mathcal{A}(s)$ , and outcome  $o \in \text{out}(a)$ , we define by  $s \llbracket o \rrbracket = (s \circ \text{eff}(o))$  the result of  $o$  in  $s$ . Different outcomes of an action may result in different states. Hence, the result of applying actions is no longer guaranteed to be a unique state, but depends on the action outcome. The choice of the action outcome is not controllable. One can think of it as being selected by external entity. The outcome probabilities provide an estimation of the likelihood that a particular outcome occurs.

**Example 10.1.** Reconsider the rover running example from Section 1.2, and the classical FDR planning task encoding (Example 2.2). Assume that we have incomplete knowledge about the surface structure. In particular, some connections between the locations might be impassable by the rover. To take into account this uncertainty inside the planning task, we introduce for each pair of connected locations  $x$  and  $y$  a new variable  $\text{con}(x, y)$  with domain  $\mathcal{D}_{\text{con}(x, y)} = \{ \text{pas}, \text{imp}, \text{unk} \}$ . The value  $\text{unk}$  represents unknown knowledge about the connection status, the values  $\text{pas}$  (passable) and  $\text{imp}$  (impassable) reflect information that rover has discovered. The collect and drop actions remain deterministic, having a single outcome whose effect is defined just as in the classical case. To reflect the connection uncertainty, the move actions from before are replaced by two variants

- *observeAndMove* is probabilistic, and applicable only if the status of the connection has not been unveiled yet:  $\text{pre}(\text{observeAndMove}(x, y, k)) = \{ \text{con}(x, y) \mapsto \text{unk}, \text{rov} \mapsto x, \text{bat} \mapsto k \}$ . There are two probabilistic outcomes  $\text{out}(\text{observeAndMove}(x, y, k)) = \{ o_{\text{pas}}, o_{\text{imp}} \}$ , the connection being impassable  $\text{eff}(o_{\text{imp}}) = \{ \text{con}(x, y) \mapsto \text{imp}, \text{bat} \mapsto k - 1 \}$  and vice versa  $\text{eff}(o_{\text{pas}}) = \{ \text{con}(x, y) \mapsto \text{pas}, \text{rov} \mapsto y, \text{bat} \mapsto k - 1 \}$ . The outcome probabilities could follow any prior belief on the availability of each individual connection. For the sake of the example, we simply assume that  $\text{prob}(o_{\text{imp}}) = \frac{1}{5}$  and  $\text{prob}(o_{\text{pas}}) = \frac{4}{5}$ .
- *move* is deterministic, but requires that the connection is known to be passable. Formally, the action has precondition  $\text{pre}(\text{move}(x, y, k)) = \{ \text{con}(x, y) \mapsto \text{pas}, \text{rov} \mapsto x, \text{bat} \mapsto k \}$ , and has a single outcome  $\text{out}(\text{move}(x, y, k)) = \{ o \}$  with effect as in the classical variant  $\text{eff}(o) = \{ \text{rov} \mapsto y, \text{bat} \mapsto k - 1 \}$ , and outcome probability  $\text{prob}(o) = 1$ .

Sometimes it can be useful to constrain the space of solutions by a finite cost budget. For instance, such constraints are naturally given by consumable resources, as in resource-constrained planning (e.g., Haslum and Geffner, 2001; Nakhost et al., 2012; Coles et al., 2013). Moreover, step-limited goal reachability objectives, such as finite-horizon goal-probability maximization, constitute a special case. Limited-budget probabilistic FDR tasks allow to specify such budget explicitly:

**Definition 10.2** (Limited-Budget Probabilistic FDR Task). A **limited-budget probabilistic FDR task** is a tuple

$$\Pi_b = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}_b, \mathcal{G} \rangle$$

where  $\mathcal{V}$ ,  $\mathcal{A}$ , and  $\mathcal{G}$  are defined as before, and

- $\mathbf{c}: \bigcup_{a \in \mathcal{A}} \text{out}(a) \rightarrow \mathbb{R}_0^+$  is the **outcome cost function**.
- $\mathcal{I}_b$  is a complete assignment to  $\mathcal{V} \cup \{ \mathbf{b} \}$ , and  $\mathcal{I}_b[\mathbf{b}] \in \mathbb{R}_0^+$  gives the **initial budget**.

The states  $\mathcal{S}^{\Pi_b}$  of  $\Pi_b$  are the complete assignments to  $\mathcal{V} \cup \{ \mathbf{b} \}$  with  $\mathcal{D}_b = \mathbb{R}$ . The value  $s[\mathbf{b}]$  in state  $s \in \mathcal{S}^{\Pi_b}$  denotes the **remaining budget** in  $s$ .

The facts  $\mathcal{F}^{\Pi_b}$  of  $\Pi_b$  are defined as before, ignoring the budget variable. We also continue to identify states  $s \in \mathcal{S}^{\Pi_b}$  by sets of facts, specifying the remaining budget  $s[\mathbf{b}]$  in addition. The outcome cost function defines how the remaining budget gets updated. An action  $a \in \mathcal{A}$  is applicable in a state  $s \in \mathcal{S}^{\Pi_b}$  if  $\text{pre}(a) \subseteq s$  and  $s[\mathbf{b}] \geq 0$ . If applicable, the result of outcome  $o \in \text{out}(a)$  is defined as

$$s[o] = (s \circ \text{eff}(o)) \circ \{ \mathbf{b} \mapsto (s[\mathbf{b}] - \mathbf{c}(o)) \}$$

In words, the FDR state variables  $\mathcal{V}$  are updated according to the effect  $\text{eff}(o)$ , as before, and the remaining budget  $\mathbf{b}$  is updated as per the outcome cost  $\mathbf{c}(o)$ . Note that the budget value may become negative. We will come back to this in the semantic definition in the next section.

**Example 10.2.** The battery level in Example 10.1 can be seen as a limited budget. To obtain a limited-budget probabilistic FDR variant, we remove *bat* from FDR state variables. The initial battery level becomes the initial budget  $\mathcal{I}_b[\mathbf{b}] = 2$ . Actions *observeAndMove* and *move* are no longer parameterized in the current battery level. The effects on *bat* are replaced by the outcome cost function, which assigns 0 to all outcomes but those of the move actions, defining  $\mathbf{c}(o) = 1$  for all their outcomes. The preconditions on *bat* are implicitly enforced as moving is no longer possible once the remaining budget becomes negative.

Definition 10.2 can be straightforwardly extended to multiple budgets and cost functions. In principle, all techniques presented later on can be easily adapted to the multiple budget case as well. For the sake of simplicity, we however restrict ourselves to a single budget only in the following.

Probabilistic (limited-budget) FDR tasks can be related back to classical FDR tasks via a *determinization* operation (Bonet and Geffner, 2005; Jimenez et al., 2006):

**Definition 10.3** (All-Outcomes Determinization). Let  $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  be a probabilistic FDR task. The **all-outcomes determinization** (short **determinization**) of  $\Pi$  is given by the classical FDR planning task  $\Pi_D = \langle \mathcal{V}, \mathcal{A}_D, \mathcal{I}, \mathcal{G} \rangle$ , with **determinized actions**  $\mathcal{A}_D = \{ a^o \mid a \in \mathcal{A}, o \in \text{out}(a) \}$ , where  $\text{pre}(a^o) = \text{pre}(a)$  and  $\text{eff}(a^o) = \text{eff}(o)$ . Additionally, if  $\Pi$  is a budget-limited task, we define the cost of the determinized action as  $\mathbf{c}(a^o) = \mathbf{c}(o)$ .

The all-outcomes determinization represents every probabilistic action outcome by a deterministic action, pretending to be always able to choose the desired action outcome. There also exist other variants, most notably *most-likely outcome determinization*, choosing the outcomes to be represented more selectively. Yet, an exhaustive outcome treatment is a requirement of the techniques that will be presented later on.

## 10.2. Markov Decision Processes

Markov Decision Processes (MDPs) are a common framework to describe systems with stochastic behavior. MDPs can be defined in a multitude of ways. Common to all definitions is some form of *states*, with transitions between states via some form of *actions*. Transitions are non-deterministic in that taking an action in a state may result in one of potentially many states, some more, some less likely as specified by the transition probability function. To reflect the transition uncertainty, MDP optimization objectives usually ask for the maximization or minimization of the *expected* value of some optimization function. Two categories are commonly studied in literature: the maximization of a reward function (e.g., Puterman, 1994); and reachability based objectives (e.g., Bertsekas, 1995; Kolobov et al., 2011), such as expected-cost minimization. In this thesis, we will exclusively deal with the latter kind. Our MDP definition closely follows that of labeled transition systems (cf. Definition 2.5):

**Definition 10.4** (Goal-Oriented MDP). A **goal-oriented MDP** is tuple

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$$

with components

- $\mathcal{S}$  is a finite set of states.
- $\mathcal{A}$  is a finite set of action symbols.
- $\mathcal{P}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$  is the **transition probability function** such that for each  $s \in \mathcal{S}$  and  $a \in \mathcal{A}$

$$\sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') \in \{0, 1\}$$

- $s_I \in \mathcal{S}$  is the initial state.
- $\mathcal{S}_* \subseteq \mathcal{S}$  are the goal states.

Since goal-oriented MDPs are the only form of MDPs that we consider, we will be simply calling them MDPs, and make a distinction to other classes of MDPs only if relevant for the discussion.

We extend the notions introduced for labeled transition systems as follows. We say that there is **deterministic transition** from  $s$  to  $s'$  via  $a$  if  $\mathcal{P}(s, a, s') = 1$ . In this case, we also call  $s'$  the **deterministic successor** of  $s$  under  $a$ . Vice versa, if  $\mathcal{P}(s, a, s') > 0$  but  $\mathcal{P}(s, a, s') < 1$ , we call it a **probabilistic transition**, and  $s'$  a **probabilistic successor** of  $s$  under  $a$ . The set of all  $a$ -successors of a state  $s$  in  $\mathcal{M}$  then is

$$\text{Succ}[\mathcal{M}](s, a) = \{s' \in \mathcal{S} \mid \mathcal{P}(s, a, s') > 0\}$$

The set of all successors of  $s$  in  $\mathcal{M}$  is

$$\text{Succ}[\mathcal{M}](s) = \bigcup_{a \in \mathcal{A}} \text{Succ}[\mathcal{M}](s, a)$$

Similarly, the predecessors of  $s$  in  $\mathcal{M}$  are the states

$$\text{Pred}[\mathcal{M}](s) = \{s' \in \mathcal{S} \mid \mathcal{P}(s', a, s) > 0 \text{ for some } a \in \mathcal{A}\}$$

The transitive closure  $\text{Succ}^+[\mathcal{M}](s)$  yields the set of all descendants of  $s$ .  $\text{Pred}^+[\mathcal{M}](s)$  gives the set of all ancestors of  $s$ . The states reachable from  $s$  in  $\mathcal{M}$  are given by  $\mathcal{R}[\mathcal{M}](s) = \text{Succ}^+[\mathcal{M}](s) \cup \{s\}$ . We denote the set of actions labeling an outgoing transition from  $s$  by

$$\mathcal{A}[\mathcal{M}](s) = \{a \in \mathcal{A} \mid \text{Succ}[\mathcal{M}](s, a) \neq \emptyset\}$$

We omit  $[\mathcal{M}]$  if  $\mathcal{M}$  is clear from the context.

A state  $s$  is a dead end if no goal state is reachable from  $s$ , i.e., if  $\mathcal{R}(s) \cap \mathcal{S}_* = \emptyset$ . The **terminal states** in  $\mathcal{M}$  are the states without outgoing transitions:

$$\mathcal{S}_\perp = \{s \in \mathcal{S} \mid \mathcal{A}(s) = \emptyset\}$$

A path in  $\mathcal{M}$  is a finite or infinite alternating sequence of states and actions  $\sigma = \langle s_0, a_1, s_2, a_2, \dots \rangle$  such that  $\mathcal{P}(s_i, a_i, s_{i+1}) > 0$  holds for all  $i \geq 0$ . A finite path  $\langle s_0, a_1, s_2, \dots, a_n, s_n \rangle$  is a cycle if  $n \geq 1$  and  $s_n = s_0$ . Cycles of the form  $\langle s, a, s \rangle$  are also called **self-loops**.  $\mathcal{M}$  is **acyclic** if it does not contain any cycle.

An MDP  $\mathcal{M}' = \langle \mathcal{S}', \mathcal{A}', \mathcal{P}', s'_I, \mathcal{S}'_* \rangle$  is a subgraph of  $\mathcal{M}$  if (i)  $\mathcal{S}' \subseteq \mathcal{S}$ , and (ii)  $\mathcal{A}' \subseteq \mathcal{A}$ , and (iii)  $\mathcal{S}'_* \subseteq \mathcal{S}_*$ , and (iv) it holds for all  $s, s' \in \mathcal{S}'$  and  $a \in \mathcal{A}'$  that  $\mathcal{P}'(s, a, s') \in \{0, \mathcal{P}(s, a, s')\}$ . Notice that the combination of (iv) and the requirement  $\sum_{s' \in \mathcal{S}'} \mathcal{P}'(s, a, s') \in \{0, 1\}$  in the MDP definition only leaves the choice of either including a transition entirely or not at all, i.e.,  $\emptyset \subset \text{Succ}[\mathcal{M}'](s, a) \subset \text{Succ}[\mathcal{M}](s, a)$  is not possible.

Let  $\emptyset \subset S \subseteq \mathcal{S}$  be a non-empty subset of states. The subgraph of  $\mathcal{M}$  induced by  $S$  is given by  $\mathcal{M}|_S = \langle S, \mathcal{A}, \mathcal{P}|_S, s'_I, \mathcal{S}_*|_S \rangle$ , where  $\mathcal{S}_*|_S = \mathcal{S}_* \cap S$ , for every  $s, s' \in S$  and  $a \in \mathcal{A}$ :  $\mathcal{P}|_S(s, a, s') = \mathcal{P}(s, a, s')$  if  $\text{Succ}[\mathcal{M}](s, a) \subseteq S$ , and  $\mathcal{P}|_S(s, a, s') = 0$  otherwise. The choice of the initial state is not important. Let  $s \in \mathcal{S}$  be some state. The subgraph of  $\mathcal{M}$  reachable from  $s$  is the subgraph induced by the states reachable from  $s$ , i.e.,  $\mathcal{M}|_{\mathcal{R}(s)}$ . Note that  $\mathcal{P}|_{\mathcal{R}(s)}$  is identical to  $\mathcal{P}$  for all states  $s' \in \mathcal{R}(s)$ , since  $\text{Succ}[\mathcal{M}](s', a) \subseteq \mathcal{R}(s)$  holds by definition. The subgraph reachable from the initial state,  $\mathcal{M}|_{\mathcal{R}(s_I)}$ , is called the reachable subgraph of  $\mathcal{M}$ .

Finally, the notion of *end components* (Courcoubetis and Yannakakis, 1995; de Alfaro, 1997) lifts strongly connected components to the probabilistic setting:

**Definition 10.5** (End Component). A subgraph  $\mathcal{M}'$  of  $\mathcal{M}$  over a non-empty set of states  $\mathcal{S}'$  is an **end component (EC)** of  $\mathcal{M}$  if it holds for all pairs  $s, s' \in \mathcal{S}'$  that  $s \in \text{Succ}[\mathcal{M}'](s')$ .

Abusing notion, we call a set of states  $S \subseteq \mathcal{S}$  an end component if there exists an end component  $\mathcal{M}'$  with states  $\mathcal{S}' = S$ . Notice that end components are SCCs closed under probabilistic branching. More concretely, while the notion of strongly connected component only requires the existence of some path between each pair of states from  $\mathcal{S}'$ , end components require that these paths must be formed from probabilistic transitions that remain within the component, i.e., where  $\sum_{s' \in \mathcal{S}'} \mathcal{P}(s, a, s') = 1$ .

### 10.3. Probabilistic State Space

Having the basic MDPs notions in place, we are now ready to define the semantics of probabilistic planning tasks. The state space induced by a probabilistic FDR task is defined analogously to the classical case, substituting the transition relation by transition probabilities:

**Definition 10.6** (Probabilistic State Space). Let  $\Pi$  be a probabilistic FDR task. The **probabilistic state space** induced by  $\Pi$  is the MDP

$$\mathcal{M}^\Pi = \langle \mathcal{S}^\Pi, \mathcal{A}^\Pi, \mathcal{P}^\Pi, s_I^\Pi, \mathcal{S}_*^\Pi \rangle$$

where

- The transition probabilities are defined as follows. Let  $s, s' \in \mathcal{S}^\Pi$  and  $a \in \mathcal{A}^\Pi$ . If  $a \notin \mathcal{A}^\Pi(s)$ , then  $\mathcal{P}^\Pi(s, a, s') = 0$ . Otherwise,

$$\mathcal{P}^\Pi(s, a, s') = \sum_{o \in \text{out}(a): s' = s[o]} \text{prob}(o)$$

aggregates the probabilities of all outcomes that result in  $s'$ ; defining  $\sum_\emptyset = 0$ .

- $s_I^\Pi = I^\Pi$  is the initial state of  $\Pi$ .
- $\mathcal{S}_*^\Pi = \{s \in \mathcal{S}^\Pi \mid \mathcal{G}^\Pi \subseteq s\}$  are the states that satisfy the goal of  $\Pi$ .

The state space induced by a limited-budget task differs only in the special treatment of states with negative remaining budget:

**Definition 10.7** (Limited-Budget Probabilistic State Space). Let  $\Pi_b$  be a limited-budget probabilistic FDR task. The **probabilistic state space** induced by  $\Pi_b$  is the MDP

$$\mathcal{M}^{\Pi_b} = \langle \mathcal{S}^{\Pi_b}, \mathcal{A}^{\Pi_b}, \mathcal{P}^{\Pi_b}, s_I^{\Pi_b}, \mathcal{S}_*^{\Pi_b} \rangle$$

whose components are defined similarly to Definition 10.6 but:

- $\mathcal{S}_*^{\Pi_b} = \{s \in \mathcal{S}^{\Pi_b} \mid \mathcal{G}^{\Pi_b} \subseteq s, s[b] \geq 0\}$  are the states that satisfy the goal of  $\Pi_b$  and whose remaining budget is not negative.

Note that limited-budget tasks  $\Pi_b$  have infinitely many states. Thus, the probabilistic state space is actually not an MDP as per Definition 10.4. However, since the initial budget is bounded, all outcomes have non-negative cost, and states with negative remaining budget are terminal, the subgraph  $\mathcal{M}^{\Pi_b}|_{\mathcal{R}(s_I)}$  reachable from the initial state is guaranteed to be finite. In other words,  $\mathcal{M}^{\Pi_b}|_{\mathcal{R}(s_I)}$  fits our MDP definition. As we will exclusively consider algorithms that work on this reachable subgraph, we glance over this subtlety in the following. Finally, observe that if  $c(o) > 0$  holds for all outcomes  $o$ , then the remaining budget strictly

decreases along every transition, and therefore  $\mathcal{M}^{\Pi_b}|_{\mathcal{R}(s_I)}$  is acyclic. We will consider acyclic MDPs as a special case.

## 10.4. Policies

As opposed to classical planning, the applications of actions in an MDP can non-deterministically result in one of potentially many states. Formulating a goal reaching strategy hence necessitates a notion that supports the specification of appropriate reactions to the arising contingencies. Common definitions of such strategies differ slightly depending on the MDP objective. For the scope of this thesis, it suffices to consider the most simplest variant, where the choice of what to do next must be unique (deterministic), and it must solely depend on the current state (memoryless). In our context, there always exists an optimal solution (defined below), satisfying these properties (see Bertsekas, 1995).

**Definition 10.8** (Policy). *Let  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  be an MDP. A **policy** for  $\mathcal{M}$  is a partial function  $\pi: (\mathcal{S} \setminus \mathcal{S}_*) \rightarrow \mathcal{A}$  such that  $\pi(s) \in \mathcal{A}(s)$  whenever  $\pi(s)$  is defined.*

If  $\pi(s)$  is not defined, we also write  $\pi(s) = \perp$ . The execution of a policy implicitly defines a subgraph of the original MDP, with transitions as determined by the policy:

**Definition 10.9** (Policy Graph). *The **policy graph** induced by  $\pi$  is the subgraph  $\mathcal{M}^\pi = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}^\pi, s_I, \mathcal{S}_* \rangle$  of  $\mathcal{M}$  where*

$$\mathcal{P}^\pi(s, a, s') = \begin{cases} \mathcal{P}(s, a, s') & \text{if } \pi(s) = a \\ 0 & \text{otherwise} \end{cases}$$

The set of terminal states in  $\mathcal{M}^\pi$  is denoted  $\mathcal{S}_\perp^\pi$ . Suppose  $s \in \mathcal{S}$ . We denote by  $\mathcal{R}^\pi(s)$  the set of states reachable from  $s$  via  $\pi$ , i.e.,  $\mathcal{R}^\pi(s) = \mathcal{R}[\mathcal{M}^\pi](s)$ . The terminal states reachable from  $s$  then are  $\mathcal{S}_\perp^\pi|_{\mathcal{R}^\pi(s)} = \mathcal{S}_\perp^\pi \cap \mathcal{R}^\pi(s)$ . We say that  $\pi$  is **closed** with respect to  $s$  if  $\mathcal{S}_\perp^\pi|_{\mathcal{R}^\pi(s)} \subseteq \mathcal{S}_\perp \cup \mathcal{S}_*$ , i.e., if it holds for all states  $s'$  reachable from  $s$  via  $\pi$ ,  $s' \in \mathcal{R}^\pi(s)$ , that  $\pi(s') \neq \perp$  or  $s' \in \mathcal{S}_* \cup \mathcal{S}_\perp$ . A closed policy is a policy closed with respect to  $s_I$ .

## 10.5. Stochastic Shortest Path Problems

Stochastic shortest path problems (short SSPs) (Bertsekas, 1995) constitute the most thoroughly studied subclass of goal-oriented MDPs. The huge conglomerate of effective MDP heuristic search approaches, parts of which we will adapt in the course of the thesis, is founded in the SSP assumptions. We provide a definition suited to our context:

**Definition 10.10** (SSP). *Let  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  be an MDP.  $\mathcal{M}$  is an **SSP** if it holds for every policy  $\pi$  and reachable state  $s \in \mathcal{R}(s_I)$  that  $\mathcal{R}^\pi(s) \cap (\mathcal{S}_* \cup \mathcal{S}_\perp^\pi) \neq \emptyset$ .*

In words, SSPs require that the execution of every policy from any state must eventually end in an *absorbing* state, i.e., goal state or terminal state in the corresponding policy graph. In particular, notice that acyclic MDPs trivially satisfy the SSP condition, as every path must end in a terminal or goal state by definition.

## 10.6. Goal-Probability Value Function

The quality of a policy is expressed as a value function:

**Definition 10.11** (Value Function). A **value function** in  $\mathcal{M}$  is a function  $V: \mathcal{S} \rightarrow [0, 1]$ .

The range is restricted to the interval  $[0, 1]$ , as we are only considering goal-probability objectives, and these are the only values that make sense in that context. Abusing notation, we write  $V \bowtie V'$  for  $\bowtie \in \{<, \leq, =, \geq, >\}$  if it holds  $V(s) \bowtie V'(s)$  for all  $s \in \mathcal{S}$ .

We follow Kolobov et al. (2011), and specify the goal reaching probabilities of a policy as the maximal non-discounted expected reward where reaching the goal gives reward 1 and all other rewards are 0:

**Definition 10.12** (Policy Value Function). Let  $\pi: (\mathcal{S} \setminus \mathcal{S}_*) \rightarrow \mathcal{A}$  be a policy for  $\mathcal{M}$ . The **goal-probability value function**, or simply **value function** or **goal probabilities**, of  $\pi$  is the piecewise minimal function  $V^\pi: \mathcal{S} \rightarrow [0, 1]$  such that:

$$V^\pi(s) = \begin{cases} 1 & \text{if } s \in \mathcal{S}_* \\ 0 & \text{if } s \notin \mathcal{S}_* \text{ and } \pi(s) = \perp \\ \sum_{s' \in \mathcal{S}} \mathcal{P}(s, \pi(s), s') V^\pi(s') & \text{otherwise} \end{cases}$$

The execution of a policy stops at goal states, or states in which the policy is not defined. The former states trivially reach the goal with a probability of 1. Vice versa, the policy never reaches the goal from the latter states, i.e., their goal-probability value is 0. For all other states, the goal-probability value is given by the expected goal probability achieved by following the (probabilistic) transition chosen by the policy.

**Example 10.3.** Consider the running example (Example 10.1). There are three locations  $B, A_1, A_2$ , all connected to each other. The rover is initially at  $A_1$ , the sample  $samp_1$  at  $A_1$ , and the sample  $samp_2$  at  $A_2$ . Assume that the goal is  $\{s_{amp_2} \mapsto B\}$ . Suppose that the initial battery level is 3, and that all connections to and from the base are initially known to be traversable. Consider the following two strategies for the initial state:

$\pi_1$  observe and move from  $A_1$  to  $A_2$ , abstain if impassable; collect  $samp_2$  at  $A_2$ ; move to the base  $B$ ; drop  $samp_2$ ,

$\pi_2$  take the detour via  $B$  to get to  $A_2$ ; collect the sample, move back to  $B$ , and drop the sample.

Since connections to the base are passable, the second policy is guaranteed to satisfy the goal:  $V^{\pi_2}(s_f) = 1$ . On the other hand, moving directly from  $A_1$  to  $A_2$  in  $\pi_1$  succeeds only if this connection is indeed passable. If this is the case, then the goal will be satisfied too. If not, the policy stops. The goal probability is  $V^{\pi_1}(s_f) = \frac{4}{5} \cdot 1 + \frac{1}{5} \cdot 0 = \frac{4}{5}$ . Note that  $\pi_1$  cannot be extended to increase the goal probability since the unsuccessful move still drains the battery, excluding also the alternative route via  $B$ .

We diverge slightly from literature in that we do not generally constrain policies to be closed, i.e., they may decide to stop at arbitrary states instead of only at terminal or goal states. If closed policies are desired,  $\pi(s) = \perp$  at a non-terminal state  $s$  can be interpreted as “act arbitrarily”. As for such states  $V^\pi(s) = 0$  by definition, augmenting policies in this manner cannot possibly decrease the expected goal probability, i.e.,  $V^\pi$  lower bounds the goal probabilities of every closed policy that agrees with  $\pi$  on all states  $s$  where  $\pi(s)$  is defined.

The piecewise minimality requirement is necessary to account for 0-reward (non-goal) cycles. For example, consider the policy  $\pi(s_0) = \pi(s_1) = a$  for the MDP in Figure 10.1. The equation characterizing  $V^\pi$  is satisfied by any  $V$  iff  $V(s_*) = 1$  and  $V(s_0) = V(s_1)$ , e.g.,  $V(s_0) = V(s_1) = V(s_*) = 1$ . The equation alone does hence not suffice to uniquely identify  $V^\pi$ . Yet, among all value functions satisfying the equation,  $V^\pi(s_*) = 1$  and  $V^\pi(s_0) = V^\pi(s_1) = 0$  is the only one that is piecewise minimal.

Finally, given the goal probabilities achieved by individual policies, we can formally characterize *optimality*:

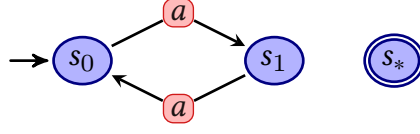


Figure 10.1.: Example MDP with three states, and (deterministic) transitions as depicted.  $s_*$  is the only goal state.  $a$  induces a 0-reward cycle between  $s_0$  and  $s_1$ .

**Definition 10.13 (MaxProb).** *The **optimal goal-probability function**, also called **maximal goal probabilities (MaxProb)** or simply **optimal value function**, is the function  $V^*: \mathcal{S} \rightarrow [0, 1]$  such that, for all states  $s \in \mathcal{S}$ :*

$$V^*(s) = \max_{\text{policy } \pi} V^\pi(s)$$

A policy  $\pi$  is **optimal** in  $s \in \mathcal{S}$  if  $V^\pi(s) = V^*(s)$ . An optimal policy for  $\mathcal{M}$  is a policy optimal in  $s_I$ .

## 10.7. Goal-Probability Objectives & Complexity

Having defined policies and their associated goal-probability measure, it is now time to formulate the algorithmic questions addressed in the remainder of this part of thesis. We will consider three goal-probability objectives:

**MaxProb:** Given a (budget-limited) probabilistic FDR task  $\Pi$ . Find an optimal policy  $\pi^*$  for  $\mathcal{M}^\Pi$ , i.e.,  $\pi^*$  such that  $V^{\pi^*}(s_I) = V^*(s_I)$ .

**AtLeastProb:** Given  $\Pi$  and a probability threshold  $\theta \in [0, 1]$ . Find a policy  $\pi$  for  $\mathcal{M}^\Pi$  such that  $V^\pi(s_I) \geq \theta$ , or prove that no such policy exists.

**ApproxProb:** Given  $\Pi$  and an error bound  $\delta \in [0, 1]$ . Find a policy  $\pi$  for  $\mathcal{M}^\Pi$  such that  $V^*(s_I) - V^\pi(s_I) \leq \delta$ .

The first question asks for a provably optimal solution akin to optimal classical planning. Optimal planning being notoriously difficult, the other two questions consider weaker objectives, aiming at sub-optimal policies that meet user defined quality constraints. The sub-optimality parameter controls the difficulty of finding a desired policy, ranging from very trivial (e.g.,  $\theta = 0$ ) to as expensive as optimal planning (e.g.,  $\delta = 0$ ). We will evaluate the impact of this parameter in detail in our experiments. In terms of the worst case complexity, however, all three questions are indistinguishable. In a decision theoretic formulation, the questions boil down to

**Probabilistic Plan-Existence:** Given  $\Pi$  and  $\theta \in [0, 1]$ . Does there exist a policy  $\pi$  for  $\mathcal{M}^\Pi$  such that  $V^\pi(s_I) \geq \theta$ ?

Probabilistic plan-existence can be decided in polynomial time in the size of the *flat* MDP representation  $\mathcal{M}^\Pi$  (Papadimitriou and Tsitsiklis, 1987). However, as we have already observed in the classical setting, there is a significant size discrepancy between syntactic  $\Pi$  and semantic  $\mathcal{M}^\Pi$  representation: the state explosion problem. With respect to the size of  $\Pi$ , the result by Papadimitriou and Tsitsiklis merely shows that probabilistic plan-existence can be decided in exponential time in the input size. Unfortunately, it turns out that this is actually the best one can do in the worst case. As has been shown by Littman (1997), probabilistic plan-existence for factored MDPs, as we consider here, is an EXPTIME-complete decision problem.

# 11. Fundamental Algorithms

There are three default algorithms when it comes to solving MDPs optimally: via linear programming; *value iteration*; and *policy iteration*. This chapter revisits variants of the LP-based method and value iteration for goal-probability analysis. Both approaches require and operate on an explicit representation of the entire (reachable) state space  $\mathcal{M}^\Pi$ . They address MaxProb specifically. While the weaker objectives are solved as a byproduct, the methods do not benefit from the possibly less strict solution requirements. The more advanced techniques presented in the next chapter build upon the two baseline methods, addressing those deficiencies.

This chapter is to a large extent based on standard text book results (Puterman, 1994; Bertsekas, 1995). We try to offer a comprehensive overview of central notions and the theoretical characteristics, underlying optimally solving MDPs. As such, this chapter also lays down the basics for upcoming chapters. Given that goal-probability analysis is considered only tangentially in the aforementioned text books, it can sometimes be difficult to crystallize the principal arguments of why certain properties hold true in this context. To provide the reader with a fundamental understanding of the most important concepts, we insert detailed proofs, specifically targeting MaxProb analysis, where we see a need. Moreover, we introduce a variant of Dai et al.'s (2011) topological VI algorithm, adapted to the MaxProb objective, and show how an optimal policy can be extracted as a post-process.

## 11.1. Linear Program Representation

Linear programming offers a *direct* method to solve MDPs. The MDP's dynamics and optimization objective can be naturally formulated as an LP (Puterman, 1994):

**Definition 11.1** (Primal Goal-Probability LP). *Let  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  be an MDP. The primal goal-probability LP is:*

$$\underset{v}{\text{minimize}} \quad \sum_{s \in \mathcal{S}} v_s \quad (11.1a)$$

$$\text{subject to} \quad v_s \geq 0 \quad s \in \mathcal{S}, \quad (11.1b)$$

$$v_{s_*} = 1 \quad s_* \in \mathcal{S}_*, \quad (11.1c)$$

$$v_s \geq \sum_{t \in \mathcal{S}} \mathcal{P}(s, a, t) v_t \quad s \in (\mathcal{S} \setminus \mathcal{S}_*), a \in \mathcal{A}(s) \quad (11.1d)$$

The LP has one variable  $v_s$  for every state  $s \in \mathcal{S}$ , which represents the state's  $V^*(s)$  value. (11.1b) enforces proper probability bounds. As per the case distinction in Definition 10.12, the other constraints of the LP assert that: (11.1c) the goal-probability value of every goal state must be 1; and (11.1d) the goal-probability value attached to non-goal states must not be lower than the expected goal probability of any of its outgoing transitions. The point-wise minimality requirement is represented via the objective function (11.1a).

**Theorem 11.1** (Puterman, 1994). *The primal goal-probability LP has a unique optimal solution  $v^*$ , for which  $v_s^* = V^*(s)$  holds for all states  $s \in \mathcal{S}$ .*

This LP encoding however has some obvious downsides. In particular, it requires an explicit representation of the *entire* MDP  $\mathcal{M}$ , with all its states and transitions. As discussed previously, this is not tractable for  $\mathcal{M}$  implicitly described by probabilistic FDR tasks, or even not possible at all in the limited-budget case. To remedy this issue, at least to some extent, note that it is actually not necessary to compute the  $V^*$  values for *all* states. In particular, it suffices to construct the LP for the typically much smaller reachable subgraph  $\mathcal{M}|_{\mathcal{R}(s_I)}$ . This restricted LP still guarantees that the optimal solution satisfies  $v_s^* = V^*(s)$  for all reachable states  $s \in \mathcal{S}|_{\mathcal{R}(s_I)}$ , since trivially their goal-probability values in  $\mathcal{M}|_{\mathcal{R}(s_I)}$  and in  $\mathcal{M}$  are the same.

The dual of the goal-probability LP will become handy later on. For the sake of brevity, we define it directly on the reachable subgraph:

**Definition 11.2** (Dual Goal-Probability LP). *Suppose  $s_I \notin \mathcal{S}_*$ . The dual goal-probability LP over the reachable subgraph  $\mathcal{M}|_{\mathcal{R}(s_I)}$  is*

$$\begin{aligned} & \underset{om}{\text{maximize}} && \sum_{s_* \in \mathcal{S}_* | \mathcal{R}(s_I)} in(s_*) \end{aligned} \quad (11.2a)$$

$$\text{subject to} \quad om_{s,a} \geq 0 \quad s \in (\mathcal{R}(s_I) \setminus \mathcal{S}_*), a \in \mathcal{A}(s), \quad (11.2b)$$

$$out(s) - in(s) \leq \mathbb{1}[s = s_I] \quad s \in (\mathcal{R}(s_I) \setminus \mathcal{S}_*) \quad (11.2c)$$

using the following shorthand:

$$\begin{aligned} out(s) &= \sum_{a \in \mathcal{A}(s)} om_{s,a} \\ in(s) &= \sum_{t \in (\mathcal{R}(s_I) \setminus \mathcal{S}_*)} \sum_{a \in \mathcal{A}(t)} \mathcal{P}(t, a, s) om_{t,a} \end{aligned}$$

$\mathbb{1}[\phi]$  denotes the **Iverson bracket** (Iverson, 1962), i.e.,  $\mathbb{1}[\phi] = 1$  if  $\phi$  is true, and  $\mathbb{1}[\phi] = 0$  otherwise. If  $s_I$  is a goal state, then trivially  $V^*(s_I) = 1$  and the empty policy is an optimal one. We exclude this case from the LP definition for simplicity. The variables of the dual LP are commonly known as the **occupation measures** (Altman, 1996). Intuitively,  $om_{s,a}$  represents the expected number of times  $a$  is executed in  $s$ . The constraint (11.2c) ensures that no state can be exited more often than entering it. The constraint bound for the initial state represents the probability mass that is fed into the MDP. The objective function (11.2a) demands maximizing the probability mass that reaches the goal states. The objective value of the optimal solutions is exactly the maximal goal probability of the initial state.

While linear programming offers a very natural way to solve MDPs, it is seldom used in practice. Since the LP-encoding encompasses a description of the entire (reachable) MDP, facing millions or even billions of states in common benchmarks, the size is typically way beyond what off-the-shelf LP solvers can handle efficiently. The iterative numeric procedures presented next tend to work better on such large inputs. In the chapter hereafter, we will revisit a recently proposed variant that adopts ideas from heuristic search to further restrict the subgraph  $\mathcal{M}|_{\mathcal{R}(s_I)}$  represented in the LP, without losing the optimality property.

## 11.2. Value Iteration

Value iteration (short VI) computes the optimal value function of MDPs by solving the associated system of *Bellman equations* (Bellman, 1957) through an iterative numeric procedure. An optimal policy can be read

off in a post-process. Each iteration computes a new value function  $V^{(k)}$  by applying so called *Bellman updates* to the value function  $V^{(k-1)}$  of the preceding iteration. The update operation guarantees that the difference between the value functions gets gradually smaller, thus converging to a fixed point  $V^{(\infty)}$ . This fixed point is a solution to the Bellman equations, yielding  $V^{(\infty)} = V^*$ .

This section is structured as follows. In Section 11.2.1, we revisit the core operation underlying VI – the Bellman update operator – in the context of goal-probability analysis. Moreover, we introduce and prove the theoretical properties of the update operator that gives rise to the computation of  $V^*$  via algorithms like VI. Some of these properties will also play an important role to show correctness of the advanced algorithms presented in the upcoming chapter. In Section 11.2.2, we introduce the termination condition, which will also be adopted by the algorithms in the next chapter. Section 11.2.3 covers topological VI (Dai et al., 2011), an efficient VI variant. Finally, Section 11.2.4 explains how a policy can be extracted from the value function, once computed by VI.

For the following discussion, let  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  be an arbitrary MDP.

### 11.2.1. Principles

The Bellman update operator takes a value function as input and produces a new value function by pretending to act optimally according the provided value estimates:

**Definition 11.3** (Q-Value). *Let  $V: \mathcal{S} \rightarrow [0, 1]$  be a value function. Suppose  $s \in (\mathcal{S} \setminus \mathcal{S}_*)$  and  $a \in \mathcal{A}(s)$ . The **Q-value** of  $s$  and  $a$  in  $V$  is*

$$(QV)(s, a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') V(s')$$

**Definition 11.4** (Bellman Update Operator). *Let  $V: \mathcal{S} \rightarrow [0, 1]$  be a value function. Suppose  $s \in \mathcal{S}$ . The **Bellman update** (sometimes called Bellman backup, or value update) of  $V$  at  $s$  is*

$$(BV)(s) = \begin{cases} 1 & \text{if } s \in \mathcal{S}_* \\ 0 & \text{if } s \notin \mathcal{S}_* \text{ and } \mathcal{A}(s) = \emptyset \\ \max_{a \in \mathcal{A}(s)} (QV)(s, a) & \text{otherwise} \end{cases}$$

In words, the  $Q$ -value  $(QV)(s, a)$  is the expected value when executing  $a$  in  $s$ , in accordance to the prior goal-probability estimates  $V$ . For goal and terminal states, the Bellman update operator simply returns the exact (known) goal probability. For other states,  $(BV)(s)$  follows the best action as per the expected  $Q$ -values in  $V$ .

We use  $BV$  as a shorthand to denote the value function resulting from synchronously applying Bellman updates to  $V$  and all states  $s \in \mathcal{S}$ . Moreover, given a policy  $\pi$ , it will be convenient to write  $B^\pi$  for denoting the Bellman update operator restricted to the policy graph  $\mathcal{M}^\pi$ , i.e.,

$$(B^\pi V)(s) = \begin{cases} 1 & \text{if } s \in \mathcal{S}_* \\ 0 & \text{if } s \notin \mathcal{S}_* \text{ and } \pi(s) = \perp \\ (QV)(s, \pi(s)) & \text{otherwise} \end{cases}$$

The Bellman update operator offers two properties that enable the computation of  $V^*$  via VI and its derivatives:

- (I) considering an infinite sequence of value functions obtained by repeatedly applying the update operator to the result of itself, this sequence converges in the limit; and
- (II) Bellman's characterization of the optimal value function as means of such a fixed point.

In finite-horizon and infinite-horizon discounted MDPs, (I) follows immediately from the syntactic MDP definition. As a matter of fact, both types of MDPs can be seen as special cases of SSPs (Bertsekas and Tsitsiklis, 1996), for which the Bellman update operator was shown to converge to a unique fixed point, regardless of the initial value function (Bertsekas, 1995). For goal-probability MDPs in their full generality, matters turn out to be more difficult. For the purpose of this thesis, it suffices to establish property (I), if starting from value functions of the following kind:

**Definition 11.5** (Monotone Value Function). *Let  $V$  be a value function.  $V$  is a **monotone upper bound** of  $V^*$  (or simply **monotone upper bound**) if  $V \geq V^*$  and  $V \geq BV$ .  $V$  is a **monotone lower bound** of  $V^*$  (or simply **monotone lower bound**) if  $V \leq V^*$  and  $V \leq BV$ .*

Monotonicity can be seen as a generalization of the consistency property of heuristics in the classical planning setting (cf. Definition 2.7). It requires the value bounds to be *consistent* according to the probabilistic transition function. In the case of upper bounds,  $V(s)$  must not be smaller than the expected value under any of the state's outgoing transitions. Vice versa, for lower bounds, it must hold that  $V(s) \leq \sum_{s'} \mathcal{P}(s, a, s')V(s')$  for each action applicable in  $s$ . In particular, notice that for deterministic transitions, the latter inequality boils down to exactly the definition of consistent classical-planning heuristics, modulo the cost function. The monotonicity property is invariant under the Bellman update operator:

**Theorem 11.2** (Monotonicity Invariance). *Let  $V$  be a value function. Suppose  $s \in \mathcal{S}$ , and let*

$$V'(t) = \begin{cases} (BV)(s) & \text{if } t = s \\ V(t) & \text{otherwise} \end{cases}$$

*If  $V$  is a monotone lower bound (upper bound), then so is  $V'$ .*

*Proof.* Suppose  $V$  is a monotone lower bound. The proof for upper bounds is symmetric. It holds for all states  $t \neq s$  that  $V'(t) = V(t)$ , and  $V(t) \leq V^*(t)$  is true by assumption, i.e.,  $V'(t) \leq V^*(t)$ . To obtain  $V' \leq V^*$ , it remains to show that  $V'(s) \leq V^*(s)$ . If  $s$  is a goal or terminal state, then  $V'(s) = V^*(s)$  holds by definition. Otherwise

$$V'(s) = (BV)(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}(s, a, s')V(s') \leq \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}(s, a, s')V^*(s') = V^*(s)$$

To show that  $V'$  is monotone, let  $t$  be any state. If  $t$  is a goal or terminal state, then  $(BV')(t) = V^*(t)$ , and  $V^*(t) \geq V'(t)$ , as we have just shown. Suppose  $t$  is neither terminal, nor a goal state. The monotonicity property of  $V$  yields the following chain of inequalities:

$$(BV')(t) = \max_{a \in \mathcal{A}(t)} \sum_{t'} \mathcal{P}(t, a, t')V'(t') \geq \max_{a \in \mathcal{A}(t)} \sum_{t'} \mathcal{P}(t, a, t')V(t') = (BV)(t) \geq V'(t)$$

□

The claim can be straightforwardly extended to the synchronous application of  $B$  to all states. Provided that the initial value function is monotone and bounded, property (I) follows via standard results from the fixed-point theory:

**Theorem 11.3** (Convergence). *Let  $V^{(0)}$  be a monotone (lower or upper) bound. Let  $V^{(1)}, V^{(2)}, \dots$  be defined such that  $V^{(i)} = \mathbf{B}V^{(i-1)}$  for all  $i \geq 1$ . There exists a value function  $V^{(\infty)}$  such that*

$$\lim_{n \rightarrow \infty} V^{(n)} = V^{(\infty)}$$

*Proof.* Suppose that  $V^{(0)}$  is a monotone lower bound. The argumentation for upper bounds is symmetric. It follows from Theorem 11.2 that  $V^{(0)} \leq V^{(1)} \leq \dots \leq V^*$ . Consider the (possibly infinite) totally ordered set  $C = \{V^{(0)}, V^{(1)}, \dots\}$ . Notice that each non-empty (possibly infinite) subset of  $C$  has a supremum (bounded above by  $V^*$ ) and an infimum (bounded below by  $V^{(0)}$ ). Hence,  $C$  constitutes a complete lattice. Let  $i < j$  be arbitrary, i.e.,  $V^{(i)} \leq V^{(j)}$ . Therefore,  $\mathbf{B}V^{(i)} = V^{(i+1)} \leq V^{(j)} \leq \mathbf{B}V^{(j)}$ . In other words,  $\mathbf{B}$  is an order-preserving function  $\mathbf{B}: C \rightarrow C$ . It follows from the Knaster-Tarski fixed-point theorem (Tarski, 1955) that  $\mathbf{B}$  must have a fixed point  $V^{(\infty)} \in C$ .  $\square$

The computation of  $V^*$  can be phrased as finding such a fixed point  $V^{(\infty)}$ , as per property (II):

**Theorem 11.4** (Bellman Equations for Goal Probability). *The optimal goal-probability function satisfies*

$$V^* = \mathbf{B}V^*$$

or equivalently, for all states  $s \in \mathcal{S}$

$$V^*(s) = \begin{cases} 1 & \text{if } s \in \mathcal{S}_* \\ 0 & \text{if } s \notin \mathcal{S}_* \text{ and } \mathcal{A}(s) = \emptyset \\ \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s') V^*(s') & \text{otherwise} \end{cases}$$

For SSPs,  $V^*$  is the unique solution to the Bellman equations (Bertsekas, 1995). In this case, finding just any fixed point  $V^{(\infty)}$  of  $\mathbf{B}$  suffices to establish the desired connection  $V^{(\infty)} = V^*$ . Unfortunately, this does not need to be the case for goal-probability MDPs in general, as the example in Figure 10.1 showed. Since  $V^*$  is however by definition the piecewise smallest value function satisfying the Bellman equations, we are guaranteed to obtain  $V^{(\infty)} = V^*$  if  $V^{(0)}$  is a monotone lower bound. Namely, consider the set  $C$  from the proof of Theorem 11.3. As per Theorem 11.2, it holds for all  $V^{(i)} \in C$  that  $V^{(i)} \leq V^*$ , so in particular,  $V^{(\infty)} \leq V^*$  for the fixed point  $V^{(\infty)} \in C$ . Together with  $V^{(\infty)} = \mathbf{B}V^{(\infty)}$  and the prior observation, this yields  $V^{(\infty)} = V^*$ .

In summary, this condenses into the following simple algorithm: initialize  $V^{(0)}(s) = 0$  for all states  $s$ ; for  $i = 1, 2, \dots$  compute  $V^{(i)} = \mathbf{B}V^{(i-1)}$ . Since  $V^{(0)}$  is trivially a monotone lower bound, the sequence of value functions converges to the optimal goal probabilities  $V^*$  as per the previous observation. Since in this method, the value of every state gets updated in each iteration, it is often referred to as *synchronous VI*. In contrast, *asynchronous VI* updates the value function only at a single state per iteration. The convergence guarantee is not affected so long as every state is updated infinitely often during the course of all iterations.

### 11.2.2. Termination

An issue that we glanced over so far is the convergence test, i.e., at which point  $n$  can the computation of the sequence  $V^{(0)}, V^{(1)}, \dots, V^{(n)}$  be terminated? The most intrusive thought is to terminate as soon as it holds  $V^{(n)} = V^{(n-1)}$ , i.e., once the fixed point has been reached. Unfortunately, this is not possible in general. Below, we give an example where the condition is not satisfied after any finite step  $n$ . The process

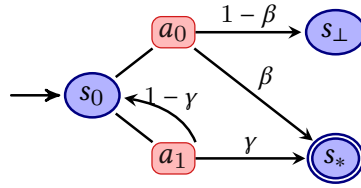


Figure 11.1.: MDP with three states, and two probabilistic transitions.  $s_*$  is a goal state;  $s_\perp$  is a dead end. The probabilities are chosen s.t.  $\gamma < \beta < \epsilon$ .

must hence be terminated before the *exact* fixed point is reached, i.e., once the difference  $V^* - V^{(n)}$  has become sufficiently small. Since  $V^*$  is however not known, this difference cannot be computed directly. The most commonly deployed approximation is via the maximal value change going into the next iteration:

**Definition 11.6** (Bellman Residual). *Let  $V$  be a value function. Suppose  $s \in \mathcal{S}$ . The **Bellman residual** of  $s$  in  $V$  is*

$$(\Delta V)(s) = |(BV)(s) - V(s)|$$

*The Bellman residual of  $V$  is*

$$\Delta V = \max_{s \in \mathcal{S}} (\Delta V)(s)$$

**Definition 11.7** ( $\epsilon$ -Consistency). *Let  $\epsilon \geq 0$  be a threshold. Let  $V$  be a value function. Suppose  $s \in \mathcal{S}$ . Then  $s$  is called  **$\epsilon$ -consistent** in  $V$  if  $(\Delta V)(s) \leq \epsilon$ .  $V$  is called  **$\epsilon$ -consistent** if  $\Delta V \leq \epsilon$ .*

VI is run until  $V^{(n)}$  becomes  $\epsilon$ -consistent, for some  $\epsilon > 0$ . Starting from a monotone value function, the Bellman residuals monotonically decrease along the sequence of computed value functions, i.e., one can show that  $\Delta V^{(i)} \geq \Delta V^{(j)}$  for  $j \geq i$ . The proof is provided in Appendix C.1.1. In other words, once the residual has become small, the effects of subsequent Bellman updates become more and more negligible. At this point, one can therefore expect that  $V^{(n)}$  is already very close to  $V^*$ . For sufficiently small  $\epsilon$ , this usually works out well. Nevertheless, note that, from a theoretical point of view,  $\epsilon$ -consistency does per se not warrant any bound on the actual difference to the optimal value function. In fact, no matter how small  $\epsilon$  is chosen, one can easily come up with an artificial example where  $\Delta V^{(n)} \leq \epsilon$ , yet  $V^{(n)}$  is still arbitrarily far off from  $V^*$ . Example 11.1 shows an extreme example.

**Example 11.1.** *Consider the MDP depicted in Figure 11.1. Table 11.1 shows the sequence of value functions  $V^{(0)}, V^{(1)}, \dots$  where  $V^{(0)}$  is initialized to 0 everywhere, and  $V^{(i)} = BV^{(i-1)}$  for  $i > 0$ .  $V^{(1)}$  is  $\epsilon$ -consistent since its Bellman residual is  $\beta < \epsilon$ . However,  $V^{(1)}(s_0) = 0$  although  $V^*(s_0) = 1$ .*

Under certain circumstances, one can infer a bound on the difference to the optimal value function directly from the Bellman residual. For instance, for infinite-horizon discounted MDPs, the error bound can be expressed directly as a combination of the Bellman residual and the discount factor. Also for SSPs, it is possible to measure the error bound, yet there no longer exists a simple closed-form characterization (Bertsekas, 1995). There however exist closed-form over-approximations, trading accuracy for computational tractability (Hansen, 2017). Such approximations are also available for goal-probability objectives (Quatmann and Katoen, 2018). Given a formula that characterizes (an approximation of) the error bound based on the Bellman residual, one can vice versa calculate an appropriate value of  $\epsilon$  that guarantees to meet a desired error bound. In the remainder of the thesis, we assume that an  $\epsilon$  parameter is given that yields a sufficiently precise solution, and focus on finding an  $\epsilon$ -consistent value function.

$k$	$V^{(k)}(s_0)$	$V^{(k)}(s_\perp)$	$V^{(k)}(s_*)$	$\Delta V^{(k)}$
0	0	0	0	1
1	0	0	1	$\beta$
$\Delta V^{(1)} = \beta < \epsilon \Rightarrow \epsilon\text{-consistent}$				
2	$\beta$	0	1	$(1-\beta)\gamma$
3	$(1-\gamma)\beta + \gamma$	0	1	$(1-\gamma)(1-\beta)\gamma$
4	$(1-\gamma)^2\beta$ $+ (1-\gamma)\gamma + \gamma$	0	1	$(1-\gamma)^2(1-\beta)\gamma$
$\vdots$				
$n$	$(1-\gamma)^{n-2}\beta$ $+ \sum_{i=0}^{n-3} (1-\gamma)^i \gamma$	0	1	$(1-\gamma)^{n-2}(1-\beta)\gamma$
$\vdots$				
$\infty$	1	0	1	0

Table 11.1.: Value functions computed by VI for the MDP from Figure 11.1, along with their Bellman residuals.

### 11.2.3. Topological Value Iteration

Updating the values of all states in every iteration is problematic in two respects. The complete enumeration of all states of FDR tasks, with or without budget-limit, is intractable. Secondly, many updates are redundant, making the overall approach very inefficient. *Topological VI* (short TVI) addresses both issues by exploiting the graph structure of the MDP (Dai et al., 2011). To reduce the number of considered states, TVI restricts its focus on the reachable subgraph. To eliminate redundant updates, TVI makes use of the *topological order* induced by the MDP's successor relation. In regards of the latter, observe that the Bellman update of a state can change only if the values of its successors have changed beforehand. Vice versa, the result of the Bellman update may keep changing as long as the values of the state's successors are changing.

Algorithm 11.1 shows our TVI variant, reflecting the aforementioned observations. While it assumes an MDP  $\mathcal{M}$  as input,  $\mathcal{M}$  does not need to be represented *explicitly*. The reachable subgraph is constructed in a pre-process. The value function is initialized to 0 for all reachable states. TVI computes all maximal SCCs of the reachable subgraph via Tarjan's algorithm (see Chapter 3). The SCCs cover exactly the mutual dependencies between the states' values. VI is applied to each SCC individually. The SCC computation guarantees that child SCCs are listed before their parents. TVI processes the SCCs following this order to ensure that the Bellman updates are performed bottom up, successor states before parents, thus propagating converged values through the MDP in inverse topological order. For states not appearing in any cycle, a single update suffices. If  $\mathcal{M}$  is acyclic, then TVI performs exactly one update per state, and the computed values are guaranteed to be exact.

**Theorem 11.5** (Dai et al., 2011). *For every MDP  $\mathcal{M}$ , and threshold  $\epsilon > 0$ , TVI terminates eventually, and the resulting value function  $V$  is  $\epsilon$ -consistent at all reachable states. If  $\mathcal{M}$  is acyclic, then  $V(s) = V^*(s)$  holds for all states  $s \in \mathcal{R}(s_I)$ .*

Dai et al. (2011) also introduce *focused TVI*, which eliminates sub-optimal transitions in a pre-process to obtain smaller SCCs. While this can be much more runtime-effective, it still requires building the entire state space. In our experiments, runtime/memory exhaustion during this process, i.e., during building the state space, was the only reason for VI failures. So we do not consider this variant here.

**Algorithm 11.1:** Topological VI for MaxProb analysis.**Input:**  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle, \epsilon \in (0, 1)$ **Output:**  $V: \mathcal{R}(s_I) \rightarrow [0, 1]$ 

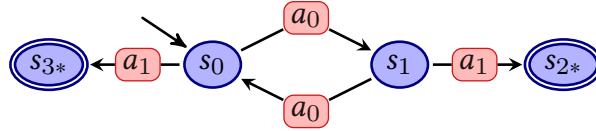

---

```

1 Build  $\mathcal{M}|_{\mathcal{R}(s_I)}$ ;
2  $S_1, S_2, \dots, S_n \leftarrow$  Maximal SCCs of  $\mathcal{M}|_{\mathcal{R}(s_I)}$  in inverse topological order ;      /* (Alg. 3.1) */
   /* Initialize value function */
3  $V \leftarrow$  empty map; foreach  $s \in \mathcal{R}(s_I)$  do  $V(s) \leftarrow 0$ ;
   /* Process SCCs, children before parents */
4 for  $i \leftarrow 1$  to  $n$  do
5   if  $S_i = \{s\}$  and  $s$  has no self-loop then
6     /* Single update suffices */
7      $V(s) \leftarrow (BV)(s)$ 
8   else
9     /* Apply VI to just the states in the current component until
10      reaching  $\epsilon$ -consistency */
11     repeat
12        $error \leftarrow 0$ ;
13       foreach  $s \in S_i$  do
14          $error \leftarrow \max\{error, (\Delta V)(s)\}$ ;
15          $V(s) \leftarrow (BV)(s)$ ;
16     until  $error < \epsilon$ ;
17 return  $V$ ;

```

---

Figure 11.2.: Example MDP with four states, and (deterministic) transitions as depicted.  $s_{2*}$  and  $s_{3*}$  are goal states.**11.2.4. Policy Extraction**

Once  $V^*$  (or a sufficiently precise approximation thereof) has been found, one can obtain an optimal policy  $\pi^*$  by acting *greedily* on the computed value function:

**Definition 11.8** (Greedy Policy). *Let  $V$  be a value function in  $\mathcal{M}$ . Suppose  $s \in (\mathcal{S} \setminus \mathcal{S}_*)$ , and  $S \subseteq \mathcal{S}$ . An action  $a \in \mathcal{A}(s)$  is **greedy on  $V$  for  $s$**  if it holds that  $(QV)(s, a) \geq (QV)(s, a')$  for all  $a' \in \mathcal{A}(s)$ . A policy  $\pi$  is **greedy on  $V$  for  $S$**  if, for all  $s \in (S \setminus \mathcal{S}_\perp \setminus \mathcal{S}_*)$ ,  $\pi(s)$  is greedy on  $V$  for  $s$ .  $\pi$  is **greedy on  $V$**  if  $\pi$  is greedy on  $V$  for  $(S \setminus \mathcal{S}_\perp)$ .*

In other words, an action is greedy on  $V$  for  $s$ , if it achieves the maximal expected value according to  $V$  under all applicable actions for  $s$ . A policy is greedy on  $V$ , if, whenever  $\pi(s) \neq \perp$ ,  $\pi(s)$  is greedy on  $V$  for  $s$ . A policy greedy on  $V$  that is also closed, needs to assign an action greedy on  $V$  to every non-terminal, non-goal state from  $\mathcal{R}^\pi(s_I)$ . Note that every optimal policy is necessarily greedy on  $V^*$ . Unfortunately, however, the opposite is not true in general, i.e., there can exist closed policies greedy on  $V^*$  that are not optimal. Example 11.2 shows such an example.

---

**Algorithm 11.2:** MaxProb policy extraction from a given value function.
 

---

**Input:**  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle, V: \mathcal{S} \rightarrow [0, 1]$ 
**Output:** Policy  $\pi_V: (\mathcal{S} \setminus \mathcal{S}_*) \rightarrow \mathcal{A}$ 

```

1  $\pi_V \leftarrow$  empty policy;
  /* start from goal states */
2  $processed \leftarrow \mathcal{S}_*$ ;
3 while  $processed \neq \emptyset$  do
4    $s \leftarrow$  pick some state from  $processed$ ;
5    $processed \leftarrow processed \setminus \{s\}$ ;
  /* check for unprocessed predecessors */
6   foreach  $s' \in Pred[\mathcal{M}](s)$  s.t.  $s' \notin \mathcal{S}_*$  and  $\pi_V(s') = \perp$  do
7     /* extend the policy if  $s'$  has a greedy action that transitions into  $s$  */
8     if  $\exists a \in \mathcal{A}(s'): \mathcal{P}(s', a, s) > 0 \wedge a$  is greedy on  $V$  for  $s'$  then
9        $\pi_V(s') \leftarrow a$ ;
      /* repeat for predecessors of  $s'$  */
       $processed \leftarrow processed \cup \{s'\}$ ;
10 return  $\pi_V$ ;

```

---

**Example 11.2.** Consider the MDP from Figure 11.2. All states  $s$  have an optimal goal-probability value of  $V^*(s) = 1$ . The  $a_0$ -transitions for  $s_0$  and  $s_1$  have maximal expected value. However, the policy  $\pi(s_0) = \pi(s_1) = a_0$  is not optimal, since it never reaches the goal.

Arbitrarily selecting actions greedy on the value function may not yield an optimal policy due to 0-reward cycles. To avoid the pitfall of 0-reward cycles by construction, we follow an idea by Kolobov et al. (2011), and build the policy backwards starting from the goal states. Algorithm 11.2 shows the procedure. The algorithm ensures that the execution of the policy from any  $s$  with  $\pi_V(s) \neq \perp$  is guaranteed to reach the goal with a non-zero probability. The set  $processed$  keeps track of the states for which this has already been satisfied. The policy  $\pi_V$  is incrementally extended by looking for transitions  $\langle s, a, s' \rangle$  from any state  $s$  where  $\pi_V(s) = \perp$  into some state  $s' \in processed$ , via an action  $a$  that is greedy on  $V$  for  $s$ . When such a transition is found,  $\pi_V(s)$  is set to  $a$ , and  $s$  is added to  $processed$ . The process is continued until the policy cannot be extended any further. The resulting policy  $\pi_V$  is guaranteed to perform no worse than  $V$ , i.e.,  $V^{\pi_V} \geq V$ , provided that  $V$  supports the construction:

**Theorem 11.6.** Let  $V$  be a monotone value function. Algorithm 11.2 on  $V$  terminates. Suppose  $\pi_V$  is the resulting policy. If there exists a policy  $\pi$  with  $V^\pi \geq V$  that is greedy on  $V$ , then  $V^{\pi_V} \geq V$ .

*Proof sketch.* Algorithm 11.2 terminates eventually since each state can be processed at most once, and the number of states is finite. Since, by assumption, there exists a policy  $\pi$  greedy on  $V$  with  $V^\pi \geq V$ , the exhaustive backpropagation done by Algorithm 11.2 guarantees that  $\pi_V(s)$  is defined for all non-goal states with  $V(s) > 0$ . By construction, executing  $\pi_V$  from any state  $s$  with  $\pi_V(s) \neq \perp$  will eventually end in either a goal state, or a state where  $\pi_V$  is not defined. Hence,  $\pi_V$  induces an SSP  $\mathcal{M}^{\pi_V}$ . Therefore,  $V^{\pi_V}$  is the unique fixed point of  $\mathbf{B}^{\pi_V}$ . This means particularly that the sequence  $V^{(0)}, V^{(1)}, \dots$ , where  $V^{(0)} = V$ , and  $V^{(i)} = \mathbf{B}^{\pi_V} V^{(i-1)}$ , converges to  $V^{\pi_V}$ . Since  $V$  is monotone, it holds that  $V \leq \mathbf{B} V$ . So, by selection of  $\pi_V$ ,  $V \leq \mathbf{B}^{\pi_V} V$ , i.e.,  $V^{(0)} \leq V^{(1)}$ . Since  $\mathbf{B}^{\pi_V}$  is a monotone function (cf. Theorem 11.2), it follows that

$$V = V^{(0)} \leq V^{(\infty)} = V^{\pi_V}.$$

□

The proof arguments are spelled out in Appendix C.1.2. Notice that it does not suffice that  $V$  is a monotone lower bound in order to guarantee the existence of a policy  $\pi$  that is greedy on  $V$  and satisfies  $V^\pi \geq V$ . For example, consider the value function  $V(s_0) = V(s_1) = 1$  and  $V(s_{2*}) = V(s_{3*}) = 0$  for the MDP from Figure 11.2. This value function is a monotone lower bound on  $V^*$ . Yet, the only closed policy greedy on  $V$  is  $\pi$  from Example 11.2, for which it holds  $V^\pi(s_0) = V^\pi(s_1) = 0$ .

Nevertheless, VI guarantees the existence of such a greedy policy at all times. When starting from the trivial monotone lower bound 0, the source of every Bellman-update value increase must necessarily be a goal state. Therefore, the actions responsible for a value increase must start some goal path. By recurring the same argument, one can hence show that each non-goal state with a non-zero goal-probability value has a path to a goal state composed of only greedy actions. In combination, these paths yield the requested policy. We formalize this intuition as two separate lemmas:

**Lemma 11.1.** *Suppose  $V^{(0)}(s) = 0$  for all states  $s \in \mathcal{S}$ , and, for  $i > 0$ ,  $V^{(i)} = \mathbf{B}V^{(i-1)}$ . At any point  $i \geq 0$ , and for every non-goal state  $s_0 \in \mathcal{S} \setminus \mathcal{S}_*$  with  $V^{(i)}(s_0) > 0$ , there exists a path  $s_0, a_0, s_1, a_1, \dots, s_n$  such that*

- (i)  $s_n \in \mathcal{S}_*$ , and
- (ii)  $\forall 0 \leq j < n$ ,  $a_j$  is greedy on  $V^{(i)}$  for  $s_j$ , and
- (iii)  $\forall 0 \leq j < n$ ,  $V^{(i)}(s_j) \leq V^{(i)}(s_{j+1})$ .

*Proof sketch.* The proof is by induction on  $i$ . The induction beginning,  $i = 0$  and  $i = 1$ , holds trivially. Suppose as the induction hypothesis (IH1), that the claim holds for  $i$ . To show the induction step, we use a second induction, now on the  $V^{(i+1)}$  values (on the sequence of states ordered by decreasing  $V^{(i+1)}$  values). For the induction beginning, let  $s$  be any non-goal state with maximal  $V^{(i+1)}(s) > 0$  value. Let  $a \in \mathcal{A}(s)$  be greedy on  $V^{(i)}$  for  $s$ , i.e.,  $(\mathbf{Q}V^{(i)})(s, a) = (\mathbf{B}V^{(i)})(s)$ , and let  $s_1 \in \text{Succ}[\mathcal{M}](s, a)$  be any successor with  $V^{(i)}(s_1) \geq (\mathbf{Q}V^{(i)})(s, a) > 0$ . By (IH1), there exists a goal path  $\sigma$ , starting from  $s_1$ , that satisfies (ii) and (iii) w.r.t.  $V^{(i)}$ . Notice that if  $\sigma$  violates (ii) or (iii) w.r.t.  $V^{(i+1)}$  at any point, then there must be a probabilistic transition out of  $\sigma$  into some state  $s'$  with  $V^{(i+1)}(s') > V^{(i+1)}(s)$ . Given the selection of  $s$ ,  $s'$  must be a goal state. We obtain the requested path for  $s$  by splitting  $\sigma$  at the failure point, inserting the corresponding transition into  $s'$ . For the induction step, let  $s$  be any state with  $V^{(i+1)}(s) > 0$ . Suppose as the second induction hypothesis (IH2) that the claim holds for all states  $s'$  with  $V^{(i+1)}(s') > V^{(i+1)}(s)$ . We obtain  $s_1$ , the path  $\sigma$ , and the failure state  $s'$ , i.e.,  $V^{(i+1)}(s') > V^{(i+1)}(s)$ , just as before. If  $s'$  is a goal state, then we can also construct the requested path for  $s$  just as before. If  $s'$  is not a goal state, then we use (IH2) to obtain a goal path  $\sigma'$  from  $s'$  that satisfies (ii) and (iii) w.r.t.  $V^{(i+1)}$ . The requested path for  $s$  results from concatenating  $s, a$ ; the part of  $\sigma$  up to the failure point  $s'$ ; and  $\sigma'$ .

□

The full proof is available in Appendix C.1.2.

**Lemma 11.2.** *Suppose  $V^{(0)}(s) = 0$  for all states  $s \in \mathcal{S}$ , and, for  $i > 0$ ,  $V^{(i)} = \mathbf{B}V^{(i-1)}$ . For all  $i \geq 0$ , there exists a policy  $\pi$  such that  $V^\pi \geq V^{(i)}$ .*

*Proof.* Let  $i \geq 0$  be arbitrary. We incrementally build  $\pi$  as follows. Let  $s \in \mathcal{S} \setminus \mathcal{S}_*$  be any non-goal state with  $V^{(i)}(s) > 0$  and  $\pi(s) = \perp$ . Let  $s = s_0, a_0, s_1, a_1, \dots, s_n$  be the goal path as per Lemma 11.1. For each  $0 \leq j < n$  where  $\pi(s_j) = \perp$ , set  $\pi(s_j) = a_j$ . Repeat until  $\pi$  is defined for all non-goal states with non-zero  $V^{(i)}$ -value. Notice that, by construction, executing  $\pi$  from any state  $s$  with  $\pi(s) \neq \perp$  has a chance to reach a goal state. Thus, the induced subgraph  $\mathcal{M}^\pi$  is an SSP, and  $V^\pi$  is the unique solution to the Bellman equations in that subgraph. Further, notice that it holds  $V^{(i)}(s) \leq (QV^{(i)})(s, \pi(a)) = (B^\pi V^{(i)})(s)$ , for all states  $\pi(s) \neq \perp$ , as per the selection of the paths and due to monotonicity. For the remaining states,  $\pi(s) = \perp$ , if  $s$  is a goal state, then  $V^{(i)}(s) \leq 1 = (B^\pi V^{(i)})(s)$  is trivially satisfied. If  $s$  is not a goal state, then  $V^{(i)}(s) = 0 \leq (B^\pi V^{(i)})(s)$ , by construction of  $\pi$ . In summary,  $V^{(i)} \leq B^\pi V^{(i)}$ . Via the monotonicity invariance, and given that  $V^\pi$  is the unique fixed point of  $B^\pi$ , we conclude that  $V^{(i)} \leq V^\pi$ .  $\square$

While Lemma 11.1 and Lemma 11.2 specifically consider the synchronous version of VI, i.e., updating the value of all states in each iteration, this assumption was made only for simplicity's sake. Both results can be trivially extended to asynchronous VI, such as the topological variant. The combination of Lemma 11.2 and Theorem 11.6 shows the desired property:

**Corollary 11.1.** *Let  $\epsilon > 0$  be a threshold. Suppose  $V$  is the value function resulting from TVI on  $\mathcal{M}$  and  $\epsilon$ . Let  $\pi_V$  be the policy constructed by Algorithm 11.2 for  $V$ . Then  $\pi_V$  satisfies  $V^{\pi_V} \geq V$ . If  $V = V^*$  is the optimal goal-probability function, then  $\pi_V$  is an optimal policy.*



# 12. MDP Heuristic Search

The algorithms presented in the previous chapter are attractive choices for solving MDPs optimally given their conceptual simplicity, and versatility in terms of supported MDP types and optimization objectives. To find a solution, they however necessitate the explicit construction of the entire (reachable) state space. For MDPs compactly described in the form of probabilistic planning tasks, this is typically not possible. MDP heuristic search has the potential to find optimal policies while building only a small fraction of the state space. This is accomplished by making use of heuristics, i.e., prior knowledge about the optimal value function, in order to identify regions of the state space not reachable from the initial state via any optimal policy. To find policies that are optimal for the initial state, yet not necessarily for every possible state, such regions can be safely ignored. Moreover, by exploiting the fact that it suffices to find just a single such policy, regions of the state space may not need to be considered even if relevant to other optimal policies.

In general, there are two categories of optimal MDP heuristic search algorithms, distinguished along the fundamental approach – LP vs. VI – on which they are based. Heuristic search as a variation of VI has a long history, and most known heuristic search instances fall into that category (e.g., Nilsson, 1971; Barto et al., 1995; Hansen and Zilberstein, 2001; Bonet and Geffner, 2003b). However, their application was so far limited almost exclusively to expected-cost minimization. Part of the reason for this is the lack of support of MDPs that do not comply with the SSP assumptions. Incompatibility issues stem from the use of the heuristic function as a means to initialize the value function. To provide the guarantee of finding an optimal solution, the heuristic function needs to be admissible, i.e., it has to optimistically bound the optimal value function. In our context, admissible heuristic functions hence need to upper bound the maximal goal probabilities. Yet, as we have seen in the previous chapter, starting from such a bound can result in converging to fixed points different from  $V^*$ . To make this class of heuristic search algorithms applicable beyond SSPs, Kolobov et al. (2011) have introduced FRET, which runs multiple iterations of complete heuristic searches, identifying and removing 0-reward cycles in between the iterations to escape sub-optimal fixed points. On the other side, there is heuristic search as an extension of the LP-formalism. This approach has been proposed only very recently (Trevizan et al., 2016). Being based on the LP-approach instead of VI, this variant eludes the problem of sub-optimal fixed point by its design, and hence it does not require the FRET outer-loop. As Trevizan et al. (2017a) show, MaxProb analysis is supported by this algorithm out of the box.

In the remainder of this chapter, we tailor a variety of existing MDP heuristic search algorithms to the goal-probability objectives. Correctness of all presented algorithms is shown with respect to the following definition:

**Definition 12.1** (Solving the goal-probability objectives). *Let  $\mathcal{A}$  be an algorithm that receives an MDP and (optionally) a convergence threshold  $\epsilon \geq 0$  as input, and returns a policy for the MDP as output. Let  $\mathcal{M}$  be an MDP with initial state  $s_I$  and maximal goal probabilities  $V^*$ . We say that  $\mathcal{A}$  **solves MaxProb** if, for some  $\epsilon \geq 0$ ,  $\mathcal{A}$  terminates on  $\mathcal{M}$ , returning a policy  $\pi$  s.t.  $V^\pi(s_I) = V^*(s_I)$ .  $\mathcal{A}$  **solves AtLeastProb** if, for some  $\epsilon \geq 0$ ,  $\mathcal{A}$  terminates on  $\mathcal{M}$ , returning either “impossible” if  $V^*(s_I) < \theta$ , or a policy  $\pi$  s.t.  $V^\pi(s_I) \geq \theta$ .  $\mathcal{A}$  **solves ApproxProb** if, for some  $\epsilon \geq 0$ ,  $\mathcal{A}$  terminates on  $\mathcal{M}$  with a policy  $\pi$  s.t.  $V^*(s_I) - V^\pi(s_I) \leq \delta$ . We*

simply say that  $\mathcal{A}$  solves the goal-probability objectives if  $\mathcal{A}$  solves all three: *MaxProb*, *AtLeastProb*, and *ApproxProb*.

We will mostly focus on traditional VI-based approaches. Concretely, we adapt AO\* (Nilsson, 1971), targeting acyclic MDPs as a simple, yet still relevant special case in which FRET is not necessary. For the general case, we design a variant of LRTDP (Bonet and Geffner, 2003b), and we introduce a family of depth-first heuristic searches (DFHS), which systematizes algorithm parameters underlying *improved* LAO\* (short ILAO\*) (Hansen and Zilberstein, 2001), heuristic dynamic programming (HDP) (Bonet and Geffner, 2003a), and learning depth-first search (LDFS) (Bonet and Geffner, 2006). All algorithms are equipped with *early-termination conditions*, addressing the *AtLeastProb* and *ApproxProb* objectives, by utilizing both lower and upper bounds on  $V^*$ . LRTDP and DFHS solve the goal-probability objectives for only SSPs. To deal with MDPs beyond SSPs, we revisit FRET as it has been proposed by Kolobov et al. (2011). Furthermore, we introduce a new FRET version, offering particular advantages in our setting. Finally, by viewing this new FRET variant as a natural extension of sub-procedures readily present in LRTDP and DFHS, we obtain VI-based heuristic search algorithms that can be applied to goal probability MDPs (and beyond) completely without any FRET outer-loop. For the sake of completeness (not a contribution of ours), we also provide a short description of the LP-based heuristic search algorithm for goal-probability analysis (Trevisan et al., 2017a) at the end of this chapter. Goal-probability heuristics will be the topic of Chapter 14.

The chapter is structured as follows. We start with an illustration of VI-based heuristic search as a whole. We continue with algorithms for the simpler acyclic case, proceed to the general case via FRET, and finally present the methods that natively solve arbitrary goal-probability MDPs.

## 12.1. Introductory Example

All VI-based heuristic search algorithms share a basic principle. In this section, we demonstrate this principle via the example MDP from Figure 12.1. Note that this MDP is acyclic, i.e., it satisfies the SSP assumptions. Hence, the Bellman equations have a unique solution. We for now ignore the complications arising from the existence of sub-optimal fixed points.

The following observation builds the basis of heuristic search. Notice that, when starting from a monotone upper bound  $V$  on  $V^*$ , to guarantee that a policy  $\pi$  greedy on  $V$  is optimal for the initial state  $s_0$ , it suffices that (1)  $\pi$  is closed for  $s_0$ , and (2) the values of the states reachable from  $s_0$  via  $\pi$  have converged. As per the definition of greedy policies, it holds that  $(BV)(s) = (B^\pi V)(s)$  for all states  $s$  visited by  $\pi$ . Then, given the convergence requirement  $V(s) = (BV)(s)$  for those states, it hence follows  $(B^\pi V)(s) = V(s)$ . In other words,  $V$  satisfies the Bellman equations of  $\pi$ 's value function. As by assumption these equations have a unique solution, we hence obtain  $V(s) = V^\pi(s)$  for all the visited states  $s$ . Finally, since  $V^* \leq V$  is preserved by the Bellman update operator, as per the monotonicity invariance, it follows

$$V^*(s) \leq V(s) = V^\pi(s) \leq V^*(s)$$

where the latter inequality holds by definition of  $V^*$ . In conclusion,  $\pi$  is optimal.

Reflecting this observation, the heuristic search approaches differ from VI's value computation by restricting Bellman updates to states reachable from  $s_0$  via actions greedy on the current value function. They terminate once during that computation a closed policy is found on which the Bellman updates no longer change. Throughout the computation, transitions are ignored whose estimated values are worse than those of the greedy options. This creates the potential to compute an optimal policy while considering only a

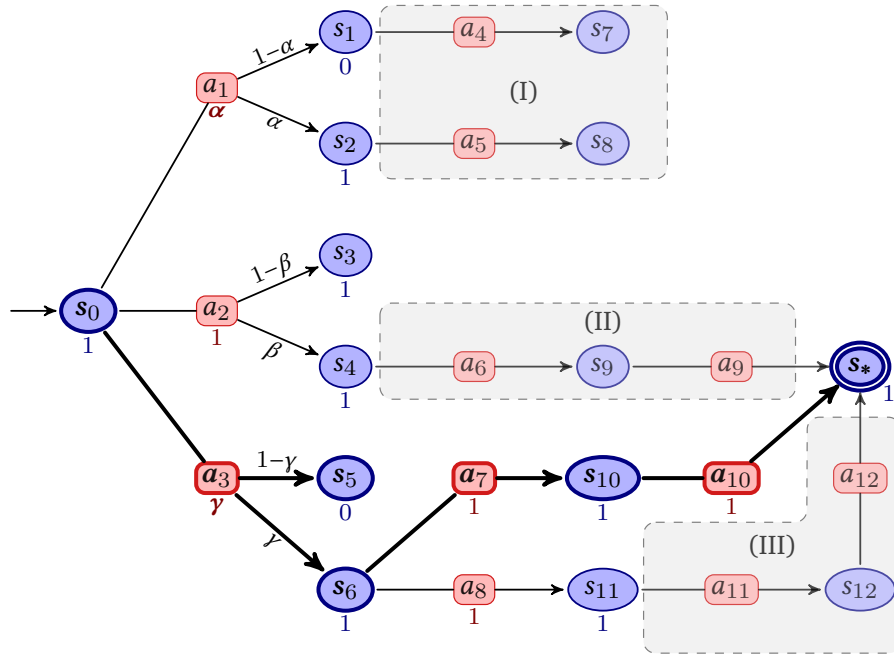


Figure 12.1.: Example MDP.  $0 < \alpha < \beta < \gamma < 1$ . There are 12 states.  $s_3, s_5, s_7, s_8$  are terminal, dead-end states.  $s_*$  is the only goal state. The  $v$  labels below state  $s$  show heuristic estimate  $H(s) = v$ . The  $q$  labels below actions show the corresponding  $QH$ -values. The computed optimal policy is highlighted in **bold**. The grayed-out areas indicate parts of the state space that were ignored.

small fraction of all states. To make effective use of this potential, admissible heuristic functions are used to initialize the value functions.

**Definition 12.2** (Goal-Probability Heuristic). *We call any monotone upper bound  $H$  on  $V^*$  an (admissible) goal-probability heuristic (or simply heuristic, if clear from the context).*

Equipped with an accurate heuristic function, heuristic search may be able to rule out many transitions right off the start. But note the general procedure does not even require a precise value initializations to guide the value updates. Namely, the information obtained through value updates themselves can often already prove transitions non-optimal, while considering just a few probabilistic successor states.

To demonstrate this generic yet basic principle, consider the MDP from Figure 12.1. Assume a monotone upper bound  $H$  with  $H(s_1) = H(s_5) = 0$ , and  $H(s) = 1$  elsewhere. Consider the value function  $V^{(0)}(s) := H(s)$ . Constrained by the greedy options under  $V^{(0)}$ , the first iteration of Bellman updates is restricted to states reached from  $s_0$  via  $a_2$ . Suppose we update the value of  $s_3$ , and propagate the resulting value change back to  $s_0$ , i.e., let  $V^{(1)}(s_3) := (BV^{(0)})(s_3) = 0$  and  $V^{(1)}(s) := V^{(0)}(s)$  for all other states;  $V^{(2)}(s_0) := (BV^{(1)})(s_0) = \gamma$  and  $V^{(2)}(s) := V^{(1)}(s)$  elsewhere. These updates change the greedy action of  $s_0$  from  $a_2$  to  $a_3$ . For the next iteration of Bellman updates, we follow the greedy actions  $\pi_{V^{(2)}}(s_0) := a_3$ ,  $\pi_{V^{(2)}}(s_6) := a_8$ , and  $\pi_{V^{(2)}}(s_{11}) := a_{11}$ . At this point, however, the values of all the states visited by  $\pi_{V^{(2)}}$  have converged. We terminate. Indeed,  $\pi_{V^{(2)}}$  is an optimal policy for  $s_0$ .

Upon termination, region (I) in Figure 12.1 has not been visited. Due to the value function initialization by  $H$ ,  $a_1$  could be proved non-optimal from the beginning. A single update was sufficient to remove  $a_2$  from the possible solution candidates at  $s_0$ , despite the lack of precise heuristic estimates for the corresponding successor states. Region (II) had not to be considered. Finally, the consideration of (III) has never become

**Algorithm 12.1:** GOALPROB-AO\*

---

**Input:** Acyclic MDP  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  (symbolically represented),  
Goal-probability heuristic  $H: \mathcal{S} \rightarrow [0, 1]$

**Output:** Policy  $\pi$  s.t.   MaxProb:      $V^\pi(s_I) = V^*(s_I)$   
                                  AtLeastProb:  $V^\pi(s_I) \geq \theta$  or “impossible”  
                                  ApproxProb:  $V^*(s_I) - V^\pi(s_I) \leq \delta$

---

```

1 Initialize empty subgraph (the search space)  $\hat{\mathcal{M}}$ ;
2 Initialize empty  $\pi^U, V^U, \pi^L, V^L$ ;
3 Insert  $s_I$  into  $\hat{\mathcal{M}}$ , i.e., insert  $s_I$  into  $\hat{\mathcal{S}}$ , and if  $s_I \in \mathcal{S}_*$ , insert it into  $\hat{\mathcal{S}}_*$ ;
4 Initialize  $V^U(s_I)$  and  $V^L(s_I)$ , i.e.,  $V^U(s_I) \leftarrow H(s_I)$ , and  $V^L(s_I) \leftarrow 1$  if  $s_I \in \hat{\mathcal{S}}_*$  else  $V^L(s_I) \leftarrow 0$ ;
5  $tip \leftarrow \{s_I\}$ ;  $solved \leftarrow \emptyset$ ;
6 while  $\top$  do
7   if AtLeastProb:  $V^L(s_I) \geq \theta$  then
8     | return  $\pi^L$ ; /* early termination (positive) */
9   if AtLeastProb:  $V^U(s_I) < \theta$  then
10    | return impossible; /* early termination (negative) */
11   if  $s_I \in solved$  then return  $\pi^U$ ; /* regular termination */
12    $s \leftarrow s_I$ ;
13   while  $s \notin tip$  do
14     |  $s \leftarrow$  pick some  $s \in Succ[\hat{\mathcal{M}}](s, \pi^U(s)) \setminus solved$ ; ♣
15   ExpandAndInitialize( $s$ ); /* (Algorithm 12.2) */
16   BackwardUpdate( $s$ ); /* (Algorithm 12.3) */

```

---

necessary even though the path from  $s_6$  via  $a_8$ ,  $a_{11}$ , and  $a_{12}$  would yield a valid alternative to the found optimal policy.

## 12.2. Acyclic MDPs

We introduce variants of AO\* and exhaustive search for goal-probability analysis. AO\* exploits an initial monotone upper bound on the goal probabilities in the way just sketched. In contrast, exhaustive search works solely with a monotone lower bound similar to VI. It offers early termination conditions over VI, yet in the limit may still need to construct the whole state space to find an optimal policy.

### 12.2.1. AO\*

AO\* (Nilsson, 1971) lifts standard graph search algorithms to AND/OR graphs. Algorithm 12.1 shows the pseudo-code of our AO\* variant for solving the goal-probability objectives. For ease of presentation, Algorithm 12.1 assumes an arbitrary (acyclic) MDP  $\mathcal{M}$  as input, abstracting away from representation details. Note, however, that  $\mathcal{M}$  does not need to be provided in an explicit form. In particular, Algorithm 12.1 can operate directly on the (budget-limited) probabilistic FDR task description  $\Pi$ , i.e., the probabilistic state space  $\mathcal{M}^\Pi$  does not need to be constructed beforehand. The algorithm incrementally builds a subgraph  $\hat{\mathcal{M}}$ , the *search space*, of the input MDP  $\mathcal{M}$ , to the extent necessary to find the desired policy.

**Algorithm 12.2:** ExpandAndInitialize of GOALPROB-AO\*

---

```

1 procedure ExpandAndInitialize( $s$ )
2    $tip \leftarrow tip \setminus \{s\}$ ;
3   /* Do not expand goal states */
4   if  $s \in \hat{S}_*$  then
5     return;
6   /* Generate transitions of  $s$ , and initialize data structures for new
7     states */
8   foreach  $t \in Succ[\mathcal{M}](s) \setminus \hat{S}$  do
9     Insert  $t$  into  $\hat{\mathcal{M}}$ ;
10    Initialize  $V^U(t)$  and  $V^L(t)$ , i.e.,  $V^U(t) \leftarrow H(t)$ , and  $V^L(t) \leftarrow 1$  if  $t \in \hat{S}_*$  else  $V^L(t) \leftarrow 0$ ;
11     $tip \leftarrow tip \cup \{t\}$ ;
12  Update  $\hat{\mathcal{P}}$  accordingly, i.e., copy all  $\hat{\mathcal{P}}(s, a, t) \leftarrow \mathcal{P}(s, a, t)$ ;

```

---

**Algorithm 12.3:** BackwardUpdate of GOALPROB-AO\*

---

```

1 procedure BackwardUpdate( $s$ )
2   /* process states in reverse topological order */
3    $queue \leftarrow$  empty heap;
4   if  $s \notin \hat{S}_* \wedge s$  is terminal then  $V^U(s) \leftarrow 0$ ;
5   if  $s \in \hat{S}_* \vee s$  is terminal then /* mark solved & propagate labels */
6      $solved \leftarrow solved \cup \{s\}$ ;
7     Insert all  $t \in Pred[\hat{\mathcal{M}}](s)$  into  $queue$ ;
8   else Insert  $s$  into  $queue$ ; /* insert for regular update */
9   while  $queue \neq \emptyset$  do
10     $t \leftarrow$  pop max state from  $queue$  w.r.t. topological order;
11    Update  $\pi^U(t)$ ,  $V^U(t)$ ,  $\pi^L(t)$ ,  $V^L(t)$ ; ★
12    if  $Succ[\hat{\mathcal{M}}](t, \pi^U(t)) \subseteq solved$  then
13       $solved \leftarrow solved \cup \{t\}$ ;
14      /* propagate solved label: */
15      Insert all  $t' \in Pred[\hat{\mathcal{M}}](t)$  into  $queue$ ;
16    else if  $V^L(t)$  or  $V^U(t)$  has changed then
17      /* propagate changed value: */
18      Insert all  $t' \in Pred[\hat{\mathcal{M}}](t)$  into  $queue$ ;

```

---

Adopting ideas from prior work (e.g., McMahan et al., 2005; Little et al., 2005; Smith and Simmons, 2006; Kuter and Hu, 2007), we maintain two value functions, namely both an upper bound  $V^U$  and a lower bound  $V^L$  on goal probability.  $V^L$  is simply initialized to 0 for all states but goal states.  $V^U$  is initialized by the provided monotone upper bound  $H$ . Additionally, we maintain two policies  $\pi^U$  and  $\pi^L$ , which are updated alongside the value functions, ensuring that they remain greedy policies of  $V^U$  and  $V^L$  throughout.

The subgraph  $\hat{\mathcal{M}}$  is initialized to consist only of the initial state  $s_I$ . The *tip states* of this MDP are the states that have not been *expanded* yet, i.e., those states of  $\hat{\mathcal{M}}$  that have not yet gone through Algorithm 12.2.

Every iteration of AO\*'s main while loop identifies a tip state  $s$  reachable from  $s_I$  via  $\pi^U$ .  $s$  is expanded, inserting its successors and transitions into  $\hat{\mathcal{M}}$ , while initializing  $V^U$  and  $V^L$  for newly created states. Then,  $V^U$  and  $V^L$  are updated at  $s$ , and possible value changes are propagated through  $\hat{\mathcal{M}}$ . When the policy  $\pi^U$  has been fully expanded, i.e., there is no tip state  $s$  reachable from  $s_I$  via  $\pi^U$  we terminate. The termination condition and the identification of the tip state  $s$  are both based on a state labeling mechanism. Goal and terminal states are labeled solved upon visiting them for the first time in AO\*'s update procedure. The remaining states are labeled solved when all their successors under the current policy  $\pi^U$  were marked solved. Every state not marked solved must have a descendant under  $\pi^U$  that has not been expanded yet. To find  $s$ , we trace a single path in  $\pi^U$ 's policy graph by starting from  $s_I$ , and iteratively selecting a successor under  $\pi^U$  that is not labeled solved. Since the MDP is acyclic, an unsolved terminal state in  $\hat{\mathcal{M}}$  must be reached eventually. Notice that this holds interdependently of the exact choice of the unlabeled successor states. However, the selections have a crucial impact on the performance of AO\*. On the one hand, guiding the exploration towards goal states, may lead to improving the lower bound  $V^L(s_I)$  quickly, fostering early termination. On the other hand, biasing the exploration towards dead-end states may allow to disprove the current policy's optimality early on, which can be beneficial for regular termination. We discuss different *outcome-selection strategies*, breaking ties at ♣, in Chapter 13.

Algorithm 12.3 shows the value update and labeling procedure. Again due to acyclicity, a single backward pass through  $\hat{\mathcal{M}}$  suffices to propagate value changes and to keep the labels in sync. Algorithm 12.3 processes states individually, starting from the expanded state  $s$ , updating its values and policies, and repeating with a state's predecessors if deemed necessary due to value changes or for label propagation. By considering the states in reverse topological order, no state must be touched more than once. When updating the policy  $\pi^U(s)$ , there might be multiple actions greedy on  $V^U$  for  $s$  at our disposal. As above, the choice does not affect correctness of the algorithm, yet it may translate into faster termination. Different policy tie-breaking strategies (★) are discussed in Chapter 13.

Regarding early termination, the lower bound enables *positive* early termination when we can already guarantee sufficient goal probability, namely  $V^L(s_I) \geq \theta$  in AtLeastProb, and  $V^L(s_I) \geq V^U(s_I) - \delta$  in ApproxProb. The upper bound enables *negative* early termination in AtLeastProb, when  $V^U(s_I) < \theta$ .

The correctness of GOALPROB-AO\* is easy to establish. Every iteration of the main while loop removes a state from the tip list. Given that every state is inserted at most once into this list, GOALPROB-AO\* must terminate after a finite number of steps. Due to the monotone initialization of  $V^L$  and  $V^U$ , and the monotonicity invariance of the Bellman update operator,  $V^L(s) \leq V^*(s) \leq V^U(s)$  holds for all states  $s \in \hat{\mathcal{S}}$  during the entire execution of the algorithm. Correctness in case of negative early termination hence follows directly. Suppose that the negative termination condition did not apply. That the returned policy fulfills the requirements of respective objective can be shown via induction on the maximal distance to terminal states in the policy graph induced by  $\pi^U$ , respectively  $\pi^L$  – which is defined since  $\mathcal{M}$  is acyclic. For MaxProb, observe that once any state  $s$  is marked solved, it holds  $V^{\pi^U}(s) = V^U(s) = V^*(s)$ . For goal states and terminal states this is trivially satisfied. Suppose a non-goal, non-terminal state  $s$  is marked solved. Then,

$$V^U(s) = \sum_{t \in \text{Succ}[\mathcal{M}](s, \pi^U(s))} \mathcal{P}(s, \pi^U(s), t) V^U(t)$$

as per the selection of  $\pi^U(s)$ . Since  $s$  is marked solved, it must hold that  $\text{Succ}[\mathcal{M}](s, \pi^U(s)) \subseteq \text{solved}$ . Via the induction hypothesis,  $V^U(t) = V^{\pi^U}(t)$  for all states  $t \in \text{Succ}[\mathcal{M}](s, \pi^U(s))$ . Plugging this into the definition of  $V^{\pi^U}(s)$  shows that  $V^U(s) = V^{\pi^U}(s)$ . Finally, the chain of relations  $V^*(s) \leq V^U(s) = V^{\pi^U}(s) \leq V^*(s)$  leads to the desired result  $V^{\pi^U}(s) = V^*(s)$ .

Finally, we need to prove that, in case of early termination, returning  $\pi^L$  achieves what we want, i.e.,

**Algorithm 12.4:** GOALPROB-EXHAO\*

---

**Input:** Acyclic MDP  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  (symbolically represented)  
**Output:** Policy satisfying the desired goal-probability objective, or “impossible”.

```

1 Initialize empty  $\hat{\mathcal{M}}$  and  $\pi^L$ ,  $V^L$ , and insert  $s_I$ ;
2  $tip \leftarrow \{s_I\}$ ;
3 while  $open \neq \emptyset$  do
4   if MaxProb:  $V^L(s_I) = 1$  then
      AtLeastProb:  $V^L(s_I) \geq \theta$ 
      ApproxProb:  $V^L(s_I) \geq 1 - \delta$ 
5   return  $\pi^L$ ; /* early termination */
6   Select some  $s$  from  $tip$ ;  $tip \leftarrow tip \setminus \{s\}$ ; ♣
7   ExpandAndInitialize( $s$ ); /* Algorithm 12.2 */
8   BackwardUpdate( $s$ ); /* Algorithm 12.3 */
/* regular termination */
9 if  $V^L(s_I) < \theta$  then return impossible;
10 else return  $\pi^L$ ;

```

---

$V^{\pi^L}(s_I) \geq \theta$  for *AtLeastProb*, and  $V^*(s_I) - V^{\pi^L}(s_I) \leq \delta$  for *ApproxProb*. Following the same inductive reasoning as above, we show that  $V^{\pi^L}(s) \geq V^L(s)$ . For the induction beginning, it holds for all goal states and terminal states that  $V^{\pi^L}(s) = V^L(s) = V^*(s)$  due to the special case treatments in Algorithm 12.2 and Algorithm 12.3. In the induction step, it holds that

$$V^L(s) = \sum_{t \in \text{Succ}[\mathcal{M}](s, \pi^L(s))} \mathcal{P}(s, \pi^L(s), t) V^L(t)$$

as per the definition of  $\pi^L(s)$ . By the induction hypothesis, then

$$V^L(s) \leq \sum_{t \in \text{Succ}[\mathcal{M}](s, \pi^L(s))} \mathcal{P}(s, \pi^L(s), t) V^{\pi^L}(t)$$

where the sum is exactly  $V^{\pi^L}(s)$ . Hence,  $V^{\pi^L}(s_I) \geq V^L(s_I) \geq \theta$ , concluding the proof for *AtLeastProb*, and  $V^*(s_I) - V^{\pi^L}(s_I) \leq V^U(s_I) - V^L(s_I) \leq \delta$  concluding the proof for *ApproxProb*.

**Theorem 12.1.** *Suppose  $\mathcal{M}$  is any acyclic MDP, and  $H$  is any monotone upper bound for  $\mathcal{M}$ . Then, GOALPROB-AO\* with  $H$  solves the goal-probability objectives for  $\mathcal{M}$ .*

**12.2.2. Exhaustive AO\***

We complement AO\* by an exhaustive search variant, which only maintains a lower bound  $V^L$  akin to VI, yet enables early termination by propagating value changes through the constructed MDP subgraph just as in GOALPROB-AO\*. Algorithm 12.4 shows the pseudo-code. The overall structure is similar to Algorithm 12.1. Due to the lack of an upper bound, the exploration of  $\mathcal{M}$  does however not follow any particular policy. Instead the tip states of  $\hat{\mathcal{M}}$  are interpreted as a global open list of states still to be expanded. Akin to the selection of the unlabeled successor states in GOALPROB-AO\*, the selection of the state to expand next has a direct influence on the development of  $V^L(s_I)$ . This can be leveraged to foster early termination via the outcome-selection strategy in ♣. Regular termination does not happen before not the entire reachable

fragment of  $\mathcal{M}$  has been visited. Analogously to GOALPROB-AO\*, the  $V^L$  value of a state is updated upon expansion, and value changes are propagated via the same backward pass through  $\hat{\mathcal{M}}$  as in Algorithm 12.3. Regarding early termination, in addition to the conditions used in GOALPROB-AO\*, we can terminate in MaxProb once  $V^L(s_I) = 1$ . This condition was not necessary in GOALPROB-AO\*, because when  $V^L(s_I) = 1$ , then all paths induced by  $\pi^L$  end in a goal state, so are labeled solved, and therefore GOALPROB-AO\* would terminate regularly anyways. We can terminate in ApproxProb, when  $V^L(s_I) \geq 1 - \delta$ . Again this condition is redundant in GOALPROB-AO\*, since it is subsumed by  $V^L(s_I) \geq V^U(s_I) - \delta$  tested there. Correctness in case of early termination follows just as before. In case of regular termination, the exhaustive construction of  $\mathcal{M}$  turns the update procedure into a variant of VI. Hence:

**Theorem 12.2.** *GOALPROB-EXHAO\* solves the goal-probability objectives for every acyclic MDP  $\mathcal{M}$ .*

### 12.3. General MDPs via FRET

In the presence of cycles, solving the goal probability objectives becomes significantly more complicated. Firstly, cycles introduce an interdependence between states, necessitating the propagation of value changes back and forth between states. A single pass of updates from leafs back to the initial state no longer suffices. Moreover, detecting convergence via the simple bottom-up labeling procedure from before is no longer possible either, given that states can be descendants of themselves. Last but not least, the Bellman equations may no longer have a unique solution. This is problematic, as due to the upper-bounding initialization of the value function, heuristic search is generally prone to converge to non-optimal fixed points. In the following, we address these issues in turn.

In Section 12.3.1, we revisit Bonet and Geffner’s (2003a) FIND-AND-REVISE algorithm, a generic schema for VI-based heuristic search on cyclic MDPs. We then design variants of LRTDP (Bonet and Geffner, 2003b) and depth-first heuristic search (DFHS), using the results for the FIND-AND-REVISE schema to prove their correctness. To optimally solve SSPs, the family of FIND-AND-REVISE algorithms can be used directly. A single run of heuristic search suffices. Handling MDPs beyond SSPs requires multiple iterations of heuristic search, yielding a series of fixed points, until the optimal one is found. Section 12.3.4 spells out this procedure, known as FRET (Kolobov et al., 2011), and introduces our new FRET variant.

#### 12.3.1. The Find-and-Revise Schema

The FIND-AND-REVISE schema (Bonet and Geffner, 2003a) summarizes the principal steps conducted by any VI-based heuristic search algorithm so to guarantee convergence to an  $\epsilon$ -consistent value function after a finite number of iterations. Algorithm 12.5 shows the pseudocode. Starting from a monotone upper bound  $V = H$ ,  $V$  is iteratively updated through an alternation of FIND and REVISE steps. FIND systematically explores the policy graph of some policy greedy on the current  $V$ , starting from the initial state, and searching for states whose values are not yet  $\epsilon$ -consistent. REVISE updates the value of such a state, possibly changing the greedy policy as a side effect. This is repeated until all states reached via the policy are  $\epsilon$ -consistent. The result is a value function  $V$  and greedy policy  $\pi$  such that  $\pi$  is closed for the initial state  $s_I$ , and  $V$  is  $\epsilon$ -consistent w.r.t.  $\pi$  and  $s_I$ , i.e.,  $(\Delta V)(s) \leq \epsilon$  holds for all  $s \in \mathcal{R}^\pi(s_I)$ .

Termination is guaranteed because of the monotone value function initialization, and since  $\epsilon > 0$ . Namely, suppose  $V^{(0)} = H, V^{(1)}, V^{(2)}, \dots$  is the series of value function computed along the FIND-AND-REVISE execution. As per the monotonicity invariance, it holds that  $V^{(0)} \geq V^{(1)} \geq V^{(2)} \geq \dots$ . Moreover, as long as FIND-AND-REVISE does not terminate, it holds that  $V^{(i-1)} - V^{(i)} > \epsilon$ . In other words, each round

**Algorithm 12.5:** FIND-AND-REVISE Schema

---

**Input:** MDP  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  (symbolically represented),  
 Goal-probability heuristic  $H: \mathcal{S} \rightarrow [0, 1]$ ,  
 Convergence threshold  $\epsilon \geq 0$  ( $\epsilon > 0$  if  $\mathcal{M}$  is not acyclic),  
**Output:**  $V \geq V^*$  and policy  $\pi$  greedy on  $V$  such that (i)  $\pi$  is closed for  $s_I$ , and  
 (ii)  $\forall s \in \mathcal{R}^\pi(s_I) : (\Delta V)(s) \leq \epsilon$

---

```

1 Initialize  $V$  by  $H$ ;
2 while  $\top$  do
3    $\pi \leftarrow$  policy closed for  $s_I$  and greedy on  $V$ ;
4    $s \leftarrow$  any state reachable from  $s_I$  via  $\pi$  s.t.  $(\Delta V)(s) > \epsilon$ ;           /* FIND */
5   if no such state exists then return  $V, \pi$ ;
6    $V(s) \leftarrow (BV)(s)$ ;                                                   /* REVISE */

```

---

of updates makes the value function become closer to a fixed point by at least  $\epsilon$ . The smallest possible fixed point in any goal-probability MDP is  $V^*$ . Hence, with  $\epsilon > 0$ , we obtain a finite upper bound on the FIND-AND-REVISE iterations:

$$\sum_{s \in \mathcal{S}} \frac{H(s) - V^*(s)}{\epsilon}$$

Upon termination, it holds for all states  $s$  reachable from  $s_I$  via  $\pi$  that

$$V(s) - (B^\pi V)(s) = V(s) - (BV)(s) \leq \epsilon$$

where the equality is true, because  $\pi$  is greedy on  $V$ . In other words, as  $\epsilon$  goes to 0,  $V$  approaches a fixed point of  $B^\pi$ . Suppose that the MDP is an SSP. Then,  $V^\pi$  is the unique such fixed point, i.e.,  $V$  approaches  $V^\pi$ . Due to the monotone initialization, and the monotonicity invariance of the Bellman update operator, it still holds that  $V \geq V^*$  upon termination. Therefore,

$$V^\pi \leq V^* \leq V$$

As discussed in Section 11.2.2, choosing an appropriate convergence threshold  $\epsilon$  allows to control the final precision  $\text{error}(\epsilon) = \max_{s \in \mathcal{R}^\pi(s_I)} V^*(s) - V(s)$ . From the previous system of inequalities, we obtain  $\max_{s \in \mathcal{R}^\pi(s_I)} V^*(s) - V^\pi(s) \leq \text{error}(\epsilon)$ . Hence, if  $\epsilon$  is chosen small enough, then FIND-AND-REVISE indeed computes an optimal policy. In summary:

**Theorem 12.3** (Bonet and Geffner, 2003a). *Let  $\mathcal{M}$  be an MDP with initial state  $s_I$ . Let  $H$  be any monotone upper bound for  $\mathcal{M}$ , and let  $\epsilon > 0$  be any convergence threshold. FIND-AND-REVISE( $\mathcal{M}, H, \epsilon$ ) terminates after at most  $\sum_{s \in \mathcal{S}} \frac{H(s) - V^*(s)}{\epsilon}$  iterations. The result is a value function  $V$  and a closed policy greedy on  $V$  such that  $V$  is  $\epsilon$ -consistent w.r.t.  $\pi$ .*

**Corollary 12.1.** *If  $\mathcal{M}$  in Theorem 12.3 is an SSP, then for sufficiently small  $\epsilon$ ,  $\pi$  is optimal.*

### 12.3.2. LRTDP

LRTDP (Bonet and Geffner, 2003b) is an extension of real-time dynamic programming (short RTDP) (Barto et al., 1995) by a state labeling procedure akin to AO\*. The solved labels provide a cheap yet sound termination condition. Moreover, they reduce redundant computations, fostering convergence, by removing

**Algorithm 12.6:** GOALPROB-LRTDP

---

**Input:** MDP  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  (symbolically represented),  
 Goal-probability heuristic  $H: \mathcal{S} \rightarrow [0, 1]$ ,  
 Convergence threshold  $\epsilon \geq 0$  ( $\epsilon > 0$  if  $\mathcal{M}$  is not acyclic),  
**Output:** Policy  $\pi$  as described in the text, or “impossible”.

```

1 Initialize empty  $\hat{\mathcal{M}}$  and  $\pi^U, V^U, \pi^L, V^L$ , and insert  $s_I$ ;
2  $tip \leftarrow \{s_I\}$ ;  $solved \leftarrow \emptyset$ ;
3 while  $\top$  do
4   [early termination criteria as in Algorithm 12.1]
5   if  $s_I \in solved$  then return  $\pi^U$ ; /* regular termination */
6    $trial \leftarrow$  empty stack;
7    $s \leftarrow s_I$ ;
8   while  $s \notin solved$  do
9     if  $s \in tip$  then
10       ExpandAndInitialize( $s$ ); /* Algorithm 12.2 */
11     if  $s \in \hat{\mathcal{S}}_* \vee s$  is terminal then
12        $solved \leftarrow solved \cup \{s\}$ ;
13       break;
14     Push  $s$  onto  $trial$ ;
15     Update  $\pi^U(s), V^U(s), \pi^L(s), V^L(s)$ ; ★
16     if Cyclic:  $V^U(s)$  has changed by less than  $\epsilon$  then break;
17      $s \leftarrow$  sample  $t \in Succ[\hat{\mathcal{M}}](s, \pi^U(s))$ ; ♣
18   while  $trial \neq \emptyset$  do
19      $s \leftarrow$  pop from  $trial$ ;
20     CheckAndMarkSolved( $s, \epsilon$ ); /* Algorithm 12.7 */
21     if  $s \notin solved$  then break;
```

---

from consideration states whose values have already converged. Algorithm 12.6 shows the pseudo-code of our goal-probability variant. The main change to the original version of LRTDP consists in maintaining both a lower bound and upper bound on  $V^*$ , along with corresponding policies, and adding the same early termination criteria as in GOALPROB-AO\* (cf. Algorithm 12.1).

LRTDP performs multiple iterations of *trials*, sampling a path induced by the current greedy policy  $\pi^U$ , starting from the initial state  $s_I$ , and ending in a terminal, goal, or any other state that is already labeled solved. Following Kolobov et al. (2011), we introduce an additional termination condition based on  $\epsilon$ -consistency in order to prevent trials from getting trapped in 0-reward cycles. Once a trial has been completed, the visited states are processed in reverse order, calling CheckAndMarkSolved (Algorithm 12.7) to update the solved labels. The labeling procedure guarantees that once any state  $s$  is labeled solved, the  $V^U$  values of all states reached from  $s$  via  $\pi^U$  are  $\epsilon$ -consistent. LRTDP terminates (regularly) when  $s_I$  is labeled solved. By default, the trials are sampled according to the MDP’s transition probabilities. The exact sampling method does not play any role for the termination or correctness guarantees. However, recall that the policy exploration strategy has an impact on the development of the lower and upper bounds, potentially translating into faster termination. Hence, to reflect this on LRTDP’s trials, we deploy different outcome-selection strategies to bias successor sampling at ♣. Secondly, recall that the actions chosen into

**Algorithm 12.7:** CheckAndMarkSolved of GOALPROB-LRTDP

---

```

1 procedure CheckAndMarkSolved( $s, \epsilon$ )
2   if  $s \in \text{solved}$  then return;
3    $\text{consistent} \leftarrow \top$ ;
4    $\text{open} \leftarrow \text{empty stack}$ ;  $\text{closed} \leftarrow \text{empty stack}$ ;
5   Push  $s$  onto  $\text{open}$ ;
6   while  $\text{open} \neq \emptyset$  do
7      $t \leftarrow \text{pop from open}$ ;
8     Push  $t$  onto  $\text{closed}$ ;
9     if  $t \in \text{tip}$  then
10      ExpandAndInitialize( $t$ );                                /* Algorithm 12.2 */
11      Update  $\pi^U(t), V^U(t), \pi^L(t), V^L(t)$ ;                  ★
12      if  $V^U(t)$  has changed by more than  $\epsilon$  then
13         $\text{consistent} \leftarrow \perp$ ;
14      else if  $t \in \hat{S}_* \vee t$  is terminal then
15         $\text{solved} \leftarrow \text{solved} \cup \{t\}$ ;
16      else
17        foreach  $t' \in \text{Succ}[\hat{M}](t, \pi^U(t))$  s.t.  $t' \notin (\text{solved} \cup \text{open} \cup \text{closed})$  do
18          Push  $t'$  onto  $\text{open}$ ;
19   if  $\text{consistent}$  then
20      $\text{solved} \leftarrow \text{solved} \cup \text{closed}$ ;
21   else
22     while  $\text{closed} \neq \emptyset$  do
23        $t \leftarrow \text{pop state from closed}$ ;
24       Update  $\pi^U(t), V^U(t), \pi^L(t), V^L(t)$ ;                  ★

```

---

$\pi^U$  upon the value updates is not less important. In particular, as suggested by Hansen and Zilberstein (2001), we stick to the currently chosen actions as long as possible, to avoid unnecessarily jumping between explorations of different, qualitatively equivalent, policies. If this is not possible, we consult a tie-breaking strategy, by default, breaking ties arbitrarily. Sticking to the convention introduced for GOALPROB-AO\*, the corresponding places in the pseudocode are indicated by ★. Different outcome-selection and policy tie-breaking strategies are the topic of Chapter 13.

Regarding the labeling method, note that because of cycles, it is not possible to simply propagate solved labels from children to parents as in AO\*. Instead, to determine whether a state  $s$  is solved, the sub-procedure CheckAndMarkSolved performs a systematic exploration of the policy graph induced by  $\pi^U$ . The exploration starts at  $s$ , and branches are cut off at  $\epsilon$ -inconsistent states, and states already marked solved. The visited states, including  $s$ , are labeled solved iff no  $\epsilon$ -inconsistent state was found. If the MDP is known to be acyclic,  $\epsilon$  can be set 0. In general, however,  $\epsilon > 0$  is required to guarantee termination (cf. Section 11.2.2). In contrast to the original version, we update the policies and value functions during the CheckAndMarkSolved explorations, which yields a small empirical advantage over computing only the Bellman residuals.

Regarding correctness. Note that each iteration of the main while loop updates the value of some state  $s$  that is reachable from  $s_I$  via  $\pi^U$  and whose Bellman residual satisfies  $(\Delta V^U)(s) > \epsilon$ , or else  $s_I$  is labeled solved. Due to `CheckAndMarkSolved`'s exhaustive exploration of the policy graph, when  $s_I$  is labeled solved, then  $V^U$  is  $\epsilon$ -consistent at all states reached from  $s_I$  via  $\pi^U$ . As per the first observation, LRTDP matches the `FIND-AND-REVISE` schema, i.e., Theorem 12.3 guarantees that  $s_I$  is labeled solved after a finite number of iterations. As per the second observation, LRTDP offers the same solution guarantees upon regular termination.

Correctness of early termination follows in a similar manner as for  $AO^*$ , i.e., by exploiting that  $V^L$  and  $V^U$  remain monotone lower, respectively upper bounds throughout. Note that this is true even in the general case, i.e., if early termination applies, then we can terminate the overall FRET process. Proving that  $V^{\pi^L}(s_I) \geq V^L(s_I)$  is a little bit more difficult than in the acyclic case. First, in contrast to  $AO^*$ , LRTDP does not guarantee that  $\pi^L$  remains greedy on  $V^L$  at all times, due to the lack of an exhaustive value propagation procedure as in Algorithm 12.3. Secondly, due to 0-reward cycles, selecting the actions into  $\pi^L$  arbitrarily does generally not guarantee to achieve the desired goal probability (cf. Example 11.2). Regarding the first issue, notice that  $\pi^L$  does actually not need to be greedy on  $V^L$ . It suffices that  $\pi^L$  selects an action whose expected value is at least as good as the value stored in  $V^L$ , i.e., if  $(*) V^L(s) \leq (QV^L)(s, \pi^L(s))$ . This property is guaranteed by LRTDP, simply because it holds  $(QV^L)(s, \pi^L(s)) \geq V^L(s)$  when  $\pi^L(s)$  and  $V^L(s)$  are updated, and since  $V^L$  is a monotone lower bound, this relation remains satisfied after updating  $V^L$  at other states. Regarding the second issue, notice that, similarly to the discussion in Section 11.2.4, as per the initialization of  $V^L$ , every value increase must originate from a goal state. Thus, if we update  $\pi^L$  only to reflect value increases, we make sure that  $\pi^L$  induces goal paths just as in Lemma 11.1 (replacing the requirement of greedy actions by  $(*)$ ). Via the arguments from Lemma 11.2, we then obtain the desired result:  $V^{\pi^L}(s) \geq V^L(s)$ . With that property in place, we can conclude

**Theorem 12.4.** *Suppose  $\mathcal{M}$  is any MDP, and  $H$  is any monotone upper bound for  $\mathcal{M}$ . Then, `GOALPROB-LRTDP` with  $H$  solves `AtLeastProb` and `ApproxProb` for  $\mathcal{M}$ . Moreover, if  $\mathcal{M}$  satisfies the SSP assumptions, then `GOALPROB-LRTDP` also solves `MaxProb` for  $\mathcal{M}$ .*

*Proof.* Due to the monotone initialization of  $V^L$  and  $V^U$ ,  $V^L(s) \leq V^*(s) \leq V^U(s)$  holds for all states  $s \in \hat{S}$  during the entire execution of the algorithm. If  $V^U(s) < \theta$ , then  $V^*(s) < \theta$ . When carefully updating  $\pi^L$  in the way described above,  $\pi^L$  satisfies  $V^{\pi^L}(s_I) \geq V^L(s_I)$ . Then, if  $V^L(s_I) \geq \theta$ ,  $\pi^L$  clearly achieves the `AtLeastProb` objective. In case of `ApproxProb` early termination, we have  $\delta \geq V^U(s_I) - V^L(s_I) \geq V^U(s_I) - V^{\pi^L}(s_I) \geq V^*(s_I) - V^{\pi^L}(s_I)$ . In conclusion, `GOALPROB-LRTDP` solves the `AtLeastProb` and `ApproxProb` objectives. Finally, given that, as explained above, `GOALPROB-LRTDP` instantiates the `FIND-AND-REVISE` schema, Corollary 12.1 shows the second part of the claim.  $\square$

### 12.3.3. Depth-First Heuristic Search

We finally consider systematic heuristic searches (not based on trials like LRTDP) with a strong depth exploration bias. Such an orientation is particularly beneficial in our setting. First and foremost, it creates a tendency towards reaching terminal or goal states quickly, i.e., states whose exact goal probability values are known. Secondly, value changes can be propagated quickly and efficiently to the initial state by leveraging the inverse exploration order. Lastly, solved labels can be computed with almost no additional overhead, even in the presence of cycles. We refer to the family of algorithms with such a strong depth bias by *Depth-First Heuristic Search (DFHS)*. Known instances are  $ILAO^*$  (Hansen and Zilberstein, 2001), heuristic dynamic programming (HDP) (Bonet and Geffner, 2003a), and learning depth-first search

**Algorithm 12.8:** GOALPROB-DFHS

---

**Input:** MDP  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  (symbolically represented),  
 Goal-probability heuristic  $H: \mathcal{S} \rightarrow [0, 1]$ ,  
 Convergence threshold  $\epsilon \geq 0$  ( $\epsilon > 0$  if  $\mathcal{M}$  is not acyclic),  
**Output:** Policy  $\pi$  as described in the text, or “impossible”.

```

1 Initialize empty  $\hat{\mathcal{M}}$  and  $\pi^U, V^U, \pi^L, V^L$ , and insert  $s_I$ ;
2  $tip \leftarrow \{s_I\}$ ;
3  $solved \leftarrow \emptyset$ ;  $visited \leftarrow \emptyset$ ;
4  $checkTerm \leftarrow \perp$ ;
5 while  $\top$  do
6   [early termination criteria as in Algorithm 12.1]
7   if  $checkTerm$  then                                     /* test for regular termination */
8     if  $LABEL \wedge s_I \in solved$  then return  $\pi^U$ ;
9     else if  $\neg LABEL$  then
10       $\pi^U, V^U, \pi^L, V^L \leftarrow$  run VI on  $visited$ , stop when their  $V^U$ -values are  $\epsilon$ -consistent ;    ★
11      if  $\pi^U$  has not changed then return  $\pi^U$ ;
12    $visited \leftarrow \emptyset$ ;
13   if  $LABEL$  then
14      $stack \leftarrow$  empty stack;  $minReach \leftarrow$  empty map;
15    $checkTerm \leftarrow$  DFHS_Exploration( $s_I$ ) ;                /* (Algorithm 12.9) */
```

---

(LDFS) (Bonet and Geffner, 2006). Their commonality lies in conducting depth-first explorations of an MDP subgraph defined by actions greedy on the current upper bound  $V^U$ . The algorithms differ in the exact subgraph considered, how depth-first branches are terminated, how the overall algorithm is terminated, and in whether or not updates are performed on the way down of exploration, on the way up, both, or neither of them. Algorithm 12.8 systematizes these parameters.

The main DFHS loop (Algorithm 12.8) consists in running a series of depth-first explorations (shown in Algorithm 12.9), stopping when either an early termination criterion is satisfied, or a signal is issued to start the regular termination test. Each exploration starts from the initial state  $s_I$ , and it considers only actions greedy on the current  $V^U$ . There are two options: searching the  $\pi^U$  policy graph specifically, or searching the MDP subgraph obtained by following in each state *all* actions greedy on  $V^U$ . However, the latter variant, employed by LDFS, is not effective for goal-probability analysis. Consider an MDP, in which many states can reach the goal with almost certainty, or just any MDPs in the context of loose upper bounds  $H$ , e.g., initializing  $V^U$  to 1 almost everywhere. The  $V^U$ -greedy graph then becomes the entire reachable state space, whose construction we try to avoid via heuristic search. We hence omit this option, and therewith LDFS, from our DFHS family.

We again maintain lower and upper bounds on goal probability, along with the corresponding policies. Early termination criteria for the AtLeastProb and ApproxProb objectives are the same as in GOALPROB-AO\*. Detecting whether a fixed point has been found can be done in two ways: (LABEL) by maintaining solved labels; or ( $\neg LABEL$ ) by running value iteration on the states of the  $\pi^U$  policy graph. Regular termination happens when (LABEL) the initial state is labeled solved; respectively ( $\neg LABEL$ ), if the greedy policy did not change during VI. Both cases require as prerequisite that  $\pi^U$  is closed for  $s_I$ , which is indicated by DFHS\_Exploration’s return value.

**Algorithm 12.9:** DFHS\_Exploration of GOALPROB-DFHS**Input:** State  $s$  s.t.  $s \notin \text{visited}$ ; Convergence threshold  $\epsilon \geq 0$  ( $\epsilon > 0$  if  $\mathcal{M}$  is not acyclic)**Output:** True if  $\pi^U$  has been fully expanded and no  $\epsilon$ -inconsistent state was found; false otherwise.

```

1 procedure DFHS_Exploration( $s, \epsilon$ )
2   if  $s \in \text{solved} \vee (s \notin \text{tip} \wedge s \text{ is terminal})$  then
3      $\text{solved} \leftarrow \text{solved} \cup \{s\}$ ;
4     return  $\top$ ;
5    $\text{visited} \leftarrow \text{visited} \cup \{s\}$ ;
6   if  $s \in \text{tip}$  then
7     ExpandAndInitialize( $s$ );                                /* Algorithm 12.2 */
8     if  $s \in \hat{S}_*$  then return  $\top$ ;
9     if  $s$  is terminal then  $V^U(s) \leftarrow 0$ ; return  $\perp$ ;
10    if CUTOFFTIP then
11      if UPDATEFORWARD  $\vee$  UPDATEBACKWARD then Update  $\pi^U(s), V^U(s), \pi^L(s), V^L(s)$ ;  ★
12      return  $\perp$ ;
13   $\text{flag} \leftarrow \top$ ;
14  if UPDATEFORWARD then
15    Update  $\pi^U(s), V^U(s), \pi^L(s), V^L(s)$ ;                      ★
16    if  $V^U(s)$  has changed by more than  $\epsilon$  then
17      if CUTOFFINCONSISTENT then return  $\perp$ ;
18      else  $\text{flag} \leftarrow \perp$ ;
19  if LABEL then
20     $\text{stackIdx} \leftarrow |\text{stack}|$ ;  $\text{minReach}[s] \leftarrow \text{stackIdx}$ ; push  $s$  onto  $\text{stack}$ ;
21  foreach  $t \in \text{Succ}[\hat{\mathcal{M}}](s, \pi^U(s))$  do
22    if  $t \notin \text{visited}$  then  $\text{flag} \leftarrow \text{DFHS\_Exploration}(t, \epsilon) \wedge \text{flag}$ ;
23    if LABEL then
24      if  $t \in \text{stack}$  then  $\text{minReach}[s] \leftarrow \min(\text{minReach}[s], \text{minReach}[t])$ ;
25      else if  $t \notin \text{solved}$  then  $\text{flag} \leftarrow \perp$ ;
26  if UPDATEBACKWARD  $\vee \neg \text{flag}$  then
27    Update  $\pi^U(s), V^U(s), \pi^L(s), V^L(s)$ ;                      ★
28    if  $V^U(s)$  has changed by more than  $\epsilon$  or  $\pi^U(s)$  has changed then
29       $\text{flag} \leftarrow \perp$ ;
30  if LABEL  $\wedge \text{stackIdx} = \text{minReach}[s]$  then
31     $S \leftarrow \text{pop from stack all states down to } s$ ;
32    if  $\text{flag}$  then  $\text{solved} \leftarrow \text{solved} \cup S$ ;
33  return  $\text{flag}$ ;

```

The states' values may be updated during the searches: (UPDATEFORWARD) on the way down of exploration, before applying  $\pi^U$  and generating the successor states; and (UPDATEBACKWARD) on the way back from exploration, once all successors under  $\pi^U$  were handled recursively. A search branch is always terminated at goal and terminal states, so as at any other solved labeled state. In addition, search branches may be cutoff: (CUTOFFTIP) at *tip* states, i.e., states that have not been expanded before, and (CUTOFFINCONSISTENT)

Parameter	ILAO*	HDP
LABEL	–	✓
UPDATEFORWARD	–	✓
UPDATEBACKWARD	✓	–
CUTOFFTIP	✓	–
CUTOFFINCONSISTENT	–	✓

Table 12.1.: Overview of parameters for known DFHS instances.

at  $\epsilon$ -inconsistent states, provided that the Bellman residual is available (UPDATEFORWARD is enabled).

Neither UPDATEFORWARD nor UPDATEBACKWARD is needed if VI is used to check termination (LABEL is set to false). However, to be able to determine correctly when to label a state as solved, one of the update flags needs to be activated. As suggested by Bonet and Geffner (2003a), to compute labels in the cyclic case, we keep track of the policy graph’s SCCs. Following Tarjan’s algorithm (cf. Chapter 3), this can be done with only little overhead alongside DFHS’s depth-first exploration. Terminal and goal states are labeled solved when visited. Other states are labeled solved when backtracking from an SCC  $S$  (line 30) such that (1) the values of all states in the SCC are  $\epsilon$ -consistent, and (2) every  $S$ -successor in the  $\pi^U$  policy graph, not part of the SCC itself, has been labeled solved already. The *flag* variable keeps track of the latter two requirements during exploration: *flag* is set to false whenever  $V^U(s)$  was not consistent during one of the updates, or when the requirements were violated in one of  $s$ ’s successors. Note that *flag* also needs to be set to false if  $\pi^U(s)$  changes after a backward update (line 28), as in this case, a fresh exploration starting from  $s$  is needed to reassure the closeness and consistency properties for the modified policy. For updates in the forward direction, this additional condition is not necessary because  $\pi^U$  is explored *after* the update. Following GOALPROB-AO\* and GOALPROB-LRTDP, the places relevant to policy tie-breaking are marked by ★. The outcome-selection strategy (previously indicated by ♣) is not as relevant for DFHS, given that DFHS\_Exploration explores all successors anyhow.

In summary, GOALPROB-DFHS offers 5 configurable parameter flags constrained by two dependencies:

- LABEL  $\Rightarrow$  UPDATEFORWARD  $\vee$  UPDATEBACKWARD
- CUTOFFINCONSISTENT  $\Rightarrow$  UPDATEFORWARD

In total, this yields 22 valid algorithm configurations. Note, however, that the combination of the forward-update flag (UPDATEFORWARD) and backward-update flag (UPDATEBACKWARD) is redundant. Backward updates will happen automatically as needed due to value changes during the forward updates. Leaving at least one of the two flags disabled leaves 14 configurations. The known DFHS instances ILAO\* (Hansen and Zilberstein, 2001) and HDP (Bonet and Geffner, 2003b) map into the configurations shown in Table 12.1.

Correctness of early termination in all algorithms variants follows via the exact same arguments as for GOALPROB-LRTDP. We emphasize here again that the cyclic case necessitates breaking ties in favor of the currently chosen actions when updating  $\pi^L$ , so to avoid issues due to 0-reward cycles. Regarding termination in general, note that it is generally mandatory to give precedence to the currently chosen action when updating  $\pi^U$ . Otherwise, the checks in GOALPROB-DFHS line 11 and in DFHS\_Exploration line 28 prevent termination. To prove termination of the DFHS family, we leverage once again the FIND-AND-REVISE schema. The policy exploration sub-procedure must return true eventually, because at some point all states will have been expanded; as per Theorem 12.3, no more  $\epsilon$ -inconsistent states are left; and therefore the policy also no longer needs to be changed. This suffices if LABEL is used for termination,

since at that point the initial state must have been labeled solved. Suppose that VI is used as the regular termination test. The termination condition is satisfied, if the values of all visited states are already  $\epsilon$ -consistent, and hence  $\pi^U$  will not have to be changed. If it is not satisfied, then at least one  $\epsilon$ -inconsistent gets updated. This cannot be repeated indefinitely, as per Theorem 12.3.

To show correctness in the case of regular termination, we prove that FIND-AND-REVISE's solution properties are satisfied: (1)  $\pi^U$  is closed for  $s_I$ , and (2)  $V^U$  is  $\epsilon$ -consistent in all states reachable from  $s_I$  via  $\pi^U$ . (1) must be true given Algorithm 12.9's exhaustive exploration, and since *checkTermination* can be true only if no search branch was cutoff. If VI is used to test termination, then (2) clearly holds. For the LABEL termination option, observe that a state  $s$  is labeled solved only if (1) and (2) are satisfied for  $s$ . This is trivially the case for terminal and goal states. On the other hand, labeling an SCC  $S$  solved requires  $\hat{flag}$  for the SCC entry state  $\hat{s} \in S$  to be true. As per the conjunction in line 15 of Algorithm 12.9,  $\hat{flag}$  aggregates the *flag* values of all states in the SCC. Hence, due to the exhaustive exploration, and the check in line 25,  $\hat{flag}$  can only be true if all successors outside the SCC are labeled solved. Finally, given the requirement that one of UPDATEFORWARD and UPDATEBACKWARD must be enabled, if  $\hat{flag}$  is true, then  $S$  can also not contain any  $\epsilon$ -inconsistent state. In summary, the states in  $S$  are labeled solved correctly, as per the requirements (1) and (2). In conclusion:

**Theorem 12.5.** *Suppose  $\mathcal{M}$  is any MDP, and  $H$  is any monotone upper bound for  $\mathcal{M}$ . Every valid instance of GOALPROB-DFHS with  $H$  solves AtLeastProb and ApproxProb for  $\mathcal{M}$ . If  $\mathcal{M}$  satisfies the SSP assumptions, then GOALPROB-DFHS also solves MaxProb for  $\mathcal{M}$ .*

#### 12.3.4. FRET Framework

To apply the FIND-AND-REVISE heuristic search schema to classes of MDPs, in which the Bellman equations have multiple solutions, Kolobov et al. (2011) proposed FIND-AND-REVISE-AND-ELIMINATE-TRAPS (short FRET). Like FIND-AND-REVISE, FRET maintains a monotone upper bound  $V^U$ , which is continuously updated throughout the FRET process. Each FRET iteration runs FIND-AND-REVISE until termination, i.e., until finding the next (potentially sub-optimal) fixed point. In between these iterations, FRET performs a *trap elimination* step, searching and processing *traps* so to escape sub-optimal fixed points, and by that ensures progress in the next iteration. Once no more traps are left, FRET has converged to  $V^*$ , and terminates.

Next, we introduce FRET more formally. Our exposition is based on the version from (Kolobov, 2013), which is what Kolobov et al. (2011) ended up implementing. We reformulate the method and theoretical results given by Kolobov (2013) in terms of the standard notions of end components (cf. Definition 10.5) and quotient systems (e.g., de Alfaro, 1997; Givan et al., 2003) (defined below). We provide alternative correctness arguments leveraging those notions. The alternative view gives rise to our new FRET variant, which is introduced at the end of this section.

#### Principles

In general, traps are end components composed of only 0-reward transitions (not to be confused with dead-end traps, as seen in Chapter 8). For the purpose of goal-probability analysis, the notion of traps simplifies to end components over non-goal states:

**Definition 12.3** (Goal-Probability Trap). *Let  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  be an MDP. Suppose  $T \subseteq \mathcal{S}$ .  $T$  is a **goal-probability trap** (trap, for simplicity) in  $\mathcal{M}$  if  $T$  is an end component of  $\mathcal{M}$  and  $T \cap \mathcal{S}_* = \emptyset$ . A trap  $T$*

in  $\mathcal{M}$  is **permanent** if  $\text{Succ}[\mathcal{M}](T) \subseteq T$ . A trap that is not permanent is called **transient**.

We say that a trap  $T$  is reachable from some  $s \in \mathcal{S}$  in  $\mathcal{M}$ , if there is any  $t \in T$  that is reachable from  $s$  in  $\mathcal{M}$ . A reachable trap in  $\mathcal{M}$  is one that is reachable from the initial state. Notice that the existence of sub-optimal fixed points has a one-to-one correspondence to the existence of traps:

**Theorem 12.6.**  $\mathcal{M}$  is an SSP if and only if  $\mathcal{M}$  contains no reachable trap.

*Proof sketch.* If  $\mathcal{M}$  is not an SSP, then there exists a policy  $\pi$  and a state  $s$  such that executing  $\pi$  from  $s$  never reaches an absorbing state (goal state, or state where  $\pi$  is not defined). The policy subgraph  $\mathcal{M}^\pi|_{\mathcal{R}^\pi(s)}$  is an EC and contains no goal states, i.e., is a trap. If  $\mathcal{M}$  has a goal-free EC  $\mathcal{M}'$ , then one can construct a counterexample  $\pi$  to the SSP conditions by assigning  $\pi(s) = a$  for all  $s \in \mathcal{S}'$  and some arbitrary  $a \in \mathcal{A}'(s)$  (which must exist due to the EC conditions). Let  $s \in \mathcal{S}'$ . Then,  $\mathcal{R}^\pi(s) \subseteq \mathcal{S}'$  by construction. Since  $\mathcal{M}'$  contains no goal states,  $\mathcal{R}^\pi(s) \cap \mathcal{S}_* = \emptyset$ , and since  $\pi$  is defined for all states in  $\mathcal{S}'$ ,  $\mathcal{R}^\pi(s) \cap \mathcal{S}_\perp^\pi = \emptyset$ . Hence, the execution of  $\pi$  from  $s$  never reaches an absorbing state, i.e.,  $\mathcal{M}$  is not an SSP.  $\square$

The proof arguments are spelled out in Appendix C.2.1.

As discussed in Chapter 11, SSPs have the nice property that  $V^*$  is the unique solution to the Bellman equations (Bertsekas, 1995). In other words, if there are no traps, then the issue of sub-optimal fixed points disappears. Now, notice that all states of a trap necessarily have the same goal probability, simply because each state of the trap can reach every other state in the trap with probability 1, as per the definition of end components. Hence, as far as the computation of goal probabilities is concerned, we could simply treat a trap as if it was a single state. This suggests the following simple (yet ineffective) algorithm: while the MDP  $\mathcal{M}$  contains a reachable trap  $T$ , obtain a new MDP  $\mathcal{Q}^T$ , in which  $T$  is collapsed into a single state  $\mathbf{t}$ ; continue with  $\mathcal{Q}^T$  and repeat until no more traps are left. The resulting MDP is guaranteed to be an SSP, which could then be handed to any off-the-shelf heuristic search algorithm. The sketched procedure is commonly known as the end-component decomposition of an MDP (e.g., Ciesinski et al., 2008). Yet, per se, it is not particularly helpful in our situation, given that it entails the construction of the whole state space, which to avoid was one of the main reasons why we moved to heuristic search in the first place. Before we detail how to leverage these observations more efficiently, let's however first flesh out the formal details, and show that the general idea does indeed work out.

Collapsing traps is captured formally via the notion of quotient systems:

**Definition 12.4** (Quotient MDP). Let  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  be an MDP. Let  $\emptyset \subset T \subseteq \mathcal{S}$  be a non-empty set of states. The **quotient** of  $\mathcal{M}$  and  $T$  is the MDP  $\mathcal{Q}^T = \langle \mathcal{S}^T, \mathcal{A}^T, \mathcal{P}^T, \mathbf{s}_I, \mathcal{S}_*^T \rangle$  where

- The states are  $\mathcal{S}^T = (\mathcal{S} \setminus T) \cup \{\mathbf{t}\}$ ;  $\mathbf{t}$  denoting a new state, representing all states of  $T$ . We use

$$\mathbf{s}^T(s) = \begin{cases} \mathbf{t} & \text{if } s \in T \\ s & \text{otherwise} \end{cases}$$

to denote the state in  $\mathcal{Q}^T$  associated with a state  $s \in \mathcal{S}$  of  $\mathcal{M}$ .

- The actions are  $\mathcal{A}^T = \mathcal{A} \cup \{ \langle s, a \rangle \mid s \in T, a \in \mathcal{A} \}$ .
- The transition probabilities are defined as follows. For all states  $s \in \mathcal{S} \setminus T$ ,  $a \in \mathcal{A}$ , and  $\mathbf{s}' \in \mathcal{S}^T$ :

$$\mathcal{P}^T(s, a, \mathbf{s}') = \begin{cases} \mathcal{P}(s, a, s') & \text{if } \mathbf{s}' = s' \in \mathcal{S} \\ \sum_{t \in T} \mathcal{P}(s, a, t) & \text{otherwise} \end{cases}$$

For all  $t \in T$ ,  $a \in \mathcal{A}$ , and  $\mathbf{s}' \in \mathcal{S}^T$

$$\mathcal{P}^T(\mathbf{t}, \langle t, a \rangle, \mathbf{s}') = \begin{cases} 0 & \text{if } \text{Succ}[\mathcal{M}](t, a) \subseteq T \\ \mathcal{P}(t, a, s') & \text{if } \mathbf{s}' = s' \in \mathcal{S} \\ \sum_{t' \in T} \mathcal{P}(t, a, t') & \text{otherwise} \end{cases}$$

- The initial state is  $\mathbf{s}_I = \mathbf{s}^T(s_I)$ .
- The goal states are  $\mathcal{S}_*^T = \{\mathbf{s}^T(s) \mid s \in \mathcal{S}_*\}$ .

The quotient of  $\mathcal{M}$  and  $T$  replaces all states from  $T$  by a single state  $\mathbf{t}$ . The incoming transitions of  $\mathbf{t}$  are those incoming to any state of  $T$ , and its outgoing transitions are those transitions of  $T$ -states exiting  $T$ . Additional action labels  $\langle s, a \rangle$  are introduced to be able to reflect multiple outgoing transitions from the states of  $T$  via the same action  $a$ . Observe that the quotient of  $\mathcal{M}$  and any trap  $T$  preserves the optimal goal probabilities:

**Theorem 12.7.** Suppose  $T$  is a trap in  $\mathcal{M}$ . Denote by  $V^*[\mathcal{Q}^T]$  the optimal goal-probability function of the quotient  $\mathcal{Q}^T$  of  $\mathcal{M}$  and  $T$ . It holds for all states  $s \in \mathcal{S}$  that  $V^*(s) = V^*[\mathcal{Q}^T](\mathbf{s}^T(s))$ .

*Proof sketch.* Since the only difference between  $\mathcal{M}$  and  $\mathcal{Q}^T$  are the transitions affecting  $T/\mathbf{t}$ , it suffices to establish  $V^*[\mathcal{Q}^T](\mathbf{t}) = V^*(t)$  for  $t \in T$ . If either side is 0, the claim follows trivially. For the remaining cases, note that optimal policies of both MDPs can be translated into one another, while preserving their goal-probability values. Let  $\pi^*$  be an optimal policy for  $\mathcal{M}$ . There must some  $t \in T$  and  $s \notin T$  such that  $\mathcal{P}(t, \pi^*(t), s) > 0$ . A corresponding policy for  $\mathcal{Q}^T$  is given  $\pi^T(\mathbf{t}) := \langle t, \pi^*(t) \rangle$ , and falling back to  $\pi^*$  in all other cases.  $\pi^T$  necessarily achieves the same goal probabilities than  $\pi^*$ . Vice versa, let  $\pi^T$  be an optimal policy for  $\mathcal{Q}^T$ . Assume  $\pi^T(\mathbf{t}) = \langle t, a \rangle$ . We construct a policy  $\pi$  for  $\mathcal{M}$  as follows. For all states  $s \notin T$ , we define  $\pi(s) := \pi^T(s)$ . To define the policy for the states in  $T$ , we start with  $\pi(t) := a$ . We proceed with the states  $t' \in T$  that have a transition  $a' \in \mathcal{A}(t')$  such that  $\mathcal{P}(t', a', t) > 0$  and  $\text{Succ}(t', a') \subseteq T$ . We set  $\pi(t') := a'$ , and continue with the states  $t'' \in T$  that have a transition  $a''$  going into  $t$  or  $t'$ , while staying within the trap. We iterate this process until all states in  $T$  were processed. The existence of the depicted transitions follows immediately from the EC properties. The construction of the policy ensures that the goal probabilities of  $\pi^T$  are preserved.

□

The full proof is provided in Appendix C.2.2.

### FRET-V

Kolobov et al. (2011) observed that in order to guarantee that FIND-AND-REVISE converges to  $V^U = V^*$ , it suffices to inspect just an MDP subgraph induced by  $V^U$  with respect to traps:

**Definition 12.5** (*V-Greedy Graph*). Let  $V$  be a value function. The *V-greedy graph* is the subgraph  $\mathcal{M}^V = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}^V, s_I, \mathcal{S}_* \rangle$  of  $\mathcal{M}$  where

$$\mathcal{P}^V(s, a, s') = \begin{cases} \mathcal{P}(s, a, s') & \text{if } a \text{ is greedy on } V \text{ for } s \\ 0 & \text{otherwise} \end{cases}$$

**Algorithm 12.10:** FRET- $V$ 


---

**Input:** MDP  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  (symbolically represented),  
 Goal-probability heuristic  $H: \mathcal{S} \rightarrow [0, 1]$ ,  
 Convergence threshold  $\epsilon > 0$ ,  
**Output:** Policy satisfying the desired goal-probability objective, or “impossible”.

```

1  $\mathcal{Q} \leftarrow \mathcal{M}$ ; /* start with the original MDP */
2  $V^U \leftarrow H$ ; /* initialize value function by the provided upper bound */
3 repeat
  /* (re)run heuristic search on the quotient MDP, starting from the
  previous value function  $V^U$ , to find the next fixed point */
4  $V^U, \pi^U, V^L, \pi^L \leftarrow \text{FIND-AND-REVISE}(\mathcal{Q}, V^U, \epsilon)$ ; /* Algorithm 12.5 */
5 if early-termination criterion applied then /* cf. Algorithm 12.1 */
6   return translate  $\pi^L$  into policy for  $\mathcal{M}$  (see proof of Theorem 12.7), respectively return
   “impossible”;
7 while the  $V^U$ -greedy graph contains some reachable state  $s$  s.t.  $(\Delta V^U)(s) > \epsilon$  do
8    $V^U(s) \leftarrow (BV^U)(s)$ ;
9    $S_1, \dots, S_n \leftarrow$  maximal SCCs of the  $V^U$ -greedy graph reachable from the initial state;
10   $\text{trapRemoved} \leftarrow \perp$ ;
11  foreach  $i = 1 \dots n$  do
12    if  $S_i$  is a leaf SCC and the  $V^U$ -greedy graph contains  $S_i$ -involving transitions then
13      /*  $\Rightarrow S_i$  is a permanent trap in the  $V^U$ -greedy graph */
14       $\mathcal{Q} \leftarrow$  quotient of  $\mathcal{Q}$  and  $S_i$  with collapsed state  $t$ ;
15       $V^U(t) \leftarrow V^U(s)$ , for any  $s \in S_i$ ;
16      delete all  $s \in S_i$  from  $V^U$ ;
17       $\text{trapRemoved} \leftarrow \top$ ;
18 until  $\neg \text{trapRemoved}$ ;
19  $\pi^U \leftarrow$  apply Algorithm 11.2 on  $V^U$  to construct a policy for  $\mathcal{Q}$ ;
20 return translate  $\pi^U$  into policy for  $\mathcal{M}$  (cf. proof of Theorem 12.7);

```

---

In other words, the  $V$ -greedy graph is simply the union of all graphs that are induced by the policies greedy on  $V$ . We abbreviate the set of states reachable from  $s$  in  $\mathcal{M}^V$  by  $\mathcal{R}^V(s) = \mathcal{R}[\mathcal{M}^V](s)$ . As per the principles of heuristic search, the focus on  $V$ -greedy actions may rule out from consideration regions of the state space whose expected values under  $V$  are already proved worse than some alternative. In effect, the  $V$ -greedy graph can be much smaller than the entire state space. At the same time, however, it still contains enough information to determine when  $V = V^*$  is satisfied:

**Theorem 12.8.** Suppose that  $V(s) = (BV)(s)$  for all states  $s \in \mathcal{R}^V(s_I)$ . If  $\mathcal{M}^V$  contains no reachable permanent trap, then  $V(s) = V^*(s)$  holds for states  $s \in \mathcal{R}^V(s_I)$ .

*Proof sketch.* Since  $\mathcal{M}^V$  contains no permanent traps,  $\mathcal{M}^V$  must contain a path from every state  $s \in \mathcal{R}^V(s_I)$  to some absorbing state. Since all paths in  $\mathcal{M}^V$  are over actions greedy on  $V$ , applying VI’s policy extraction procedure (Algorithm 11.2) on  $V$ , starting from the absorbing states, yields a policy  $\pi_V$  such that  $V^{\pi_V}(s) \geq V(s)$  is satisfied for all  $s \in \mathcal{R}^V(s_I)$  (cf. Theorem 11.6). The claim follows with  $V^*(s) \geq V^{\pi_V}(s)$ , and  $V(s) \geq V^*(s)$ , where the latter is true because  $V^*$  is the piecewise smallest fixed point.  $\square$

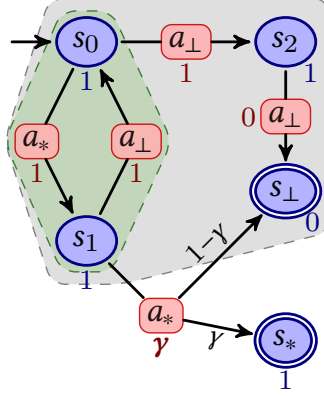


Figure 12.2.: Example MDP showing the necessity of value convergence over the entire  $V^U$ -greedy graph in FRET-V. The initial state is  $s_0$ .  $s_*$  is the only goal state. States are annotated by their  $V^U$ -value. Transitions are annotated by the corresponding  $Q^{V^U}$ -value. Colored areas as discussed in the text.

The full proof is available in Appendix C.2.3.

Algorithm 12.10 consolidates the theoretical observations made so far. In contrast to Kolobov et al.’s (2011) FRET variant, we additionally include a check to appropriately handle early termination of the underlying FIND-AND-REVISE algorithm. Moreover, the value-update part (lines 7 – 8) was not mentioned by Kolobov et al. explicitly, yet is required for the analysis as per Theorem 12.8 to be sound. Specifically, as FIND-AND-REVISE only guarantees that the values of states reached via the returned  $\pi^U$  have converged,  $V^U$  may have not yet converged on all states in the  $V^U$ -greedy graph, and this hence needs to be verified in a post process. Example 12.1 provides an illustration.

**Example 12.1.** Consider the MDP shown in Figure 12.2. The states are annotated by their  $V^U$ -value upon termination of the first FIND-AND-REVISE call. The gray area marks the  $V^U$ -greedy graph. The green part highlights the greedy policy  $\pi^U$  corresponding to the FIND-AND-REVISE call. The values of the states in the  $\pi^U$  policy graph ( $s_0$  and  $s_1$ ) have converged, as per the FIND-AND-REVISE termination guarantee. However,  $V^U$  has not yet converged on all states in the  $V^U$ -greedy graph since  $(BV^U)(s_2) = 0 < 1 = V^U(s_2)$ . The only trap in the  $V^U$ -greedy graph is given by the  $\pi^U$  subgraph, which is not permanent given the  $a_{\perp}$  transition leaving  $s_0$ . Even though there are no permanent traps, it is still not possible to extract a policy  $\pi$  from the  $V^U$ -greedy graph such that  $V^{\pi} = V^U$ , as per Theorem 12.8. In fact,  $V^U > V^*$ , since  $V^*(s) < V^U(s)$  holds for all  $s \in \{s_0, s_1, s_2\}$ .

Termination of the post-hoc update cycle follows similarly to FIND-AND-REVISE via the standard monotonicity argument. It is worth noting that the  $V^U$ -greedy graph can only become smaller during the course of these updates, and that the policy graph of  $\pi^U$  remains a subgraph of  $\mathcal{M}^{V^U}$  at all times. The latter is true, since the values in the  $\pi^U$ -graph have converged as per the FIND-AND-REVISE properties. The former then follows, as due to monotonicity, the values of states outside the  $\pi^U$ -graph either have converged too, or must decrease. An alternative to verifying value convergence explicitly is to instead remove *all* traps. In the absence of any trap, the properties of FIND-AND-REVISE together with Theorem 12.6 guarantee convergence to  $V^*$ . However, this variant may lead to removing more traps than necessary, as per Theorem 12.8. Moreover, identifying transient traps is algorithmically more difficult. Lastly, the analysis is still conducted on the  $V$ -greedy graph, which as we discuss in the next section, can become a bottleneck in the context of goal-probability objectives. We hence stick to FRET-V in the version proposed by Kolobov et al. (2011).

Algorithm 12.10 follows the general procedure sketched at the beginning of this section: it continuously updates a monotone upper bound  $V^U$  via FIND-AND-REVISE; after FIND-AND-REVISE has converged, it

searches and collapses permanent traps in the  $V^U$ -greedy graph; and these steps are repeated until no more permanent traps are left. Note that the permanent traps are exactly the leaf SCCs of  $\mathcal{M}^{V^U}$  that contain at least one transition (see also the detailed proof of Theorem 12.8 in Appendix C.2.3). To identify all permanent traps, it hence suffices to compute the maximal SCCs via, e.g., Tarjan's algorithm (cf. Chapter 3). The identified traps are collapsed one-by-one by building the corresponding quotient MDPs. The MDP transformations preserve optimal goal probabilities, as per Theorem 12.7. Since collapsing a trap strictly reduces the available transitions, of which there are only finitely many in the first place, FRET- $V$  must terminate eventually. The final  $V^U$ -greedy graph no longer contains permanent traps. As per Theorem 12.8, for sufficiently small  $\epsilon$ , we can construct from  $V^U$  an optimal policy for the quotient system. This policy can be translated into an optimal policy for the original MDP by acting, within collapsed traps, in a way so that an absorbing state is eventually reached with certainty (as sketched in proof of Theorem 12.7).

**Theorem 12.9.** *Let  $\mathcal{M}$  be any MDP, and  $H$  be any monotone upper bound for  $\mathcal{M}$ . FRET- $V$  with  $H$  and any FIND-AND-REVISE algorithm solves all the goal-probability objectives for  $\mathcal{M}$ .*

### FRET- $\pi$

Constructing the  $V^U$ -greedy graph may not always be feasible. In general, the number of policies greedy on  $V^U$  can be exponential in the number of states, and all of them are represented in the  $V^U$ -greedy graph. Specifically, consider the last trap analysis step in FRET- $V$ , i.e., verifying that  $V^*$  satisfies Theorem 12.8. Notice that, even if each optimal policy visits only a small number of states on its own, every state could very well be visited by some optimal policy. Hence, the  $V^*$ -greedy graph, considered in that last step, may already cover the entire state space. Although this issue may sound pathological at first glance, it can in fact be a problem in goal-probability analysis. For example, the delineated situation immediately arises in cases, where a large fraction of the states can reach the goal with certainty. Secondly, note that the  $V^U$ -greedy graph can be smaller than the entire state space only if the values in  $V^U$  allow to discriminate between some transitions. In particular, with an imprecise initialization, the  $V^U$ -greedy graph analyzed in the first trap elimination step will likely cover large parts of the state space.

To address these shortcomings, notice that the computation of an optimal policy does actually not require to consider the entire  $V^U$ -greedy graph. Namely, considering the policy  $\pi^U$  associated with the fixed point returned by FIND-AND-REVISE, a direct consequence of Theorem 12.6 is that  $V^U(s) = V^*(s)$  holds for the states visited by  $\pi^U$  if the  $\pi^U$  policy graph is trap-free:

**Theorem 12.10.** *Let  $V$  be a value function, and let  $\pi$  be a closed policy greedy on  $V$ . Suppose that  $V(s) = (BV)(s)$  for all  $s \in \mathcal{R}^\pi(s_I)$ . If  $\mathcal{M}^\pi$  contains no reachable permanent trap, then  $V(s) = V^*(s)$  holds for all states  $s \in \mathcal{R}^\pi(s_I)$ , and  $\pi$  is an optimal policy.*

*Proof.*  $\mathcal{M}^\pi$  cannot contain any transient trap, because each state may have at most one outgoing transition, as given by  $\pi$ 's action selection. Since  $\mathcal{M}^\pi$  neither contains permanent traps by assumption, it follows via Theorem 12.6 that  $\mathcal{M}^\pi$  satisfies the SSP assumptions. Hence,  $V^\pi$  is the unique solution of the Bellman equations in  $\mathcal{M}^\pi$ . Let  $s \in \mathcal{R}^\pi(s_I)$  be arbitrary. Since  $\pi$  is closed, and  $V$  is a fixed point of the Bellman equations in  $\mathcal{M}$  over  $\mathcal{R}^\pi(s_I)$ , it follows  $V(s) \geq V^*(s)$ . Since  $\pi$  is greedy on  $V$ , it holds that  $(BV)(s) = (B^\pi V)(s)$ . Therefore,  $V$  also satisfies the Bellman equations in  $\mathcal{M}^\pi$  over  $\mathcal{R}^\pi(s_I)$ . In conclusion,  $V^\pi(s) = V(s)$ . The claim follows with  $V^*(s) \leq V(s) = V^\pi(s) \leq V^*(s)$ .  $\square$

Algorithm 12.11 details our new FRET variant, which we baptize FRET- $\pi$ . It differs from the original version in terms of (1) the MDP subgraph considered: we only inspect the  $\pi^U$  policy graph instead of the

**Algorithm 12.11:** FRET- $\pi$ 


---

**Input:** MDP  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  (symbolically represented),  
 Goal-probability heuristic  $H: \mathcal{S} \rightarrow [0, 1]$ ,  
 Convergence threshold  $\epsilon > 0$ ,  
**Output:** Policy satisfying the desired goal-probability objective, or “impossible”.

```

1  $\mathcal{Q} \leftarrow \mathcal{M}$ ; /* start with the original MDP */
2  $V^U \leftarrow H$ ; /* initialize value function by the provided upper bound */
3 repeat
  /* (re)run heuristic search on the quotient MDP, starting from the
  previous value function  $V^U$ , to find the next fixed point */
4  $V^U, \pi^U, V^L, \pi^L \leftarrow \text{FIND-AND-REVISE}(\mathcal{Q}, V^U, \epsilon)$ ; /* Algorithm 12.5 */
5 if early-termination criterion applied then /* cf. Algorithm 12.1 */
6   return translate  $\pi^L$  into policy for  $\mathcal{M}$  (see proof of Theorem 12.7), respectively return
   “impossible”;
7  $S_1, \dots, S_n \leftarrow$  maximal SCCs of the  $\pi^U$  policy graph reachable from the initial state;
8  $\text{trapRemoved} \leftarrow \perp$ ;
9 foreach  $i = 1 \dots n$  do
10   if  $S_i$  is a leaf SCC and the  $\pi^U$  policy graph contains  $S_i$ -involving transitions then
11     /*  $\Rightarrow S_i$  is a permanent trap in the  $\pi^U$  policy graph */
12      $\mathcal{Q} \leftarrow$  quotient of  $\mathcal{Q}$  and  $S_i$  with collapsed state  $t$ ;
13      $V^U(t) \leftarrow V^U(s)$ , for any  $s \in S_i$ ;
14     delete all  $s \in S_i$  from  $V^U$ ;
15      $\text{trapRemoved} \leftarrow \top$ ;
16 until  $\neg \text{trapRemoved}$ ;
17 return translate  $\pi^U$  into policy for  $\mathcal{M}$  (cf. proof of Theorem 12.7);

```

---

entire  $V^U$ -greedy graph; (2) ensuring value convergence over all states reached via any policy greedy on  $V^U$  is then no longer necessary; and (3) the final policy  $\pi^U$  computed by FIND-AND-REVISE is already guaranteed to be an optimal policy for the quotient system  $\mathcal{Q}$ , as per Theorem 12.10, i.e., the provided  $\pi^U$  can be translated into an optimal policy for  $\mathcal{M}$  directly. Termination and correctness, for sufficiently small  $\epsilon$ , follow in the same manner as for FRET- $V$ , substituting Theorem 12.8 by Theorem 12.10.

**Theorem 12.11.** *Let  $\mathcal{M}$  be any MDP, and  $H$  be any monotone upper bound for  $\mathcal{M}$ . FRET- $\pi$  with  $H$  and any FIND-AND-REVISE algorithm solves all the goal-probability objectives for  $\mathcal{M}$ .*

Since FRET- $\pi$  and its FIND-AND-REVISE sub-procedure operate on a subgraph of the  $V^U$ -greedy graph at all times, the part of the state space explicitly constructed by FRET- $\pi$  is guaranteed to be a subgraph of that touched by FRET- $V$ . There are cases where FRET- $\pi$  finds an optimal policy while touching exponentially fewer states than FRET- $V$ . In terms of the number of iterations, or number of traps eliminated until termination, both methods are generally incomparable though. FRET- $V$  potentially eliminates more traps in each iteration, and may hence require fewer iterations overall. Yet not all these traps may actually need to be eliminated (we might eventually find an optimal policy not entering them), and each trap elimination step may be much more costly. Vice versa, since permanent traps in the  $\pi^U$  policy graph can be transient traps in the  $V^U$ -greedy graph, FRET- $\pi$  may eliminate traps that need not to be eliminated by FRET- $V$ . In summary, we obtain the following three observations:

**Theorem 12.12.** *There exist parameterized families of MDPs  $\mathcal{M}_n$ , and monotone upper bounds  $H_n$  where*

- I) FRET- $\pi$  is guaranteed to find an optimal policy while considering only states polynomial in  $n$ , whereas FRET- $V$  must consider exponentially many states in  $n$ .*
- II) FRET- $V$  terminates after a polynomial number of iterations in  $n$  (after eliminating a polynomial number of traps in  $n$ ), whereas FRET- $\pi$  needs an exponential number of iterations in  $n$  (needs to eliminate exponentially many traps in  $n$ ).*
- III) FRET- $V$  requires an exponential number of iterations in  $n$  (needs to eliminate an exponential number of traps in  $n$ ), whereas FRET- $\pi$  has the potential to terminate after just a polynomial number of iterations in  $n$  (after eliminating a polynomial number of traps in  $n$ ).*

We provide detailed examples for all three cases in Appendix C.2.4.

## 12.4. General MDPs without FRET

We finally consider heuristic search algorithms capable of solving the goal-probability objectives for arbitrary MDPs without relying on FRET. We present four such algorithms. First, we observe that the ELIMINATE-TRAPS part of our new FRET variant can be naturally viewed as an extension of sub-procedures readily available in LRTDP and the DFHS family. This results in new heuristic search variants that can be applied directly to goal-probability analysis without the need of any FRET outer-loop. Our adaptations can be straightforwardly extended to also handle the class of *generalized SSPs* (Kolobov et al., 2011), containing goal-probability analysis as a special case. Afterwards, we introduce a variant of the exhaustive anytime algorithm from Section 12.2.2 supporting cyclic MDPs. We close the section with Trevizan et al.’s (2016) IDUAL heuristic search algorithm, which in contrast to all algorithms presented so far, is based on the LP formulation.

### 12.4.1. Trap-Aware LRTDP

To prevent termination with a sub-optimal fixed point, we embed FRET- $\pi$ ’s post-convergence analysis directly into LRTDP. We do so by modifying LRTDP’s solved labeling procedure, additionally requiring the  $\pi^U$  policy subgraph rooted at a state  $s$  to be trap-free, before labeling  $s$  solved. If the values of the states in the policy graph have converged, but there are traps, we collapse the traps as in FRET- $\pi$ , yet seamlessly continue running LRTDP afterwards. Once the initial state is labeled solved, we have found an optimal policy for the traps-removed MDP, as per Theorem 12.10. It then only remains to map this policy into an optimal policy for the original MDP. The necessary changes almost only pertain to LRTDP’s solved labeling procedure. Algorithm 12.12 shows the adapted pseudo-code. We refer by TA-LRTDP to this new LRTDP variant.

Recall that the (permanent) traps in the policy graph are exactly the leaf SCCs of this graph. Hence, the main difference between Algorithm 12.12 and the original version (Algorithm 12.7) lies in computing the maximal SCCs alongside the policy exploration. Given that the original `CheckAndMarkSolved` function already explores the policy in a depth-first fashion, this computation boils down to only little bookkeeping overhead as per Tarjan’s algorithm. For the sake of readability, Algorithm 12.12 implements the exploration via a recursive procedure instead of the iterative method in Algorithm 12.7. A permanent trap is identified once the exploration backtracks out of an SCC  $S$ , none of whose states  $s \in S$  had a transition in the  $\pi^U$  policy graph that leaves  $S$ . The latter condition is indicated by the `isLeafSCC` flag. The collected traps are

**Algorithm 12.12:** CheckAndMarkSolvedAndEliminateTraps

---

```

1 procedure CheckAndMarkSolvedAndEliminateTraps( $s, \epsilon$ )
2   if  $s \in \text{solved}$  then return;
3    $\text{consistent} \leftarrow \top$ ;
4    $\text{closed} \leftarrow \text{empty stack}$ ;
5    $\text{traps} \leftarrow \emptyset$ ;  $\text{sccStack} \leftarrow \text{empty stack}$ ;  $\text{minReach} \leftarrow \text{empty map}$ ;
6   ExpandPolicy( $s, \epsilon$ );
7   if  $\text{consistent}$  then
8     if  $\text{traps} \neq \emptyset$  then
9       foreach  $S \in \text{traps}$  do
10         $\hat{M} \leftarrow \text{quotient of } \hat{M} \text{ and } S \text{ with collapsed state } t$ ;
11        Replace all  $t \in S$  by  $t$  in  $\pi^U, V^U, \pi^L, V^L$ ;
12      else  $\text{solved} \leftarrow \text{solved} \cup \text{closed}$ ;
13   else
14     while  $\text{closed} \neq \emptyset$  do
15        $t \leftarrow \text{pop state from closed}$ ;
16       Update  $\pi^U(t), V^U(t), \pi^L(t), V^L(t)$ ;
17 procedure ExpandPolicy( $s, \epsilon$ )
18   push  $s$  onto  $\text{closed}$ ;
19   if  $s \in \text{tip}$  then
20     ExpandAndInitialize( $s$ );
21   Update  $\pi^U(s), V^U(s), \pi^L(s), V^L(s)$ ;
22   if  $V^U(s)$  has changed by more than  $\epsilon$  then
23      $\text{consistent} \leftarrow \perp$ ;
24   else if  $s \in \hat{S}_* \vee s$  is terminal then
25      $\text{solved} \leftarrow \text{solved} \cup \{s\}$ ;
26   else
27      $\text{isLeafSCC} \leftarrow \top$ ;
28      $\text{stackIdx} \leftarrow |\text{sccStack}|$ ;  $\text{minReach}[s] \leftarrow \text{stackIdx}$ ;
29     push  $s$  onto  $\text{sccStack}$ ;
30     foreach  $t \in \text{Succ}[\hat{M}](s, \pi^U(s))$  do
31       if  $t \notin \text{closed} \wedge t \notin \text{solved}$  then
32          $\text{isLeafSCC} \leftarrow \text{ExpandPolicy}(t, \epsilon) \wedge \text{isLeafSCC}$ 
33       if  $t \in \text{sccStack}$  then
34          $\text{minReach}[s] \leftarrow \min(\text{minReach}[s], \text{minReach}[t])$ 
35       else  $\text{isLeafSCC} \leftarrow \perp$ ;
36     if  $\text{minReach}[s] = \text{stackIdx}$  then
37        $S \leftarrow \text{pop from sccStack all states down to } s$ ;
38       if  $\text{isLeafSCC}$  then  $\text{traps} \leftarrow \text{traps} \cup \{S\}$ ;
39       return  $\perp$ ;
40     else return  $\text{isLeafSCC}$ ;
41 return  $\perp$ ;

```

---

/\* Algorithm 12.2 \*/

♣

eliminated when the values of all states visited during policy exploration are consistent. States are labeled solved only if neither an inconsistent state, nor a trap was found. Note that each iteration of our new LRTDP variant either updates an  $\epsilon$ -inconsistent state, labels a state as solved, or eliminates a trap. Since there are only finitely many states and traps, the last two options cannot happen indefinitely. The former can also occur just a finite number of times, as per the usual FIND-AND-REVISE arguments. Hence, the TA-LRTDP must terminate eventually. Correctness in the case of early termination holds as before. In the case of regular termination, correctness follows via the exact same arguments as provided for FRET- $\pi$  and GOALPROB-LRTDP.

**Theorem 12.13.** *Let  $\mathcal{M}$  be any MDP, and  $H$  be any monotone upper bound for  $\mathcal{M}$ . TA-LRTDP with  $H$  solves all the goal-probability objectives for  $\mathcal{M}$ .*

By incorporating trap analysis into LRTDP directly, one gets rid of many redundant computations. Namely, after each iteration of FRET- $\pi$ , GOALPROB-LRTDP needs to reassure convergence for every state, including states for which an optimal policy has already been found. Likewise, after each fixed point computation of GOALPROB-LRTDP, FRET- $\pi$  needs to re-expand the entire policy, including parts that were already proved trap-free in some prior iteration. In contrast, once `CheckAndMarkSolvedAndEliminateTraps` is called on a state, whose policy successors are consistent and trap-free, the solved labels prevent this state from ever being touched again.

#### 12.4.2. Trap-Aware Depth-First Heuristic Search

FRET- $\pi$ 's post-convergence analysis can be integrated into the DFHS algorithm family in a manner similar to LRTDP. For the sake of brevity, we omit the full pseudocode, and just list the necessary changes. As in TA-LRTDP, TA-DFHS operates on a quotient MDP, in which traps are eliminated so to guarantee convergence to an optimal policy. Upon termination of the DFHS driver (Algorithm 12.8), the computed quotient system policy needs to be translated back into a policy of the original MDP. In the depth-first exploration (Algorithm 12.9), we enable the SCC computation regardless of whether solved labeling is turned on or off. To identify the leaf SCCs, we maintain an *isLeafSCC* flag, as in Algorithm 12.12. The policy graph contains a trap iff we backtrack out of an SCC (line 30 in Algorithm 12.9), while *isLeafSCC* is set to true. We abstain from eliminating a trap if *flag* was set to false, i.e., if either forward or backward updates are enabled, and an inconsistent state was found. Since in this case the policy might have changed, the identified trap might no longer need to be eliminated. When we backtrack out of a trap while *flag* is true, we eliminate the trap, but do not set solved labels. If a trap was eliminated, the policy exploration procedure returns false to signal the need of a policy exploration, starting from the newly created collapsed trap state.

Notice that each policy exploration via the modified `DFHS_Exploration` procedure updates the value of some  $\epsilon$ -inconsistent state, expands a tip state, marks a state as solved, or eliminates a trap. As per the arguments from previous section, none of these can happen infinitely often. Hence, `DFHS_Exploration` must eventually return true, informing the DFHS driver to check the termination condition. If DFHS then does not terminate, it must have updated the value of at least one  $\epsilon$ -inconsistent state, continuing with another policy exploration. Eventually, all states must be  $\epsilon$ -consistent. In conclusion, TA-DFHS terminates. Early termination is again not affected by any of the changes. Correctness of regular termination with solved labels follows exactly as in TA-LRTDP: a state  $s$  is labeled solved only if the policy graph rooted at  $s$  is  $\epsilon$ -consistent, and does not contain any trap. Assume VI is used for the termination test. Note that VI is only started if the call to `DFHS_Exploration` returned true, which entails that the policy graph is trap-free. After VI, the values of the states in that policy graph are  $\epsilon$ -consistent. Therefore, if the policy has not changed during the course of value updates, correctness follows just as before.

**Algorithm 12.13:** GOALPROB-EXHDFS**Input:**  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$ ,  $\epsilon \in (0, 1)$ **Output:** Policy satisfying the desired goal-probability objective, or “impossible”.

---

```

1 Initialize empty  $\hat{\mathcal{M}}$  and  $\pi^L$ ,  $V^L$ , and insert  $s_I$ ;
2  $tip \leftarrow \{s_I\}$ ;
3  $trace \leftarrow$  empty stack;  $sccStack \leftarrow$  empty stack;  $minReach \leftarrow$  empty map;
4 if  $Expand(s_I)$  then return  $\pi^L$ ;
5 else if  $V^L(s_I) < \theta$  then return impossible;
6 else return  $\pi^L$ ;

7 procedure  $Expand(s)$ 
8    $stackIdx = |stack|$ ;  $minReach[s] \leftarrow stackIdx$ ;
9   push  $s$  onto  $sccStack$  and  $trace$ ;
10   $ExpandAndInitialize(s)$ ; /* Algorithm 12.2 */
11  if  $PropagateValuesOnTrace()$  then return  $\top$ ;
12  foreach  $s' \in Succ[\hat{\mathcal{M}}](s) \setminus \mathcal{S}_*$  do ♣
13    if  $s' \in tip$  then
14      if  $Expand(s')$  then return  $\top$ ;
15    if  $s' \in sccStack$  then
16       $minReach[s] = \min(minReach[s], minReach[s'])$ ;
17  pop  $s$  from  $trace$ ;
18  if  $minReach[s] = stackIdx$  then
19     $S \leftarrow$  pop all states from  $sccStack$  down to  $s$ ;
20    Run VI on  $S$  until  $\epsilon$ -convergence;
21    return  $PropagateValuesOnTrace()$ ;
22  return  $\perp$ ;

23 procedure  $PropagateValuesOnTrace()$ 
24  foreach  $s \in trace$  from top to bottom do
25    Update  $V^L(s)$  and  $\pi^L(s)$ ;
26  return check early-termination criteria as in Algorithm 12.4;

```

---

**Theorem 12.14.** Let  $\mathcal{M}$  be any MDP, and  $H$  be any monotone upper bound for  $\mathcal{M}$ . TA-DFHS with  $H$  solves all the goal-probability objectives for  $\mathcal{M}$ .

### 12.4.3. Anytime Exhaustive Depth-First Search

As per VI's correctness arguments, heuristic search on a monotone lower bound converges to  $V^*$  without the need of FRET's post-convergence trap analysis. Algorithm 12.13 shows the pseudocode.

The principle idea is the same as in Section 12.2.2. Due to the absence of an upper bound, we generally have to explore the state space completely in order to find an optimal policy. However, by backpropagating value changes, from children to parents, during the exploration, we might be able to terminate early when sufficient goal probability is achieved. To guarantee convergence to an  $\epsilon$ -consistent value function, we keep track of the state space's SCCs during search, similar to TVI. When having fully expanded a maximal

SCC  $S$ , and all descendants of the states in  $S$ , we run VI on  $S$  until reaching  $\epsilon$ -consistency. By enforcing a depth-first order on the exploration of the state space, the SCCs can be computed with only little overhead, utilizing Tarjan's algorithm. We backpropagate value changes upon each state expansion (line 11), and when backtracking from an SCC after VI has converged (line 21). The depth-first exploration removes the necessity of tracking parent pointers. The ancestors of the currently expanded state in the search space are exactly the states stored in *sccStack*. However, in preliminary experiments, we have observed that updating the values of all states on the stack is almost always detrimental, causing too much overhead. In Algorithm 12.4, we hence only backpropagate values along the path traversed by the search, which already suffices to incentivize early termination. The outcome-selection strategy ( $\clubsuit$ ) may control locally at each state, in which order the successors are going to be processed.

As per the usual monotonicity arguments, each call to VI must reach  $\epsilon$ -convergence after a finite number of iterations. Given that the algorithm expands every (reachable) state exactly once in the worst case, it must hence terminate eventually. Correctness in the case of early termination follows similarly to GOALPROB-LRTDP. In the case of regular termination, the exhaustive search algorithm boils down to TVI, performing additional value updates, which however does not impede any of TVI's correctness arguments.

**Theorem 12.15.** *GOALPROB-EXHDFS solves the goal-probability objectives for every MDP  $\mathcal{M}$ .*

#### 12.4.4. I-Dual

Breaking with the tradition, IDUAL (Trevizan et al., 2016; Trevizan et al., 2017a) is based on the LP formulation of an optimal policy, and by that eludes the issue of getting trapped in sub-optimal fixed points by design. But despite of exchanging the foundation, IDUAL is not so much different from the heuristic search approaches that we have seen so far. IDUAL incrementally expands a state space subgraph  $\hat{\mathcal{M}}$  via policy explorations, using a monotone upper bound  $H \geq V^*$  to estimate the goal probability of the tip states *tip* of that graph. Initially,  $\text{tip} = \hat{\mathcal{S}} = \{s_I\}$ . Each iteration computes a policy  $\pi$  up to the tip states that would be optimal if the tip states' values were indeed the exact goal probabilities. The subgraph is extended by expanding the tip states  $\mathcal{R}^\pi(s_I) \cap \text{tip}$  reached from the initial state through that policy, possibly inserting new tip states into  $\hat{\mathcal{M}}$ . Once  $\mathcal{R}^\pi(s_I) \cap \text{tip}$  becomes empty,  $\pi$  is guaranteed to be an optimal policy, and IDUAL terminates. The policy  $\pi$  is computed by solving the MaxProb dual LP (Definition 11.2) over  $\hat{\mathcal{M}}$ , adjusting the optimization function according to the tip states:

$$\underset{y}{\text{maximize}} \sum_{s_* \in \hat{\mathcal{S}}_*} in(s_*) + \sum_{\hat{s} \in \text{tip}} in(\hat{s})H(\hat{s}) \quad (12.1)$$

Since  $\hat{\mathcal{M}}$  will eventually become the entire reachable state space, which cannot be extended any further, IDUAL is guaranteed to terminate. Notice that, due to the monotone upper bound  $H$ , the LP cannot underestimate the value of any policy. Suppose  $\pi$  is the final policy before termination. Hence,  $\sum_{\hat{s} \in \text{tip}} in(\hat{s}) = 0$ , and the objective value of the LP becomes exactly the goal probability achieved by the policy for  $s_I$ . Since the LP optimizes over all possible policies, given the previous observation,  $\pi$  must therefore be optimal. In vein of Section 12.1, IDUAL follows the very same principles as the VI-based heuristic search algorithms from before: (1) state space explorations are guided by policies optimal according to the information gathered so far; and (2) sub-optimal parts of the state space are identified without exploration via an initial estimate  $H$  on  $V^*$ . Consequently, IDUAL also shares the same potential of finding an optimal policy without building the entire reachable state space.

IDUAL can be straightforwardly adapted to support early termination on the weaker goal-probability objectives. The objective values of the generated LP solutions yield a sequence of non-increasing goal-probability upper bounds. Hence, we can interrupt IDUAL's iterations via the same *negative* early termination conditions as in all our other heuristic search algorithms. Specifically, we can terminate in case of AtLeastProb with “impossible” once the objective value  $\hat{V}^*$  of IDUAL's current LP drops below  $\theta$ ,  $\hat{V}^* < \theta$ . Moreover, we can terminate in ApproxProb, when  $\hat{V}^* \leq \delta$ , returning the corresponding policy. Given that IDUAL does not maintain a goal-probability lower bound, and it cannot be easily extended to maintain such bound, *positive* early termination is not possible.

# 13. Heuristic Search Tie-Breaking Strategies

A heuristic search algorithm lays down the general framework to guarantee convergence to an optimal solution. However, this framework leaves some room for diverging implementations, given that not all operations are uniquely determined. As has already been pointed out by Hansen and Zilberstein (2001) for  $\text{LAO}^*$ , smartly resolving these ambiguities can be vital for the efficiency of the search. This especially pertains to our setting, where good anytime behavior on  $V^L$  and/or  $V^U$  can be particularly beneficial for early termination. We explore the potential of fostering this via

**Action selection tie-breaking (★):** Biasing the tie-breaking in the selection of “best” actions  $\pi^U$  greedy with respect to  $V^U$ .

**Outcome selection (♣):** Biasing the outcome-state sampling during trials (LRTDP), respectively the choice of expanded states in  $\text{AO}^*$ , and in the exhaustive search variants.

As suggested by Hansen and Zilberstein (2001), in (★), we generally stick to the currently chosen actions as long as possible. We apply tie-breaking, changing  $\pi^U(s)$ , only if some other action becomes strictly better in  $s$ . This follows the rationale of effectively using already gathered information, given the likelihood of a different policy visiting parts of the state space that have not been explored so far, and hence in which the current  $V^U$  estimates might still be rather imprecise. Furthermore, recall that this strict updating rule is generally necessary to guarantee termination of DFHS.

**Example 13.1.** To illustrate the effect of these tie-breaking strategies on heuristic search, consider the MDP from Figure 13.1. We assume MaxProb analysis. The outcome probabilities of the probabilistic transitions do not matter for the purpose of this example. Note that the MDP is acyclic. For the sake of simplicity, we follow a procedure like  $\text{AO}^*$  to find an optimal policy. However, the following observation does apply to any VI-based heuristic search algorithm alike. Suppose that we start from the (not very accurate) initial goal-probability estimates, as given in the figure. Hence, initially, every policy is greedy on  $V^U$ . To demonstrate the effect of

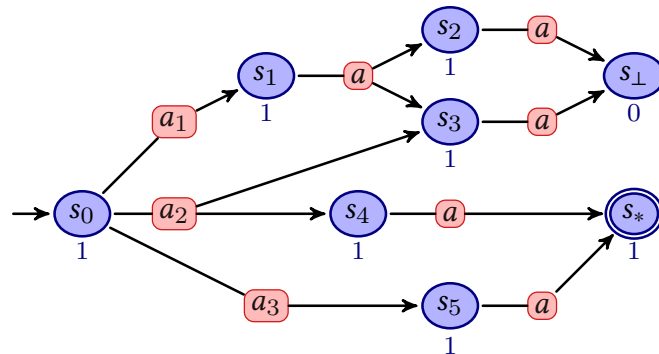


Figure 13.1.: MDP used in Example 13.1 to demonstrate the impact of different tie-breaking strategies.  $s_0$  is the initial state,  $s_*$  is the (only) goal state,  $s_\perp$  is a terminal dead-end state. Transition function as depicted. States are annotated by their heuristic value.

(♣), consider first the policy, which explores the top part of the MDP:  $\pi^U(s_0) = a_1$  (and  $\pi^U(s) = a$  for the remaining non-goal, non-terminal states  $s$ ). During the expansion of the policy,  $AO^*$  has the option (\*) to follow from  $s_1$  either the transition to  $s_2$  or to  $s_3$ . Suppose  $s_2$ . Then the Bellman updates after the expansion change the value of  $s_2$  to  $V^U(s_2) = 0$ , and the value of  $s_1$  to some  $V^U(s_1) < 1$ . Afterwards,  $a_1$  is no longer greedy on  $V^U$  for  $s_0$ . Note, however, that  $a_2$  is still a greedy action for  $s_0$ , and  $a_2$  remains greedy as long as  $s_3$  has not been explored. In effect, when choosing next  $\pi^U(s_0) = a_2$ ,  $AO^*$  will in the end have explored the entire MDP before the optimal policy,  $\pi^U(s_0) = a_3$ , was found. Now, observe that if we had instead chosen  $s_3$  in (\*),  $a_2$  would have been removed from the choices of greedy actions at  $s_0$  right away. The  $a_2$  subgraph would have not been expanded. Finally, notice that we could have avoided considering any of the  $a_1$  and  $a_2$  subgraphs, if the (★) tie-breaking strategy had chosen  $\pi^U(s_0) = a_3$  from the start.

In general, we cannot expect to always select the best option, such as the ones in Example 13.1, as this ultimately requires knowledge attainable only via the options' explorations. To still bias the selections to the "good" options, we experiment with a variety of strategies, some of which had been used already successfully in other contexts. In what follows, where a strategy specifies one of (★) or (♣) only, the other setting is as in the *default* strategy. Before we describe the different strategies, recall that the outcome-selection strategy is relevant only for  $AO^*$ , LRTDP, and the exhaustive search variants. The latter algorithms behave differently from the other heuristic search variants, as they do not maintain an upper bound, i.e., there is no selection (★) of actions greedy with respect to  $V^U$ , and the explorations do not follow any particular policy, i.e., the selection in (♣) pertains to *all* open states. We discuss the concrete use of (♣) in our exhaustive search algorithms at the end of this chapter.

**Default strategy.** We adopt the commonly used settings. This is, we break ties arbitrarily during greedy policy construction (★). Specifically, in our implementation, we always choose the first action encountered with maximal expected value. There is no bias (♣) on outcome states in  $AO^*$  (open outcome states are selected in the order in which they are generated); in LRTDP the outcome states are sampled according to the transition probabilities.

**Most-likely outcome bias.** We also tried LRTDP's default outcome-selection bias in  $AO^*$ , always selecting the most likely open outcome state during the policy graph traversals.

**Gap-bias strategy.** Inspired by BRTDP (McMahan et al., 2005), we bias the selections according to the gap between the upper and lower bounds. This gap can be understood as the uncertainty in the values computed so far, larger gaps meaning more uncertainty. As per our motivation for action selection tie-breaking, we want to introduce a bias towards policies for which we have gathered the most information so far. Hence, in (★), we break ties in favor of actions  $a$  that *minimize* the expected gap

$$\sum_t \mathcal{P}(s, a, t) (V^U(t) - V^L(s))$$

In contrast, the purpose of (♣) is to foster the retrieval of new information from less explored parts of the policy graph. Therefore, for  $AO^*$ , we prefer states in (♣) with *larger* value gap  $V^U(t) - V^L(t)$ . In LRTDP, we sample the successors according to the renormalized weighed probabilities

$$(V^U(t) - V^L(t)) \mathcal{P}(s, \pi^U(s), t)$$

attaching successors more probability weights the larger their current value gap.

***h*-bias strategy.** We use a heuristic function  $h$  from classical planning to estimate the cost of the cheapest goal path from a state. We generally break ties in favor of states with smaller  $h$  value. Specifically, in (★), we select among the greedy actions for a state  $s$  one that minimizes the expected heuristic value

$$\sum_t \mathcal{P}(s, a, t) h(t)$$

To select the outcome state (♣) in  $AO^*$ , we choose one with smallest heuristic value. For LRTDP, we bias the random sampling of the successor state  $t \in \text{Succ}(s, \pi^U(s))$  by renormalizing the weighted probabilities:

$$\frac{1}{h(t)} \mathcal{P}(s, \pi^U(s), t)$$

i.e., we prioritize high probability outcomes with small  $h$  value. We also experimented with a variant, setting the determinized action costs to the negated logarithm of outcome probability (e.g., Jimenez et al., 2006). This is compelling in principle because, then, the maximal success probability of any goal path from a state can be computed through a *bisimulation*-based merge-and-shrink heuristic (Helmert et al., 2014). However, while interesting from a theoretical perspective, in practice, constructing such a heuristic is typically not feasible.

**Helpful actions strategy.** Inspired by common methods in classical planning (e.g., Hoffmann and Nebel, 2001; Helmert, 2006), we experiment with a *helpful actions* strategy, which in (★) prefers to set  $\pi^U(s)$  to actions participating in a delete-relaxed determinized plan for  $s$ , if such an action exists.

**State selection in exhaustive search.** We apply the selection strategies for (♣) to the set of states in the current search graph  $\hat{\mathcal{M}}$  that still need to be expanded. Following ExhDFS, the default strategy in ExhAO\* is depth-first, the rationale being to try to reach absorbing states quickly. The (♣) selection strategy then serves as tie-breaking rule to select among the deepest yet-to-be expanded states. For ExhAO\* we also experimented with a breadth-first strategy, just for comparison.



# 14. Goal-Probability Occupation-Measure Heuristics

While heuristic search is a popular method to optimally solve probabilistic planning problems, research on MDP heuristic functions that actually do reason about action-outcome uncertainty has so far been scarce. The most widely deployed approach to obtain heuristics for MDP planning is based on the determinization of the MDP into a classical planning task (e.g., Bonet and Geffner, 2005; Jimenez et al., 2006). This is appealing in principle, given that, e.g., the cost of the cheapest *determinized* plan for a state is an admissible estimation of the minimal expected-cost to reach the goal from that state in the MDP. While computing the cost of the cheapest determinized plan is still intractable in general, given the computational complexity of optimal classical planning, there however exists huge conglomerate of highly developed optimal classical planning heuristics that one can leverage to tightly approximate the plans' costs. The limits inherent to the determinization approach become quickly apparent in the context of goal-probability analysis. Here, heuristic search requires the heuristics to upper bound the goal probability. Yet, through determinization, one can solely obtain upper bounds of a *qualitative* kind: dead-end detection. Namely, on the one hand, the goal probability of any state, for which not even a determinized plan exists, is known exactly: 0. But, on the contrary, if a state does have a determinized plan, then we merely know that the state can reach the goal via *some* selection of probabilistic action outcomes, but we have no information for all the other action outcomes. Hence, the only safe upper bound in this case is the trivial one: 1. In Chapter 15, we explore dead-end detection as a means to *pruning*, reducing the MDP directly, instead of relying on the heuristic search's ability to efficiently exploit the dead-end bounds.

Recently, Trevizan et al. (2017b) have introduced a heuristic capable of admissibly estimating expected cost without determinization. The crux lies in the interpretation of expected cost as a weighted sum over the occupation measures of the probabilistic actions, i.e., the expected number of times an action needs to be executed until reaching the goal (cf. Section 11.1). Starting from the LP-formulation of expected-cost MDPs over these occupation measures, Trevizan et al. replaced the constraints that represent the optimal policies exactly by necessary properties that every policy must satisfy. Every optimal solution to the resulting LP translates into an admissible estimate of the expected cost, simply because every optimal policy's occupation measures must satisfy the constraints. The LP's constraints are generated directly from the planning task description by *syntactically projecting* the task onto each of its state variables, formulating constraints on action applications locally with respect to each of the variables. The occupation-measure heuristic relates closely to the framework of LP-based *operator-counting heuristics* from classical planning (Pommerening et al., 2014). Motivated by that relation, Trevizan et al. (2017b) further introduced the *regrouped operator-counting heuristic*  $h^{\text{roc}}$ .  $h^{\text{roc}}$  extends the operator-counting heuristic LP, defined over the MDP's determinization, with additional constraints that *regroup* operator counts, making sure that the executions of a probabilistic action's outcomes match the outcome probabilities. While  $h^{\text{roc}}$  is theoretically dominated by the occupation-measure heuristic, i.e., it can provably not provide closer bounds to  $V^*$ , the  $h^{\text{roc}}$  LP is however considerably smaller, leading to tremendous practical advantages.

In this chapter, we adapt Trevizan et al.'s (2017b) heuristics towards estimating goal probability. The necessary changes are in principle straightforward, and mostly pertain to exchanging the expected-cost minimization LP by the LP from Definition 4.2. Additionally to those changes, we provide a more gen-

eral definition of the occupation-measure heuristic, adding the support of syntactic projections onto arbitrary variable subsets, instead of only the singleton variable ones. Moreover, we define the *probabilistic operator-counting heuristic* directly based upon the probabilistic planning task description, avoiding the determinization step. On the one hand, this results in an LP much smaller than that underlying  $h^{\text{TOC}}$ , removing the need of representing each action outcome via an individual LP variable, while, at the same, making the regrouping constraints obsolete. On the other hand, it gives rise to a more natural characterization of probabilistic operator-counting heuristics, in terms of a generic family of LP-based heuristics, similarly to Pommerening et al.'s (2014) classical planning variant. Different elements from this family arise from combinations of different *probabilistic operator-counting constraints*, i.e., properties of optimal MDP policies formulated in terms of LP constraints over a set of LP variables that is commonly shared among all the operator-counting constraints. We identify concrete properties that these constraints must satisfy, in order that their combination is guaranteed to result in admissible goal probability estimates. We provide probabilistic operator-counting constraints based on the syntactic projections, a reinterpretation of the *state equation* (Bonet, 2013) for goal probability, and action landmarks (e.g., Hoffmann et al., 2004; Karpas and Domshlak, 2009). Finally, going beyond Trevizan et al.'s (2017b) admissibility proofs, we show that all presented heuristics are not only upper-bounds on goal probability, but also satisfy the more strict monotonicity property.

### 14.1. Goal-Probability Occupation-Measure Heuristic

The occupation-measure heuristic is centered around the syntactic projection of probabilistic planning tasks. Our definition is a straightforward extension of that given by Trevizan et al. (2017b) to arbitrary variable subsets:

**Definition 14.1** (Syntactic Projection). *Let  $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  be a probabilistic FDR task. Let  $\emptyset \subset X \subseteq \mathcal{V}$  be a non-empty subset of variables. The **syntactic projection of  $\Pi$  onto  $X$**  is the probabilistic FDR task  $\Pi|_X = \langle X, \mathcal{A}|_X, \mathcal{I}|_X, \mathcal{G}|_X \rangle$ , defined as follows:*

- *The projection of a (partial) variable assignment  $P$  onto  $X$  is given by  $P|_X = \{v \mapsto P[v] \mid v \in X \cap \text{vars}(P)\}$ . Hence, the initial state is  $\mathcal{I}|_X = \{v \mapsto \mathcal{I}[v] \mid v \in X\}$ . The goal is  $\mathcal{G}|_X = \{v \mapsto \mathcal{G}[v] \mid v \in X \cap \text{vars}(\mathcal{G})\}$ .*
- *The actions are given by  $\mathcal{A}|_X = \{a|_X \mid a \in \mathcal{A}\}$ , where for each  $a \in \mathcal{A}$ ,  $a|_X$  is defined as  $\text{pre}(a|_X) = \text{pre}(a)|_X$ , and  $\text{out}(a|_X) = \{o|_X \mid o \in \text{out}(a)\}$  with  $\text{prob}(o|_X) = \text{prob}(o)$  and  $\text{eff}(o|_X) = \text{eff}(o)|_X$ .*

In other words, the syntactic projection of  $\Pi$  onto some variable subset  $X$  simply throws away all parts of  $\Pi$  on variables not contained in  $X$ . We refer to the syntactic projections onto a single variables  $X = \{v\}$  as the **atomic projections** of  $\Pi$ . For the sake of readability, we abbreviate the state space  $\mathcal{M}^{\Pi|_X}$  induced by the syntactic projection  $\Pi|_X$  through  $\mathcal{M}^X = \langle \mathcal{S}^X, \mathcal{A}^X, \mathcal{P}^X, s_f^X, \mathcal{S}_*^X \rangle$ . Moreover, for atomic projections, we occasionally omit the set notation, e.g., writing  $\mathcal{M}^v$  to denote the state space of the syntactic project  $\Pi|_v$ , and we identify the states  $\mathcal{S}^v$  directly through the variable's values  $\mathcal{D}_v$ . Example 14.1 shows a basic example task, alongside some syntactic projections. The example will be reused throughout this chapter.

**Example 14.1.** *Consider the probabilistic FDR task  $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  with variables  $\mathcal{V} = \{p, q, r\}$ , domains  $\mathcal{D}_p = \mathcal{D}_q = \{1, 0\}$  and  $\mathcal{D}_r = \{0, 1, 2\}$ ; initial state  $\mathcal{I} = \{p \mapsto 1, q \mapsto 0, r \mapsto 0\}$ ; goal  $\mathcal{G} = \{p \mapsto 1, q \mapsto 1, r \mapsto 0\}$ ; and three actions  $\mathcal{A} = \{a_1, a_2, a_3\}$ , where*

- *$a_1$  has precondition  $\text{pre}(a_1) = \{r \mapsto 0\}$ , and two outcomes  $\text{out}(a_1) = \{o_{1,1}, o_{1,2}\}$ :*

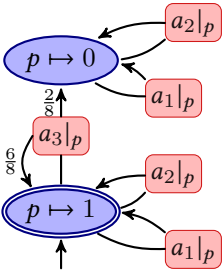
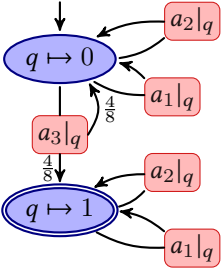
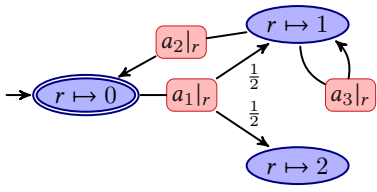
$\mathcal{M}^{\{p\}}$	
	$om_{p,1,a_1} \geq 0, om_{p,1,a_2} \geq 0, om_{p,1,a_3} \geq 0, \quad (14.1b)$ $om_{p,0,a_1} \geq 0, om_{p,0,a_2} \geq 0$ $-\frac{2}{8}om_{p,1,a_3} \leq 0 \quad (14.1c): \mathfrak{s} = \{p \mapsto 0\}$ $\frac{2}{8}om_{p,1,a_3} \leq 1 \quad (14.1c): \mathfrak{s} = \{p \mapsto 1\}$ $\frac{2}{8}om_{p,1,a_3} + v_{\mathcal{G}} \leq 1 \quad (14.1d)$
$\mathcal{M}^{\{q\}}$	
	$om_{q,1,a_1} \geq 0, om_{q,1,a_2} \geq 0, \quad (14.1b)$ $om_{q,0,a_1} \geq 0, om_{q,0,a_2} \geq 0, om_{q,0,a_3} \geq 0$ $\frac{4}{8}om_{q,0,a_3} \leq 1 \quad (14.1c): \mathfrak{s} = \{q \mapsto 0\}$ $-\frac{4}{8}om_{q,0,a_3} \leq 0 \quad (14.1c): \mathfrak{s} = \{q \mapsto 1\}$ $-\frac{4}{8}om_{q,0,a_3} + v_{\mathcal{G}} \leq 0 \quad (14.1d)$
$\mathcal{M}^{\{r\}}$	
	$om_{r,0,a_1} \geq 0, om_{r,1,a_2} \geq 0, om_{r,1,a_3} \geq 0 \quad (14.1b)$ $om_{r,0,a_1} - om_{r,1,a_2} \leq 1 \quad (14.1c): \mathfrak{s} = \{r \mapsto 0\}$ $om_{r,1,a_2} - \frac{1}{2}om_{r,0,a_1} \leq 0 \quad (14.1c): \mathfrak{s} = \{r \mapsto 1\}$ $-\frac{1}{2}om_{r,0,a_1} \leq 0 \quad (14.1c): \mathfrak{s} = \{r \mapsto 2\}$ $om_{r,0,a_1} - om_{r,1,a_2} + v_{\mathcal{G}} \leq 1 \quad (14.1d)$
Tying constraints (14.1e), choosing $X_0 = \{r\}$ .	
$om_{p,0,a_1} + om_{p,1,a_1} = om_{r,0,a_1}$ $om_{p,0,a_2} + om_{p,1,a_2} = om_{r,1,a_2}$ $om_{p,1,a_3} = om_{r,1,a_3}$	$om_{q,0,a_1} + om_{q,1,a_1} = om_{r,0,a_1}$ $om_{q,0,a_2} + om_{q,1,a_2} = om_{r,1,a_2}$ $om_{q,0,a_3} = om_{r,1,a_3}$

Figure 14.1.: Left: state spaces of the syntactic projections from Example 14.1. Right: the corresponding instantiations of the  $H^{\text{gpm}}$  constraints.

- $\text{eff}(o_{1,1}) = \{r \mapsto 1\}$  and  $\text{prob}(o_{1,1}) = \frac{1}{2}$ ,
- $\text{eff}(o_{1,2}) = \{r \mapsto 2\}$  and  $\text{prob}(o_{1,2}) = \frac{1}{2}$ .
- $a_2$  has precondition  $\text{pre}(a_2) = \{r \mapsto 1\}$  and a single outcome with effect  $\text{eff}(o) = \{r \mapsto 0\}$ .
- $a_3$  has precondition  $\text{pre}(a_3) = \{q \mapsto 0, r \mapsto 1\}$ , and four outcomes  $\text{out}(a_3) = \{o_{3,1}, o_{3,2}, o_{3,3}, o_{3,4}\}$ :

- $\text{eff}(o_{3,1}) = \emptyset$  and  $\text{prob}(o_{3,1}) = \frac{3}{8}$ ,
- $\text{eff}(o_{3,2}) = \{p \mapsto 0\}$  and  $\text{prob}(o_{3,2}) = \frac{1}{8}$ ,
- $\text{eff}(o_{3,3}) = \{q \mapsto 1\}$  and  $\text{prob}(o_{3,3}) = \frac{3}{8}$ ,
- $\text{eff}(o_{3,4}) = \{p \mapsto 0, q \mapsto 1\}$  and  $\text{prob}(o_{3,4}) = \frac{1}{8}$ .

The syntactic projection of  $\Pi$  onto  $r$  has initial state  $\mathcal{I}|_r = \{r \mapsto 0\}$ ; goal  $\mathcal{G}|_r = \{r \mapsto 0\}$ ; and the actions  $a_1|_r = a_1$ ,  $a_2|_r = a_2$ , and  $a_3|_r$  with  $\text{pre}(a_3|_r) = \{r \mapsto 1\}$ , and all its outcomes have an empty effect. The left-hand side of Figure 14.1 depicts the state spaces of the atomic projections  $\Pi|_p$ ,  $\Pi|_q$ , and  $\Pi|_r$ .

Consider any syntactic projection  $\Pi|_X$ , and any optimal policy  $\pi$  for the original task  $\Pi$ . Let  $LP(\Pi|_X)$  be the dual MaxProb LP encoding (Definition 11.2) for  $\Pi|_X$ , with occupation-measure variables  $om_{X,s,a}$  for each  $s \in \mathcal{S}^X$  and  $a|_X \in \mathcal{A}^X$ . Notice that every execution  $s_0, \pi(s_0), s_1, \pi(s_1), \dots$  of  $\pi$  in  $\Pi$  induces a corresponding path  $s_0|_X, \pi(s_0)|_X, s_1|_X, \pi(s_1)|_X, \dots$  for  $\Pi|_X$ . Hence, given that  $\pi$ 's occupation measures  $om_{s,a}^\pi$  satisfy  $LP(\Pi)$  by definition, it is easy to show that setting each  $om_{X,s,a}$  to the sum of  $om_{s,a}^\pi$  over all states  $s$  with  $s|_X = s$  yields a feasible solution to  $LP(\Pi|_X)$ . Moreover, note that the objective value of this solution cannot be less than the value of the policy in  $\Pi$ , since every execution of  $\pi$  that ends in a goal state of  $\Pi$ , necessarily also ends in a goal state of  $\Pi|_X$ . As this applies to all syntactic projections alike, we can analogously construct from  $\pi$  a feasible solution to the conjunction  $\bigwedge_{X \in \mathcal{X}} LP(\Pi|_X)$  of the LP constraints, for any set of variable-subsets  $X$ , while the minimum over all the LP's objective functions is an upper bound on the policy's goal-probability value. Although the occupation measures  $om_{X,s,a}$  and  $om_{X',t,a}$ , for any action  $a$  and individual states  $s \in \mathcal{S}^X, t \in \mathcal{S}^{X'}$ , can in general not be put in direct relation, observe that, as per the construction, the chosen occupation measures of two syntactic projections agree on the total mass associated with  $a$ . Namely,  $\sum_{s \in \mathcal{S}^X} om_{X,s,a} = \sum_{t \in \mathcal{S}^{X'}} om_{X',t,a} = \sum_{s \in \mathcal{S}} om_{s,a}^\pi$ . By making this relation explicit in forms of additional constraints for each  $a$ , one can tie together the LPs of the individual syntactic projections. In summary, this results in the following construction:

**Definition 14.2** (Goal-Probability Projection Occupation-Measure Heuristic). Let  $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  be a probabilistic FDR task. Let  $\mathcal{X}$  be a set of non-empty variable subsets. Let  $\hat{s}$  be a state of  $\Pi$ . The **goal-probability projection occupation-measure heuristic**  $H_X^{\text{gpm}}(\hat{s})$  over  $\mathcal{X}$  assigns to  $\hat{s}$  the optimal objective value of the following LP:

$$\begin{array}{ll} \underset{om, v_{\mathcal{G}}}{\text{maximize}} & v_{\mathcal{G}} \end{array} \quad (14.1a)$$

$$\begin{array}{ll} \text{subject to} & om_{X,s,a} \geq 0 \quad X \in \mathcal{X}, s \in \mathcal{S}^X, \\ & a|_X \in \mathcal{A}^X(s), \end{array} \quad (14.1b)$$

$$out_X(s) - in_X(s) \leq [\mathcal{G}|_X \subseteq \hat{s}] \quad X \in \mathcal{X}, s \in \mathcal{S}^X, \quad (14.1c)$$

$$\sum_{s \in \mathcal{S}_*^X} (out_X(s) - in_X(s)) + v_{\mathcal{G}} \leq [\mathcal{G}|_X \subseteq \hat{s}] \quad X \in \mathcal{X}, \quad (14.1d)$$

$$om_X(a) - om_{X_0}(a) = 0 \quad X \in \mathcal{X}, a \in \mathcal{A} \quad (14.1e)$$

where  $X_0 \in \mathcal{X}$  can be chosen arbitrarily, and we use the following shorthand:

$$\begin{aligned} out_X(s) &= \sum_{a|_X \in \mathcal{A}^X(s)} om_{X,s,a} \\ in_X(s) &= \sum_{a \in \mathcal{A}} \sum_{t \in \mathcal{S}^X} \mathcal{P}^X(t, a|_X, s) om_{X,t,a} \\ om_X(a) &= \sum_{s \in \mathcal{S}^X: a|_X \in \mathcal{A}^X(s)} om_{X,s,a} \end{aligned}$$

In a nutshell, the  $H_X^{\text{gpm}}$  LP glues together the dual MaxProb LPs (Definition 11.2) of the individual syntactic projections, with the usual occupation-measure variables (14.1b), and the flow constraints (14.1c). The tying constraints (14.1e) ensure that every action is executed overall the same number of times in each syntactic projection. The additional variable  $v_G$  represents the minimum over the individual objective values, whose value shall be maximized just like in the objective of dual MaxProb LP. The *goal constraints* (14.1d) ensure that this interpretation is correct, enforcing  $v_G$  to not exceed the probability mass that sinks at the goal states of any  $\Pi|_X$ . Importantly, note that as opposed to the LP from Definition 11.2, Definition 14.2 allows the execution of actions in goal states  $\mathbf{s} \in \mathcal{S}_*^X$ . This is necessary because, as per the projection operation, being in a goal state in one of the syntactic projections does not necessarily mean that we have already reached a goal state in all the considered projections. However, when strictly interpreting goal states as sink states as in Definition 11.2, it would also not be possible any longer to apply additional actions in the other projections, given the tying constraints. Example 14.2 details the complete  $H_X^{\text{gpm}}$  LP for the planning task from Example 14.1.

**Example 14.2.** Consider again the planning task and the syntactic projections from Example 14.1. The right-hand side of Figure 14.1 depicts the constraints of  $H_X^{\text{gpm}}(s_I)$  over the atomic projections. Notice that the pure self-loop transitions cancel out in the flow constraints. Moreover, notice that  $om_{p,1,a_3} = om_{r,1,a_3} > 0$  is needed in order to generate  $q$ 's goal value, and thus achieve a goal-probability value  $v_G > 0$ . The optimal solutions to the LP lie in the intersection of the two straight lines:  $1 - \frac{2}{8}om_{p,1,a_3} = \frac{4}{8}om_{q,0,a_3}$ , where  $om_{q,0,a_3} = om_{p,1,a_3}$  due to the tying constraints. Solving the equation results in  $om_{p,1,a_3} = om_{q,0,a_3} = om_{r,1,a_3} = \frac{4}{3}$  and  $v_G = \frac{2}{3}$ . Consequently, the heuristic estimate of  $s_I$  is  $H_X^{\text{gpm}}(s_I) = v_G = \frac{2}{3} \geq \frac{1}{3} = V^*(s_I)$ .

**Theorem 14.1.**  $H_X^{\text{gpm}}$  is a monotone upper bound for  $\mathcal{M}^\Pi$ .

*Proof sketch.* To show that  $H_X^{\text{gpm}}$  is a monotone upper bound, it suffices to show that it holds  $H_X^{\text{gpm}}(s) \geq (BH_X^{\text{gpm}})(s)$  for all states  $s \in \mathcal{S}^\Pi$ . Due to the monotonicity invariance (Theorem 11.2), and the convergence property as per Theorem 11.3, the limit  $\lim_{k \rightarrow \infty} B^{(k)}H_X^{\text{gpm}} = V^{(\infty)}$  exists, and it holds  $H_X^{\text{gpm}} \geq V^{(\infty)}$ . Since  $V^*$  is by definition the piecewise smallest fixed point of  $B$ , i.e.,  $V^{(\infty)} \geq V^*$ , it transitively follows that  $H_X^{\text{gpm}} \geq V^*$ .

Suppose  $\hat{s} \in (\mathcal{S}_\perp^\Pi \setminus \mathcal{S}_*^\Pi)$  is some non-goal terminal state. Then  $(BH_X^{\text{gpm}})(\hat{s}) = 0$ , and  $H_X^{\text{gpm}}(\hat{s}) \geq 0$  is trivially satisfied (choosing 0 for all LP variables always satisfies the constraints).

Suppose  $\hat{s} \in \mathcal{S}_*^\Pi$  is a goal state. Note that  $v_G = 1$  and  $om_{X,t,a} = 0$  satisfies all constraints (14.1b) – (14.1e). Hence,  $H_X^{\text{gpm}}(\hat{s}) \geq 1 = (BH_X^{\text{gpm}})(\hat{s})$ .

Finally, suppose  $\hat{s} \in (\mathcal{S}^\Pi \setminus \mathcal{S}_\perp^\Pi \setminus \mathcal{S}_*^\Pi)$  is some non-terminal, non-goal state. We show that

$$H_X^{\text{gpm}}(\hat{s}) \geq \sum_{t \in \text{Succ}(\hat{s}, a)} \mathcal{P}(\hat{s}, a, t) H_X^{\text{gpm}}(t)$$

holds for every action applicable in  $\hat{s}$ , and hence  $\geq$  also holds when taking the maximum over all those actions. Let  $\hat{a} \in \mathcal{A}(\hat{s})$  be arbitrary, and for each  $t \in \text{Succ}(\hat{s}, \hat{a})$ , let  $om^t$  denote an optimal solution to the  $H_X^{\text{gpm}}(t)$  LP. Consider

$$\begin{aligned} v_G &:= \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) v_G^t \\ om_{X,\mathbf{s},a} &:= \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) om_{X,\mathbf{s},a}^t + [\mathbf{s} \subseteq \hat{s} \text{ and } a = \hat{a}|_X] \end{aligned}$$

We next show that  $om$  satisfies the  $H_X^{\text{gpm}}(\hat{s})$  constraints. The range constraint (14.1b) is trivially satisfied. The tying constraint is satisfied because we changed the overall occupation measure only for  $\hat{a}$ , and the change for  $\hat{a}$  is the same for all  $X \in \mathcal{X}$ . Namely, denoting by  $\mathcal{S}^X(a) = \{s \in \mathcal{S}^X \mid a|_X \in \mathcal{A}^X(s)\}$  the set of projection states in which  $a|_X$  is applicable, it holds that

$$\begin{aligned}
om_X(a) &= \sum_{s \in \mathcal{S}^X(a)} om_{X,s,a} \\
&= \sum_{s \in \mathcal{S}^X(a)} \left( \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) om_{X,s,a}^t + [s \subseteq \hat{s} \text{ and } a = \hat{a}] \right) && \text{(def. of } om) \\
&= \sum_{s \in \mathcal{S}^X(a)} \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) om_{X,s,a}^t + [a = \hat{a}] && (s \subseteq \hat{s} \text{ exactly if } s = \hat{s}|_X) \\
&= \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) \sum_{s \in \mathcal{S}^X(a)} om_{X,s,a}^t + [a = \hat{a}] && \text{(commutat. and distributivity)} \\
&= \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) om_X^t(a) + [a = \hat{a}] && \text{(def. of } om_X \text{ for } t) \\
&= \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) om_{X_0}^t(a) + [a = \hat{a}] && (om^t \text{ satisfies (14.1e)}) \\
&= om_{X_0}(a) && \text{(invert the steps above for } X_0)
\end{aligned}$$

Showing that the chosen values for  $om$  also satisfies (14.1d) and (14.1c) amounts to equation transformations, similar to those above, together with exploiting the fact that  $om^t$  satisfies the respective constraints for the successor state  $t$ . We spare the reader the tedious details, and illustrate only the main argument. The detailed transformations are available in Appendix C.3.1. The key observation is that the difference between “in-flow” and “out-flow” of any state  $s \in \mathcal{S}^X$  over  $om$  can be rewritten into (cf. Equation (C.3) in the appendix):

$$out_X(s) - in_X(s) = \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) (out_X^t(s) - in_X^t(s)) + [s \subseteq \hat{s}] - \mathcal{P}^X(\hat{s}|_X, \hat{a}|_X, s)$$

In words, it is the expected flow difference over the successor states, plus additional factors taking into account the flow inserted for  $\hat{s}$ , and the additional execution of  $\hat{a}|_X$  in  $\hat{s}|_X$ , as per the construction of  $om$ . Consider the flow constraint (14.1c). The additional  $[s \subseteq \hat{s}]$  term cancels out with the inequality’s right-hand side. Given that the flow constraint is satisfied for all successors  $t$ , it is left to show that:

$$\sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) [s \subseteq t] \leq \mathcal{P}^X(\hat{s}|_X, \hat{a}|_X, s)$$

i.e., that the probability of transitioning from  $\hat{s}|_X$  into  $s$  in the state space of the syntactic projection is not less than the probability of transitioning from  $\hat{s}$  into any  $t$  with  $t|_X = s$ . This can be shown by plugging in

the syntactic definitions:

$$\begin{aligned}
\sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) [\mathfrak{s} \subseteq t] &= \sum_{o \in \text{out}(\hat{a})} \text{prob}(o) [\mathfrak{s} \subseteq \hat{s} \llbracket o \rrbracket] && \text{(def. of state space)} \\
&= \sum_{o \in \text{out}(\hat{a})} \text{prob}(o) [\mathfrak{s} = \hat{s} \llbracket o \rrbracket|_X] && \text{(def. of states in } \mathcal{S}^X) \\
&= \sum_{o \in \text{out}(\hat{a})} \text{prob}(o) [\mathfrak{s} = \hat{s} \llbracket o|_X \rrbracket|_X] && \begin{array}{l} \text{(effects of } o \text{ on variables outside } X \\ \text{are projected away afterwards any-} \\ \text{ways)} \end{array} \\
&= \sum_{o \in \text{out}(\hat{a})} \text{prob}(o) [\mathfrak{s} = \hat{s}|_X \llbracket o|_X \rrbracket] && \text{(does not matter whether we project} \\
& && \text{before or after the application of } o|_X) \\
&= \mathcal{P}^X(\hat{s}|_X, \hat{a}|_X, \mathfrak{s}) && \text{(def. of } \mathcal{P}^X \text{ in the state space } \mathcal{M}^X \text{ of} \\
& && \text{the syntactic projection)}
\end{aligned}$$

This completes the proof of (14.1c). That the goal constraint (14.1d) is satisfied can be shown analogously. Appendix C.3.1 spells out the details. □

## 14.2. Probabilistic Operator-Counting Heuristic

The size of the projection occupation-measure heuristic LP is polynomial in the sizes of the considered syntactic projections. Consider specifically the atomic projections. Notice that the  $H^{\text{gpm}}$  LP over these projections consists of variables and constraints in the order of  $\sum_{v \in \mathcal{V}} |\mathcal{D}_v| \cdot |\mathcal{A}|$ . Trevizan et al. (2017b) observed in their experiments that even in this smallest possible instantiation, the occupation-measure heuristic is often already prohibitively expensive. The regrouped operator-counting heuristic  $h^{\text{roc}}$  avoids the introduction of LP variables for each action fact pair by leveraging the *state equation* from classical planning (Bonet, 2013; Pommerening et al., 2014) (see also Chapter 7). In a nutshell, the state equation formulates constraints directly on the action outcome occurrences, based on viewing facts as tokens that the outcomes produce and consume. This results in an LP with no more than  $\sum_{a \in \mathcal{A}} |\text{out}(a)|$  LP variables overall, and a number of constraints comparable to the occupation-measure heuristic LP.  $h^{\text{roc}}$  extends the state equation LP by additional  $\sum_{a \in \mathcal{A}} |\text{out}(a)|^2$  *regrouping constraints* to enforce that the outcome occurrences of every action are chosen proportionally to the outcome probability distribution. In this section, we take Trevizan et al.’s (2017b) idea one step further. Following Pommerening et al.’s (2014) operator-counting heuristics from classical planning, we introduce *probabilistic operator-counting heuristics* as an entire framework of heuristics, postulating constraints on the execution counts of probabilistic actions directly, instead of their outcomes. To instantiate the framework, we in particular reinterpret the state equation in the context of probabilistic actions. As a result, we obtain a new state equation heuristic variant, using only  $|\mathcal{A}|$  LP variables, and that comes without the need of any additional regrouping constraints.

The basic building block of our heuristic framework are *probabilistic operator-counting constraints*:

**Definition 14.3** (Goal-Probability Operator-Counting Constraint). *Let  $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  be a probabilistic FDR task. Let  $\mathcal{Y}$  be a set of real variables, which at least includes for each action  $a \in \mathcal{A}$  a non-negative **probabilistic operator-counting variable**  $y_a$ , and a non-negative goal-probability variable  $v_{\mathcal{G}}$ . Let  $\Gamma$  be a function that maps each state  $s$  to a set of linear inequalities over  $\mathcal{Y}$ . We call  $\Gamma$  **feasible**, if all states  $s \in \mathcal{S}^\Pi$  have a feasible solution  $y$  to  $\Gamma(s)$ , where  $v_{\mathcal{G}} = y_a = 0$ , for all  $a \in \mathcal{A}$ . We say that  $\Gamma$  is **goal-respecting**, if every goal state  $s_* \in \mathcal{S}_*^\Pi$  has a feasible solution  $y$  to  $\Gamma(s_*)$  with  $v_{\mathcal{G}} = 1$  and  $y_a = 0$ , for all  $a \in \mathcal{A}$ . We say*

that  $\Gamma$  is **consistent**, if for all non-goal states  $\hat{s} \in \mathcal{S}^\Pi \setminus \mathcal{S}_*^\Pi$ , all applicable actions  $\hat{a} \in \mathcal{A}(\hat{s})$ , and all feasible solutions  $y^t, v_{\mathcal{G}}^t$  to  $\Gamma(t)$  of all successors  $t \in \text{Succ}(\hat{s}, \hat{a})$ , there is a feasible solution  $y, v_{\mathcal{G}}$  to  $\Gamma(\hat{s})$  such that (i)  $v_{\mathcal{G}} = \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) v_{\mathcal{G}}^t$ , and (ii) for all  $a \in \mathcal{A}$ ,  $y_a = \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) y_a^t + \mathbb{1}[a = \hat{a}]$ . If  $\Gamma$  is feasible, goal-respecting, and consistent, then we call  $\Gamma$  a **goal-probability operator-counting constraint** for  $\Pi$ .

The definition of goal-probability operator-counting constraints makes sure that, for each state  $s$  and every policy  $\pi$ ,  $\pi$  defines a feasible solution to  $\Gamma(s)$  with  $v_{\mathcal{G}} \geq V^\pi(s)$ . Notice, however, that the opposite direction does generally not hold. For instance, consider an MDP with states  $s_0, s_1$ , a terminal dead-end state  $s_\perp$ , and a goal state  $s_*$ . Say that there is a deterministic transition  $s_0 \xrightarrow{a} s_1$ , and a probabilistic transition from  $s_1$  via  $a$  going into  $s_*$  with probability  $\alpha$  and into  $s_\perp$  with probability  $1 - \alpha$ . Suppose the constraint function  $\Gamma_{\hat{z}}$  assigns  $\Gamma_{\hat{z}}(s_0) = \{v_{\mathcal{G}} \leq \alpha\}$ , and  $\Gamma_{\hat{z}}(s) = \emptyset$  for the other states. Obviously, for every policy, we can find feasible solutions to  $\Gamma_{\hat{z}}(s_0)$  and  $\Gamma_{\hat{z}}(s_1)$ , where  $v_{\mathcal{G}}$  matches the respective policy value. Still,  $\Gamma_{\hat{z}}$  is not a goal-probability operator-counting constraint, as the consistency property is violated:  $v_{\mathcal{G}}^{s_1} = 1$  and  $y_a^{s_1} = 0$  is a feasible solution to  $\Gamma_{\hat{z}}(s_1)$ , yet  $v_{\mathcal{G}} = v_{\mathcal{G}}^{s_1} = 1$  and  $y_a = 1$  does not satisfy  $\Gamma_{\hat{z}}(s_0)$ . As we shall see below, the stricter constraint characterization is needed for the monotonicity property.

**Definition 14.4** (Goal-Probability Operator-Counting Heuristic). Let  $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  be a probabilistic FDR task. Let  $\Gamma$  be a goal-probability operator-counting constraint over variables  $\mathcal{Y}$ . The **goal-probability operator-counting heuristic**  $H_\Gamma^{\text{gpoc}}$  for  $\Gamma$  assigns to each state  $\hat{s}$  the optimal objective value of the following LP

$$\begin{aligned} & \text{maximize} && v_{\mathcal{G}} \\ & && \mathcal{Y}, v_{\mathcal{G}} \\ & \text{subject to} && 0 \leq v_{\mathcal{G}} \leq 1, \\ & && y_a \geq 0 \quad a \in \mathcal{A}, \\ & && \mathcal{Y} \models \gamma \quad \gamma \in \Gamma(\hat{s}) \end{aligned}$$

Consider the example from before. It holds that  $H_{\Gamma_{\hat{z}}}^{\text{gpoc}}(s_0) = \alpha$ , and  $H_{\Gamma_{\hat{z}}}^{\text{gpoc}}(s) = 1$  for the other three states, i.e.  $H_{\Gamma_{\hat{z}}}^{\text{gpoc}}$  is actually an upper bound on  $V^*$ . However,  $H_{\Gamma_{\hat{z}}}^{\text{gpoc}}$  is not monotone, since  $(BH_{\Gamma_{\hat{z}}}^{\text{gpoc}})(s_0) = H_{\Gamma_{\hat{z}}}^{\text{gpoc}}(s_1) = 1 > \alpha = H_{\Gamma_{\hat{z}}}^{\text{gpoc}}(s_0)$ . The consistency property guarantees that this is the case:

**Theorem 14.2.** Let  $\Gamma$  be a goal-probability operator-counting constraint. Then,  $H_\Gamma^{\text{gpoc}}$  is a monotone upper bound for  $\mathcal{M}^\Pi$ .

*Proof.* First, notice that  $H_\Gamma^{\text{gpoc}}(s)$  is properly defined for all states  $s \in \mathcal{S}^\Pi$ : since  $\Gamma$  is feasible, the  $H_\Gamma^{\text{gpoc}}(s)$  LP has a feasible solution; and since  $v_{\mathcal{G}}$  is bounded from above, there also exists an optimal solution. Similar to the proof of Theorem 14.1, it suffices to show that  $H_\Gamma^{\text{gpoc}} \geq BH_\Gamma^{\text{gpoc}}$ . We omit the  $\Gamma$  subscript for the sake of readability. Let  $s_* \in \mathcal{S}_*^\Pi$  be any goal state. Since  $\Gamma$  is goal-respecting, it directly follows that  $H^{\text{gpoc}}(s_*) \geq 1 = (BH^{\text{gpoc}})(s_*)$ . Let  $s$  be any non-goal state. Since  $\Gamma$  is feasible, it holds that  $H^{\text{gpoc}}(s) \geq 0$ . This already completes the proof for terminal states. Suppose that  $s$  is not terminal, and let  $a \in \mathcal{A}(s)$  be any action applicable in  $s$ . As per the consistency property, there exists a feasible solution to  $\Gamma(s)$ , whose objective value is  $\sum_{t \in \text{Succ}(s, a)} \mathcal{P}(s, a, t) H^{\text{gpoc}}(t)$ , i.e.,  $H^{\text{gpoc}}(s) \geq (QH^{\text{gpoc}})(s, a)$ . As this in particular includes the action with maximal  $Q$ -value, it thus follows that  $H^{\text{gpoc}}(s) \geq (BH^{\text{gpoc}})(s)$ .  $\square$

The power of the classical-planning operator-counting framework lies in its ability to form admissible heuristics by glewing together operator-counting constraints generated from different sources. The con-

ditions in our probabilistic version make sure that this is still possible. On this matter, notice that for the previous proof, it would be sufficient if the consistency conditions were satisfied for just a single selection of optimal solutions  $y^t$  of the successor states  $t \in \text{Succ}(\hat{s}, \hat{a})$ . However, when combining multiple goal-probability operator-counting constraints, one would then have to argue whether all the constraints satisfy the consistency condition on the *same* set of successor solutions  $y^t$ . Definition 14.3 guarantees that the combination is safe without making further assumptions:

**Proposition 14.1.** *Let  $\Gamma_1$  and  $\Gamma_2$  be two goal-probability operator-counting constraints over real variables  $\mathcal{Y}_1$ , respectively  $\mathcal{Y}_2$ . Suppose that the operator-counting and the goal-probability variables are the only variables that commonly appear in both  $\mathcal{Y}_1$  and  $\mathcal{Y}_2$ . Let  $\Gamma(s) = \Gamma_1(s) \cup \Gamma_2(s)$ , for all  $s \in \mathcal{S}^\Pi$ . Then  $\Gamma$  is a goal-probability operator-counting constraint.*

*Proof.* Since  $\Gamma_1$  and  $\Gamma_2$  are feasible and goal-respecting, it directly follows that  $\Gamma$  is feasible and goal-respecting. To show that  $\Gamma$  is consistent, let  $\hat{s}$  be any non-goal state,  $\hat{a} \in \mathcal{A}(\hat{s})$  be any applicable action, and for each successor  $t \in \text{Succ}(\hat{s}, \hat{a})$ , let  $y^t$  be any feasible solution to  $\Gamma(t)$ . Note that, as per the selection,  $y^t$  is a feasible solution to  $\Gamma_1(t)$ , as well as a feasible solution to  $\Gamma_2(t)$ . Since  $\Gamma_1$  and  $\Gamma_2$  are consistent, there must exist a feasible solution  $y_1, v_{1\mathcal{G}}$  to  $\Gamma_1(\hat{s})$ , and a feasible solution  $y_2, v_{2\mathcal{G}}$  to  $\Gamma_2(\hat{s})$  such that: (i)  $v_{1\mathcal{G}} = v_{2\mathcal{G}} = \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) v_{\mathcal{G}}^t$ , and similarly (ii) for all actions  $a \in \mathcal{A}$ :  $y_{1a} = y_{2a} = \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) y_a^t + [a = \hat{a}]$ . Given that these are the only variables that commonly appear in  $\mathcal{Y}_1$  and  $\mathcal{Y}_2$ , it hence follows that letting each  $x \in \mathcal{Y}_1 \cup \mathcal{Y}_2$  be

$$x := \begin{cases} \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) v_{\mathcal{G}}^t, & \text{if } x = v_{\mathcal{G}} \\ \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) y_a^t + [a = \hat{a}], & \text{if } x = y_a \text{ for some } a \in \mathcal{A} \\ y_1[x], & \text{if } x \in \mathcal{Y}_1 \\ y_2[x], & \text{if } x \in \mathcal{Y}_2 \end{cases}$$

where  $y_i[x]$  denotes the value of  $x$  in the solution  $y_i$ ; yields a feasible solution to both  $\Gamma_1(\hat{s})$  and  $\Gamma_2(\hat{s})$ , i.e.,  $x$  is a feasible solution to  $\Gamma(\hat{s})$ , and it satisfies the consistency conditions by construction.  $\square$

Finally, notice that adding more constraints to  $\Gamma$  can only be beneficial in terms of the heuristic estimates:

**Proposition 14.2.** *Let  $\Gamma$  and  $\Gamma'$  be two goal-probability operator-counting constraints for  $\Pi$ . If it holds  $\Gamma(s) \subseteq \Gamma'(s)$  for all states  $s \in \mathcal{S}^\Pi$ , then  $H_\Gamma^{\text{g poc}} \geq H_{\Gamma'}^{\text{g poc}}$ .*

The remainder of this section presents three methods for generating goal-probability operator-counting constraints.

### Projection Occupation-Measure Constraints

The family of goal-probability operator-counting heuristics can be straightforwardly linked back to the projection occupation-measure heuristic:

**Corollary 14.1.** *Let  $\mathcal{X}$  be a set of non-empty variable subsets. Suppose the constraint function  $\Gamma_{\mathcal{X}}^{\text{g pom}}(s)$  returns the set of  $H_{\mathcal{X}}^{\text{g pom}}(s)$  constraints (14.1b) – (14.1e), together with*

$$y_a = \text{om}_{X_0}(a) \quad a \in \mathcal{A} \quad (14.2)$$

for an arbitrary  $X_0 \in \mathcal{X}$ . Then  $\Gamma_{\mathcal{X}}^{\text{g pom}}$  is a goal-probability operator counting constraint. Moreover, it holds that  $H_{\Gamma_{\mathcal{X}}^{\text{g pom}}}^{\text{g poc}} = H_{\mathcal{X}}^{\text{g pom}}$ .

That  $\Gamma_{\chi}^{\text{gpm}}$  is feasible and goal-respecting follows immediately from the constraint definitions. That  $\Gamma_{\chi}^{\text{gpm}}$  is consistent can be shown via the same arguments as in the monotonicity proof of Theorem 14.1. Obviously,  $\Gamma_{\chi}^{\text{gpm}}$  by itself is not very useful. Instead of using it inside the operator-counting framework, one could simply use  $H^{\text{gpm}}$  directly, and by that also get rid of the additional operator counting variables. However, this interpretation gives rise to additional enhancements of the projection occupation-measure heuristic, via the combination with other operator-counting constraints, as per Proposition 14.2.

### Goal-Probability State Equation

Recall from Chapter 7 that the state equation describes a relation between fact achievement and deletion counts that holds true in every plan. Obviously, during the execution of any action sequence, no fact may ever be deleted more often than it is achieved. Moreover, the total difference between achievements and deletions of any fact cannot be larger than 1, since in between every two deletions, the fact has to be re-achieved, and vice versa for the achievements. Since goal facts must be satisfied at the end of every plan, they must be achieved more often than deleted, i.e., their achievement-deletion difference has to be exactly 1. These relations can be postulated as operator-counting constraints by simply partitioning for each fact  $v \mapsto d$  the set of actions into ones that achieve  $v \mapsto d$ , the *producers*, and ones that delete  $v \mapsto d$ , the *consumers*, and bounding the difference between their operator counts accordingly.

Lifting the classical state equation to the probabilistic setting amounts to the following changes. In the presence of probabilistic actions, not every execution of an action may delete or achieve a fact, as this now depends on the action's probabilistic outcomes. Hence, instead of categorizing deterministic actions into producers and consumers, we attach to each probabilistic action the probability that it produces and consumes some fact, weighting the probabilistic operator counts accordingly. Secondly, as in our setting the goal may not be reachable with certainty, the production-consumption difference of goal facts can no longer be bounded by 1. We replace the bound by the goal-probability variable  $v_{\mathcal{G}}$ .

**Definition 14.5** (Goal-Probability State-Equation Constraint). *Let  $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  be a probabilistic FDR task. Let  $s$  be a state of  $\Pi$ . The **goal-probability state-equation constraint**  $\Gamma^{\text{seq}}(s)$  for  $s$  is*

$$\sum_{a \in \mathcal{A}} (\text{cons}_{v \mapsto d}^a - \text{prod}_{v \mapsto d}^a) y_a + [\mathcal{G}[v] = d] v_{\mathcal{G}} \leq [s[v] = d] \quad v \in \mathcal{V}, d \in \mathcal{D}_v, \quad (14.3)$$

using the following probability weights:

$$\text{prod}_{v \mapsto d}^a = \begin{cases} \sum_{\substack{o \in \text{out}(a): \\ \text{eff}(o)[v]=d}} \text{prob}(o), & \text{if } \text{pre}[v] \neq d \\ 0, & \text{otherwise} \end{cases}$$

$$\text{cons}_{v \mapsto d}^a = \begin{cases} \sum_{\substack{o \in \text{out}(a): \\ \text{eff}(o)[v]=d' \neq d}} \text{prob}(o), & \text{if } \text{pre}[v] = d \\ 0, & \text{otherwise} \end{cases}$$

Intuitively,  $\sum_{a \in \mathcal{A}} (\text{cons}_{v \mapsto d}^a - \text{prod}_{v \mapsto d}^a) y_a$  represents the difference between total expected consumption and the total expected production of  $v \mapsto d$ . For a non-goal fact, (14.3) therefore requires that the expected consumption does not exceed the expected production, counting plus 1 on the production side, if  $v \mapsto d$  is true in the state. The leftovers of the expected production after consumption of a fact can be interpreted as a bound on the probability that the fact holds after the execution of any policy that fits the operator counts. With this interpretation, the goal probability of any such policy is bounded by the minimum leftover over all the goal facts. Notice that this is exactly what the additional  $v_{\mathcal{G}}$  term in (14.3) for goal fact does.

**Theorem 14.3.**  $\Gamma^{\text{seq}}$  constitutes a goal-probability operator-counting constraint.

*Proof.* Since the right-hand side of (14.3) is non-negative for all facts and states, assigning all operator-counting and the goal-achiever variables to 0 trivially satisfies those constraints. Hence,  $\Gamma^{\text{seq}}$  is feasible. Let  $s_* \in \mathcal{S}_*^\Pi$  be any goal state. Note that  $v_{\mathcal{G}} = 1$  and  $y_a = 0$ , for all  $a \in \mathcal{A}$ , satisfies  $\Gamma^{\text{seq}}(s_*)$ , since for all  $v$  and  $d$ ,  $\mathcal{G}[v] = d$  implies  $s_*[v] = d$ , i.e.,  $[\mathcal{G}[v] = d] \leq [s_*[v] = d]$ . Thus,  $\Gamma^{\text{seq}}$  is goal-respecting. To show that  $\Gamma^{\text{seq}}$  is consistent, let  $\hat{s}$  be any non-goal state,  $\hat{a} \in \mathcal{A}(\hat{s})$  be any applicable action, and  $y^t, v_{\mathcal{G}}^t$  be any feasible solution to  $\Gamma^{\text{seq}}(t)$ , for each  $t \in \text{Succ}(\hat{s}, \hat{a})$ . Let  $v_{\mathcal{G}} = \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) v_{\mathcal{G}}^t$ , and  $y_a = \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) y_a^t + [a = \hat{a}]$ , for all  $a \in \mathcal{A}$ , as in Definition 14.5. Furthermore, let  $\delta_{v \mapsto d}^a = \text{cons}_{v \mapsto d}^a - \text{prod}_{v \mapsto d}^a$  be the weight associated with  $a$  in (14.3). Let  $v \in \mathcal{V}$  and  $d \in \mathcal{D}_v$  be arbitrary. In the following, we omit the  $v \mapsto d$  subscript for the sake of brevity. Plugging in the definition of  $y$  into the left-hand side of (14.3) gives:

$$\begin{aligned} & \sum_{a \in \mathcal{A}} \delta^a \left( \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) y_a^t + [a = \hat{a}] \right) + [\mathcal{G}[v] = d] \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) v_{\mathcal{G}}^t \\ &= \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) \left( \sum_{a \in \mathcal{A}} \delta^a y_a^t + [\mathcal{G}[v] = d] v_{\mathcal{G}}^t \right) + \delta^{\hat{a}} \\ &\leq \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) [t[v] = d] + \delta^{\hat{a}} \end{aligned} \quad (14.4a)$$

$$\begin{aligned} &= \sum_{o \in \text{out}(\hat{a})} \text{prob}(o) [\hat{s}[o][v] = d] + \delta^{\hat{a}} \\ &= [\hat{s}[v] = d] \sum_{\substack{o \in \text{out}(\hat{a}): \\ \text{eff}(o)[v] = \perp}} \text{prob}(o) + \sum_{\substack{o \in \text{out}(\hat{a}): \\ \text{eff}(o)[v] = d}} \text{prob}(o) + \delta^{\hat{a}} \end{aligned} \quad (14.4b)$$

$$\begin{aligned} &= [\hat{s}[v] = d] \left( 1 - \sum_{\substack{o \in \text{out}(\hat{a}): \\ \text{eff}(o)[v] \neq \perp}} \text{prob}(o) \right) + \sum_{\substack{o \in \text{out}(\hat{a}): \\ \text{eff}(o)[v] = d}} \text{prob}(o) + \delta^{\hat{a}} \\ &= [\hat{s}[v] = d] \left( 1 - \sum_{\substack{o \in \text{out}(\hat{a}): \\ \text{eff}(o)[v] \neq \perp}} \text{prob}(o) \right) + \text{prod}^{\hat{a}} + (\text{cons}^{\hat{a}} - \text{prod}^{\hat{a}}) \\ &= [\hat{s}[v] = d] \left( 1 - \sum_{\substack{o \in \text{out}(\hat{a}): \\ \text{eff}(o)[v] \neq \perp}} \text{prob}(o) \right) + \text{cons}^{\hat{a}} \end{aligned} \quad (14.4c)$$

where (14.4a) holds because all  $y^t$  satisfy (14.3); and (14.4b) uses the fact that  $v \mapsto d$  holds after an outcome  $o$  either if  $o$  has no effect on  $v$  and  $v \mapsto d$  already holds in  $\hat{s}$ , or if  $o$  sets  $v$  to  $d$ . If  $\text{cons}^{\hat{a}} = 0$ , then it directly follows that (14.4c) is  $\leq [\hat{s}[v] = d]$ , i.e., (14.3) is satisfied. If  $\text{cons}^{\hat{a}} > 0$ , then  $\hat{s}[v] = \text{pre}(\hat{a})[v] = d$ . Hence, by definition,  $\text{cons}^{\hat{a}} = \sum_{o \in \text{out}(\hat{a}), \text{eff}(o)[v] \neq \perp} \text{prob}(o)$ . Plugged into (14.4c) yields  $(1 - \text{cons}^{\hat{a}}) + \text{cons}^{\hat{a}} = 1 = [\hat{s}[v] = d]$ . In summary,  $y, v_{\mathcal{G}}$  is a feasible solution to  $\Gamma^{\text{seq}}(\hat{s})$ , and thus  $\Gamma^{\text{seq}}$  is consistent.  $\square$

While we have not introduced Trevizan et al.'s (2017b)  $h^{\text{roc}}$  heuristic formally, we still want to point out that  $h^{\text{roc}}$  and the operator-counting heuristic as per Definition 14.5 are equivalent, glossing over the different objective settings. Intuitively, the action-outcome counts  $y_{a,o}$  of  $h^{\text{roc}}$  map equivalently into the probabilistic operator-counting variables through the division by the outcome probability, i.e., defining  $y_a = \frac{1}{\text{prob}(o)} y_{a,o}$  for each  $a$ , and some arbitrary  $o \in \text{out}(a)$ . Note that all outcomes yield the same probabilistic operator count due to the regrouping constraints. Vice versa, the action-outcome counts can be defined from the probabilistic operator-counting by multiplying with the respective outcome probability.

The domination relation between  $h^{\text{roc}}$  and the atomic projection occupation-measure heuristic shown by Trevizan et al. (2017b) still holds in our setting. The goal-probability state-equation heuristic cannot provide tighter goal-probability bounds than the projection occupation-measure heuristic over the atomic projections. We provide the proof in Appendix C.3.2.

**Theorem 14.4.** *Let  $\mathcal{X} = \{ \{v\} \mid v \in \mathcal{V} \}$  be the set of singleton variable sets. For every state  $s$ , there is an optimal solution  $om^*, v_{\mathcal{G}}^*$  to the  $H_{\mathcal{X}}^{\text{gpm}}(s)$  LP such that, for an arbitrary  $v_0 \in \mathcal{V}$ ,*

$$\begin{aligned} v_{\mathcal{G}} &= v_{\mathcal{G}}^* \\ y_a &= \sum_{d \in \mathcal{D}_{v_0}} om_{v_0, d, a}^* \end{aligned} \quad \text{for all } a \in \mathcal{A}$$

*satisfies  $\Gamma^{\text{seq}}(s)$ .*

**Corollary 14.2.** *Let  $\mathcal{X}$  be the set of singleton variable sets. It holds for all states  $s$  that  $H_{\mathcal{X}}^{\text{gpm}}(s) \leq H_{\Gamma^{\text{seq}}}^{\text{gpm}}(s)$ .*

Trevizan et al. (2017b) hypothesize for the expected-cost versions that the dominance relation also holds in the opposite direction, i.e., that both heuristics are actually equivalent. This hypothesis is further substantiated by Pommerening et al.’s (2014) result in the classical planning context, showing that the state equation heuristic dominates *optimal cost-partitioning* over the atomic projections, i.e., the direct analogue to the projection occupation-measure heuristic. This hence strongly suggests that the equivalence also holds for the goal-probability variants. We however leave the proof for future work.

### Goal-Probability Landmark Constraints

Adapting another one of Pommerening et al.’s (2014) operator-counting constraints from classical planning, our last goal-probability operator-counting constraint is based on the notion of *disjunctive action landmarks*:

**Definition 14.6** (Disjunctive Action Landmark). *Let  $\Pi = \langle \mathcal{V}, \mathcal{A}, I, \mathcal{G} \rangle$  be a probabilistic FDR task. Let  $s$  be a state of  $\Pi$ . A set of actions  $L \subseteq \mathcal{A}$  is a **disjunctive action landmark** (short **landmark**) for  $s$ , if every path in  $\mathcal{M}^{\Pi}$  from  $s$  to a goal state includes an action from  $L$ .*

In the classical planning variant, the landmark operator-counting constraint for a landmark  $L$  simply requires that the sum of operator counts over  $L$  is at least 1. Notice that this does not work in our setting directly. For example, consider the task from Example 14.1.  $L = \{a_3\}$  is a landmark for the initial state. Yet, no policy can execute  $a_3$  more than  $\frac{1}{2}$  times, since its  $r \mapsto 1$  precondition is achieved by  $a_1$  only with a probability of  $\frac{1}{2}$ . However, observe that the probabilistic operator counts from  $L$  necessarily upper bound the goal probability, given that every goal path must include some action from  $L$ . Therefore, we simply substitute the strict 1 bound by  $v_{\mathcal{G}}$ , and obtain:

**Definition 14.7** (Goal-Probability Landmark Constraint). *Let  $s$  be a state. Let  $L \subseteq \mathcal{A}$  be a landmark of  $s$ . The **goal-probability landmark constraint**  $\gamma_L^{\text{lm}}$  for  $L$  is*

$$\sum_{a \in L} y_a \geq v_{\mathcal{G}} \tag{14.5}$$

**Example 14.3.** *Consider the goal-probability operator-counting heuristic equipped with projection occupation-measure constraints from Figure 14.1. Note that  $L = \{a_2\}$  is a landmark for  $s_I$ . The occupation-measure variables for  $r$  equal to the probabilistic operator-counting variables. Hence, the flow constraint of  $r \mapsto 1$*

can be interpreted equivalently as  $2y_{a_2} \leq y_{a_1}$ . The goal constraint can be viewed as  $v_G \leq 1 - y_{a_1} + y_{a_2}$ , i.e., combined with flow constraint, (1)  $v_G \leq 1 - y_{a_2}$ . Consider the additional landmark constraint for  $L$ : (2)  $y_{a_2} \geq v_G$ . The largest value of  $v_G$  that satisfies both (1) and (2) is given by the intersection of both inequalities:  $1 - y_{a_2} = y_{a_2}$ , i.e.,  $v_G = y_{a_2} = \frac{1}{2}$ . With the landmark constraint for  $L$ , we hence obtain an estimate of  $\frac{1}{2}$ , improving the prior bound  $\frac{2}{3}$ .

**Theorem 14.5.** Let  $L \subseteq \mathcal{A}$  be a subset of actions. Suppose

$$\Gamma_L^{\text{lm}}(s) = \begin{cases} \{\gamma_L^{\text{lm}}\}, & \text{if } L \text{ is a landmark of } s \\ \emptyset, & \text{otherwise} \end{cases}$$

Then  $\Gamma_L^{\text{lm}}$  is a goal-probability operator-counting constraint.

*Proof.* Setting all variables to 0 trivially satisfies (14.5), i.e.,  $\Gamma_L^{\text{lm}}$  is feasible. Since goal states can have by definition no landmarks, i.e.,  $\Gamma_L^{\text{lm}}(s_*) = \emptyset$  for all  $s_* \in \mathcal{S}_*^\Pi$ ,  $\Gamma_L^{\text{lm}}$  is also goal-respecting. Suppose for contradiction that  $\Gamma_L^{\text{lm}}$  is not consistent. Let  $\hat{s}$  and  $\hat{a}$  be the state and action, and  $y^t, v_G^t$  be the feasible successor solutions, where the consistency property is violated. Let  $y, v_G$  be the combination of the successor solutions as in Definition 14.3. By assumption,  $y, v_G$  does not satisfy  $\Gamma_L^{\text{lm}}(\hat{s})$ , i.e.,  $\Gamma_L^{\text{lm}}(\hat{s}) = \{\gamma_L^{\text{lm}}\}$ , and  $L$  must be landmark for  $\hat{s}$ . Plugged the landmark constraint (14.5), this gives:

$$\sum_{a \in L} \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) y_a^t + [\hat{a} \in L] < \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) v_G^t$$

Notice that  $\hat{a}$  cannot be contained in  $L$ , since all operator-counting variables must be non-negative, and the right hand side can be at most 1. Hence,  $L$  is still a landmark for all successors states  $t \in \text{Succ}(\hat{s}, \hat{a})$ . This yields a contradiction:

$$\begin{aligned} & \sum_{a \in L} \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) y_a^t < \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) v_G^t \\ \Leftrightarrow & \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) \sum_{a \in L} y_a^t < \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) v_G^t \\ \Leftrightarrow & \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) \left( \sum_{a \in L} y_a^t - v_G^t \right) < 0 \end{aligned}$$

where the difference  $\sum_{a \in L} y_a^t - v_G^t$  must be non-negative, since all  $y^t$  satisfy (14.5) by assumption.  $\square$



# 15. State-Space Reduction Techniques

The tighter the heuristic function bounds the optimal value function, the more effective heuristic search becomes. Yet, upper-bounding goal-probability heuristics are still sparse. Moreover, even if we make the idealistic assumption of being provided with an *almost perfect* heuristic, i.e., one whose estimates are off the optimal values by just a constant factor, Helmert and Röger (2008) have shown that search effort can still scale exponentially in the problem size. While Helmert and Röger have considered specifically classical planning, their result directly extends to the probabilistic setting. In this chapter, we explore two optimality-preserving state-space reduction techniques, applicable independent of the search algorithm. Section 15.1 leverages the well-known notion of *bisimulation* to this end. Section 15.2 discusses dead-end detection as a means to identify and prune irrelevant states. Dead-end pruning is particularly promising in our budget-limited MDP variant. In Section 15.3, we consider this case specifically, exploring a related dead-end pruning technique from budget-limited (oversubscription) classical planning (Domshlak and Mirkis, 2015).

## 15.1. Probabilistic Bisimulation

Bisimulation is a known method to reduce the size of state spaces (Milner, 1990; Larsen and Skou, 1991; Dean and Givan, 1997). The idea essentially is to group sets of equivalent states together as block states, and then solve the smaller MDP over these block states. Here, we observe that this approach can be fruitfully combined with state-of-the-art classical planning techniques, namely *merge-and-shrink* heuristics (Dräger et al., 2009; Helmert et al., 2014), which allow to effectively compute a bisimulation over the *determinized* state space. Determinized-bisimilar states are bisimilar in the probabilistic state space as well, so this identifies a practical special case of probabilistic bisimulation given a factored problem specification.

Let us spell this out in a little more detail. Formally, a bisimulation is a relation between states with the following property (Larsen and Skou, 1991; Dean and Givan, 1997):

**Definition 15.1** (Probabilistic Bisimulation). *Let  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  be an MDP. Let  $\sim \subseteq \mathcal{S} \times \mathcal{S}$  be an equivalence relation over states.  $\sim$  is called a **probabilistic bisimulation** for  $\mathcal{M}$  if, whenever  $s \sim s'$ , it holds that*

(i)  $s \in \mathcal{S}_*$  iff  $s' \in \mathcal{S}_*$ , and

(ii) for every  $a \in \mathcal{A}$ , and  $t \in \mathcal{S}$ ,  $\sum_{t' \in \mathcal{S}: t \sim t'} \mathcal{P}(s, a, t') = \sum_{t' \in \mathcal{S}: t \sim t'} \mathcal{P}(s', a, t')$

We call two states  $s$  and  $s'$  probabilistic bisimilar, if there exists a probabilistic bisimulation  $\sim$  for  $\mathcal{M}$  with  $s \sim s'$ . Let  $S_1^\sim, \dots, S_n^\sim$  be the equivalence classes induced by such  $\sim$ , and consider the quotient MDP  $\mathcal{Q}^\sim$  over these equivalence classes (cf. Definition 12.4), i.e., the MDP, in which every  $S_i^\sim$  is collapsed into a single block state  $t_{S_i^\sim}$ . Dean and Givan show that an optimal solution to this quotient MDP induces an optimal solution to the MDP itself:

**Theorem 15.1** (Dean and Givan, 1997). Let  $\mathcal{M}$  be an MDP with states  $\mathcal{S}$ , and  $\sim$  be a probabilistic bisimulation for  $\mathcal{M}$ . Suppose  $\pi^\sim$  is a policy for  $\mathcal{Q}^\sim$ . Then defining

$$\pi(s) := \pi^\sim(\mathbf{t}_{[s]_\sim})$$

for all  $s \in \mathcal{S}$  yields a policy for  $\mathcal{M}$  such that  $V^\pi(s) = V^{\pi^\sim}(\mathbf{t}_{[s]_\sim})$  holds for all  $s \in \mathcal{S}$ . The state set  $[s]_\sim$  denotes the equivalence class to which  $s$  belongs.

Now, consider a *determinized bisimulation* (Milner, 1990; Nissim et al., 2011) for the state space  $\Theta^{\Pi_D}$  induced by the all-outcomes determinization  $\Pi_D$ :

**Definition 15.2** (Bisimulation). Let  $\Theta = \langle \mathcal{S}, \mathcal{L}, \mathcal{T}, s_I, \mathcal{S}_*, \mathbf{c} \rangle$  be an LTS. Let  $\sim \subseteq \mathcal{S} \times \mathcal{S}$  be an equivalence relation over states. Then  $\sim$  is called a **bisimulation** for  $\Theta$  if, whenever  $s \sim s'$ , it holds that

- (1)  $s \in \mathcal{S}_*$  iff  $s' \in \mathcal{S}_*$ , and
- (2) if  $\langle s, a, t \rangle \in \mathcal{T}$ , for some  $t \in \mathcal{S}$ , then there exists  $t' \in \mathcal{S}$  such that  $\langle s', a, t' \rangle \in \mathcal{T}$ , and  $t \sim t'$ .

It is easy to see that any bisimulation for  $\Theta^{\Pi_D}$  is a probabilistic bisimulation for  $\mathcal{M}^\Pi$ :

**Theorem 15.2.** Let  $\Pi$  be a (budget-limited) probabilistic FDR task. Suppose  $\sim_D$  is a bisimulation for  $\Theta^{\Pi_D}$ . Then  $\sim_D$  is a probabilistic bisimulation for  $\mathcal{M}^\Pi$ .

*Proof.* Condition (i) is clearly satisfied. To show (ii), let  $s, s' \in \mathcal{S}^\Pi$  be any states with  $s \sim_D s'$ . Consider any action  $a \in \mathcal{A}$ . Note that  $a$  is applicable in  $s$  iff  $a$  is applicable in  $s'$ , as per condition (2) of Definition 15.2. If  $a$  is not applicable in  $s$ , (ii) follows immediately. Suppose  $a$  is applicable in  $s$ , and let  $t \in \mathcal{S}^\Pi$  be any state. Observe that

$$\begin{aligned} \sum_{t' \in \mathcal{S}^\Pi : t' \sim_D t} \mathcal{P}^\Pi(s, a, t') &= \sum_{o \in \text{out}(a) : s \llbracket o \rrbracket \sim_D t} \text{prob}(o) \\ &= \sum_{\langle s, a^o, t' \rangle \in \mathcal{T}^{\Pi_D} : t' \sim_D t} \text{prob}(o) \end{aligned} \quad (15.1)$$

$$\begin{aligned} &= \sum_{\langle s', a^o, t' \rangle \in \mathcal{T}^{\Pi_D} : t' \sim_D t} \text{prob}(o) \\ &= \sum_{t' \in \mathcal{S}^\Pi : t' \sim_D t} \mathcal{P}^\Pi(s', a, t') \end{aligned} \quad (15.2)$$

where (15.1) uses the definition of the all-outcomes determinization; and (15.2) holds, because of condition (2) of the determinized bisimulation. The claim follows.  $\square$

Notice that condition (2) over the all-outcomes determinization is more restrictive than needed, as it insists on the *subset of outcomes* being the same on both sides, rather than only their *summed-up probability* being the same. In particular, one can show that there are probabilistic bisimulations for  $\Pi$  that are not bisimulations for  $\Pi_D$ :

**Example 15.1.** Consider a probabilistic FDR task with binary variables  $\mathcal{V} = \{v_1, v_2, v_3, v_4\}$ ; goal  $\mathcal{G} = \{v_4 \mapsto 1\}$ ; and two actions  $\mathcal{A} = \{a_1, a_2\}$ , where:

- $a_1$  has precondition  $\text{pre}(a_1) = \{v_3 \mapsto 0\}$  and two equally likely outcomes with  $\text{eff}(o_1) = \{v_1 \mapsto 1, v_3 \mapsto 1\}$  and  $\text{eff}(o_2) = \{v_2 \mapsto 1, v_3 \mapsto 1\}$

- $a_2$  has precondition  $\text{pre}(a_2) = \{v_1 \mapsto 1, v_2 \mapsto 1, v_3 \mapsto 1, v_4 \mapsto 0\}$ , and a single outcome with  $\text{eff}(o) = \{v_4 \mapsto 1\}$ .

Consider the states  $s_1 = \{v_1 \mapsto 1, v_2 \mapsto 0, v_3 \mapsto 0, v_4 \mapsto 0\}$  and  $s_2 = \{v_1 \mapsto 0, v_2 \mapsto 1, v_3 \mapsto 0, v_4 \mapsto 0\}$ . Note that both states are probabilistic bisimilar: both states have only one applicable action  $a_1$ , transitioning with  $\frac{1}{2}$  probability into the same state  $\{v_1 \mapsto 1, v_2 \mapsto 1, v_3 \mapsto 1, v_4 \mapsto 0\}$ , and with  $\frac{1}{2}$  they transition into two different terminal states. The terminal states are probabilistic bisimilar by definition. However,  $s_1$  and  $s_2$  are not bisimilar in the all-outcomes determinization:  $s_1$  transitions via  $o_1$  into the terminal state;  $s_2$  via  $o_2$ .

But how to compute a determinized bisimulation for  $\Pi$ ? The naive solution is to build the state space up front and then computing a determinized bisimulation on it. One can potentially do much better though, by using merge-and-shrink with the widely employed shrinking strategies based on bisimulation (Nissim et al., 2011; Helmert et al., 2014). In a nutshell, this algorithm framework constructs an abstraction by starting with a collection of abstractions each considering a single state variable only, then iteratively “merging” two abstractions (replacing them with their synchronized product) until only a single abstraction is left, and “shrinking” abstractions to a bisimulation thereof in between every merging step. As we shall see in the experiments, this often still incurs a prohibitive overhead, but it can be feasible, and lead to substantial state space size reductions. In some cases, it results in tremendous performance improvements.

## 15.2. Dead-End Pruning

Clearly, dead-end states can be treated exactly like terminal states, without harming correctness or optimality guarantees of any search algorithm. Hence, if we are able to detect a state  $s$  as dead end, the part of the state space below  $s$  no longer needs to be explored. Apart from this pruning itself, for the heuristic search algorithms, dead-end information can be leveraged as an additional source of heuristic information, yielding the initialization of the upper bound as  $V^U(s) = V^*(s) = 0$ . This more informed initial upper bound typically leads to additional search reductions.

The only question remaining open is how to detect dead ends? Kolobov et al. (2011) employ SIXTH-SENSE (Kolobov et al., 2010a), which learns dead-end detection rules by generalizing from information obtained using a classical planner. Here we instead exploit the power of classical-planning heuristic functions, run on the all-outcomes determinization. This is especially promising in limited-budget planning, where we can use lower bounds on determinized plan cost to detect states with insufficient remaining budget. Observe that this is natural and effective using admissible remaining-cost estimators, yet would be impractical using an actual classical planner (which would need to be optimal and thus prohibitively slow). In the unlimited-budget case, we can use any heuristic function able to detect dead ends (returning  $\infty$ ), which applies to most known heuristics.

**Proposition 15.1.** *Let  $\Pi$  be a probabilistic FDR task. Let  $s$  be a state in  $\Pi$ . Suppose  $h^*(s)$  is the cost of an optimal plan for  $s$  in  $\Pi_D$ , or  $h^*(s) = \infty$  if  $s$  is a dead end in  $\Pi_D$ . Then  $s$  is a dead end in  $\Pi$  iff  $h^*(s) = \infty$ . If  $\Pi$  is a limited-budget task, then  $s$  is a dead end in  $\Pi$  iff  $h^*(s) > \mathbf{b}(s)$ .*

Proposition 15.1 can be utilized via any classical-planning heuristic that admissibly approximates  $h^*$ .

### 15.3. Budget Pruning via Landmarks: Heuristic vs. Budget Reduction

For limited-budget planning, we also considered adopting the problem reformulation by Domshlak and Mirkis (2015) for oversubscription planning, which reduces the budget  $\mathbf{b}$  using landmarks and in exchange allows traversing yet unused landmarks at a reduced cost during search. It turns out, however, that pruning states whose reformulated budget is  $< 0$  is equivalent to the much simpler method pruning states whose heuristic exceeds the (original/not reformulated) remaining budget. The added value of Domshlak and Mirkis's reformulation thus lies, not in its pruning per se, but in its compilation into a planning language, creating the potential of synergizing with other heuristics.

For the sake of simplicity, let's consider a classical planning setup. The arguments in probabilistic and oversubscription setups are essentially the same. Let  $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  be an FDR task with a global budget  $\mathbf{b}_I$ . Let  $\mathcal{L}$  be a set of disjunctive action landmarks for  $s_I$  (cf. Definition 14.6), i.e., every  $L \in \mathcal{L}$  is a set of actions that contains an action from every plan for  $s_I$ . Let furthermore  $\{\mathbf{c}_L \mid L \in \mathcal{L}\}$  be a *cost partitioning* (Karpas and Domshlak, 2009) for  $\mathcal{L}$ , i.e., a set of non-negative action cost functions such that, for each  $a \in \mathcal{A}$ ,  $\sum_{L \in \mathcal{L}} \mathbf{c}_L(a) \leq \mathbf{c}(a)$ . Denote  $h(L) = \min_{a \in L} \mathbf{c}_L(a)$ , and for a subset  $\mathcal{L}' \subseteq \mathcal{L}$  of landmarks denote  $h(\mathcal{L}') = \sum_{L \in \mathcal{L}'} h(L)$ . Intuitively, each landmark  $L \in \mathcal{L}$  is assigned a weight  $h(L)$ , and the admissible heuristic value  $h(\mathcal{L})$  for  $s_I$  is obtained by summing up these weights.

We now describe Domshlak and Mirkis's (2015) pruning technique in these terms. Domshlak and Mirkis's formulation is based on a compilation into a planning language, which is more complicated, but is equivalent to our description here as far as the pruning is concerned.

Domshlak and Mirkis's technique maintains the “non-used” landmarks as part of states. Namely, for a state  $s$  reached from  $s_I$  via the path  $\sigma$ ,  $L \in \mathcal{L}$  is non-used in  $s$  iff  $\sigma$  contains no action from  $L$ . We denote the set of non-used landmarks in  $s$  by  $\mathcal{L}(s)$ . Obviously, every  $L \in \mathcal{L}(s)$  remains a landmark for  $s$ . Note also that, as  $\mathcal{L}(s)$  is part of the state, even if two search paths lead to the same end state but use different landmarks, their end states are considered to be different. This restriction arises from the compilation approach, where the book-keeping of landmarks must happen inside the language, i.e., inside states. One could formulate the pruning technique without this restriction; we get back to this below.

The pruning technique now arises from the interplay of a reduced global budget and reduced action costs depending on non-used landmarks. Define the reduced global budget as  $\mathbf{b}'_I = \mathbf{b}_I - h(\mathcal{L})$ . For any action  $a$ , denote by  $\mathcal{L}(a)$  the set of landmarks  $a$  participates in, i.e.,  $\mathcal{L}(a) = \{L \mid L \in \mathcal{L}, a \in L\}$ . For any state  $t$  during search, and an applicable action  $a$ , the transition from  $t$  to  $t[a]$  has a reduced cost, namely the cost  $\mathbf{c}(a) - h(\mathcal{L}(a) \cap \mathcal{L}(t))$ . In words, we reduce the cost of  $a$  by the summed-up weights of the non-used landmarks  $a$  participates in.

Consider now some state  $s$  during search. Denote the remaining reduced budget in  $s$  by  $\mathbf{b}'(s)$ . Say that we prune  $s$  iff  $\mathbf{b}'(s) < 0$ .<sup>1</sup> Consider any path  $\sigma$  from  $s_I$  to  $s$ . As non-used landmarks are part of the state, all these paths must touch the same subset of landmarks from  $\mathcal{L}$ , namely  $\mathcal{L} \setminus \mathcal{L}(s)$ . Denote the sum of the original (non-reduced) costs of the actions from  $\sigma$  by  $\mathbf{c}(\sigma)$ . Relative to this cost, the cost saved thanks to

<sup>1</sup>Domshlak and Mirkis (2015) do not maintain the remaining budget as part of the state, but instead prune  $s$  if  $g(s) > \mathbf{b}'_I$ , where  $g(s)$  is the cost of the path  $\sigma$  from  $s_I$  to  $s$ . This is, obviously, equivalent, except that duplicate detection is more powerful as it compares states based on their state variable assignments only. For the purpose of our discussion here, this does not make a difference. Note that, in the probabilistic setting, we do have to distinguish states based the remaining budget  $\mathbf{b}'(s)$  as well, as goal probability depends on both.

the cost reduction is exactly  $h(\mathcal{L} \setminus \mathcal{L}(s))$ , the weight of the touched landmarks. Hence

$$\begin{aligned}
 \mathbf{b}'(s) &= \mathbf{b}'_I - (\mathbf{c}(\sigma) - h(\mathcal{L} \setminus \mathcal{L}(s))) \\
 &= (\mathbf{b}_I - h(\mathcal{L})) - \mathbf{c}(\sigma) + h(\mathcal{L} \setminus \mathcal{L}(s)) \\
 &= (\mathbf{b}_I - \sum_{L \in \mathcal{L}} h(L)) - \mathbf{c}(\sigma) + \sum_{L \in \mathcal{L} \setminus \mathcal{L}(s)} h(L) \\
 &= \mathbf{b}_I - \mathbf{c}(\sigma) - \sum_{L \in \mathcal{L}(s)} h(L) \\
 &= \mathbf{b}_I - \mathbf{c}(\sigma) - h(\mathcal{L}(s))
 \end{aligned}$$

Thus,  $s$  is pruned,  $\mathbf{b}'(s) < 0$ , iff  $\mathbf{b}_I - \mathbf{c}(\sigma) < h(\mathcal{L}(s))$ . The latter condition is the same as  $\mathbf{b}(s) < h(\mathcal{L}(s))$ , which is exactly the pruning condition resulting from using  $h(\mathcal{L}(s))$  as an admissible heuristic function pruning against the remaining budget.

In a non-compilation setting, one could, as is indeed customary in admissible landmark heuristics, handle landmarks in a path-dependent manner. That is, non-used landmarks are maintained as annotations to states rather than as part of them, and multiple search paths may end in the same state  $s$  but use different landmarks. The set of remaining landmarks  $\mathcal{L}(s)$  for  $s$  then is the union over those for each individual path; that is,  $L \in \mathcal{L}$  is non-used in  $s$  iff there exists at least one path that does not touch  $L$ . This still suffices to guarantee that  $L$  is a landmark for  $s$ . The landmark heuristic approach as per Karpas and Domshlak (2009) does this kind of book-keeping, and uses the admissible heuristic value  $h(\mathcal{L}(s))$ .

If one were to apply Domshlak and Mirkis's (2015) reformulation technique without maintaining landmarks as part of state, then the notion of transition-cost reduction would have to become more complicated (lest one loses information). This is because, if  $s$  is reached on  $\sigma_1$  with a reduced cost due to touching landmark  $L_1$ , but later on we find another path  $\sigma_2$  to  $s$  that does not touch  $L_1$ , then  $L_1$  actually still is a valid landmark for  $s$ , and therefore there was no need to reduce the cost on  $\sigma_1$ . To account for this, we would have to revise path costs posthoc, every time a new path to  $s$  becomes available. After these revisions, the cost reduction on each path  $\sigma$  to  $s$  is exactly  $h(\mathcal{L} \setminus \mathcal{L}(s))$ : the weight of the non-used landmarks  $\mathcal{L}(s)$  is no longer subtracted, and the weight of the other landmarks  $\mathcal{L} \setminus \mathcal{L}(s)$  is subtracted on every  $\sigma$  because, by definition, every  $\sigma$  touches every  $L \in \mathcal{L} \setminus \mathcal{L}(s)$ . So the cost saved on every path  $\sigma$  to  $s$ , relative to  $\sigma$ , is exactly  $h(\mathcal{L} \setminus \mathcal{L}(s))$ , from which point the same arguments as above apply to show that the pruning is equivalent to pruning via  $\mathbf{b}(s) < h(\mathcal{L}(s))$ .

In summary, pruning  $s$  based on reduced remaining budget  $\mathbf{b}'(s) < 0$  is equivalent to pruning  $s$  based on original remaining budget vs. the landmark heuristic  $\mathbf{b}(s) < h(\mathcal{L}(s))$ . It should be noted, though, that such pruning is not the only benefit of Domshlak and Mirkis's (2015) reformulation technique. The technique allows to compute another, complementary, admissible heuristic  $h$  on the reformulated task  $\Pi'$  (and this is what Domshlak and Mirkis point out as part of the motivation, and what they do in practice). From our perspective here, the landmark heuristic and  $h$  are used *additively* for admissible pruning against the remaining budget, where additivity is achieved with a method generalizing cost partitionings: in  $\Pi'$ , the cost-reduced variant of each action can be applied only once. So if  $h$  does not abstract away this constraint, and if  $h$  uses an action twice, then it employs the reduced cost only once, yet pays the full cost the second time. Exploring this kind of generalized cost partitioning in more detail is an interesting research line for future work.



# 16. Experimental Evaluation

Over the course of the previous chapters, we have developed an extensive design space of probabilistic state-space search methods for solving goal-probability objectives. Here, we systematically explore this design space through a comprehensive empirical evaluation, and therewith shed light on the current state of the art in goal-probability planning.

In summary, the algorithm design space encompasses the following top-level choice points:

- (a) *Search algorithm* (cf. Chapters 11 and 12).
- (b) *Tie-breaking strategy* (cf. Chapter 13).
- (c) *State-space reduction technique* (cf. Chapter 15).
- (d) *Goal-probability heuristic* (cf. Chapter 14).

These algorithm dimensions are mostly orthogonal. To make our experiments feasible, we split them into two parts. In part I), we inspect the various options for (a) – (c) from the preceding chapters, analyzing the resulting algorithm behavior on the different goal-probability objectives. Based upon the findings, we then investigate in part II) the impact of the goal-probabilistic heuristics (d) on the best-performing configurations from I). In what follows, we first describe our implementation and the benchmarks. We then discuss the results for I) and II).

The source code and benchmarks are publicly available<sup>1</sup>. All experiments were run on a cluster of Intel Xeon E5-2660 machines running at 2.20 GHz, with time and memory cut-offs of 30 minutes and 4 GB. We used Cplex 12.6.3 as the LP solver. In cases where necessary, the convergence parameter  $\epsilon$  was set to  $5 \cdot 10^{-5}$  (the same value as used by Kolobov et al. (2011), the main prior work on optimal MDP heuristic search for goal-probability probabilistic planning).

## 16.1. Implementation

We implemented all algorithms in FAST DOWNWARD (FD) (Helmert, 2006). As our focus is on goal-oriented MDPs with probabilistic actions, whose outcomes are listed explicitly, naturally we use PPDDL (Younes et al., 2005), rather than RDDL (Sanner, 2010), as the surface-level language. FD’s translator component was extended to handle PPDDL, and we added support for specifying a numeric budget limit.

Given the FD implementation framework in contrast to previous works on optimal probabilistic planning, we implemented all algorithms from scratch. For FRET, we closely followed the original implementation, up to details not specified by Kolobov et al. (2011), based on personal communication with Andrey Kolobov. Kolobov’s original source code is not available anymore, which also plays a role in our state-of-the-art comparison, see next.

---

<sup>1</sup><https://doi.org/10.5281/zenodo.6992688>

Given the scant prior work on optimal goal-probability analysis in planning, the state of the art is represented by topological VI, by LRTDP with dead-end pruning on acyclic problems, and by FRET-V using LRTDP with dead-end pruning on cyclic problems. All these configurations are particular points in the space of configurations we explore, so the comparison to the state of the art is part of our comparison across configurations. The only thing missing here is the particular form of dead-end detection, which was SIXTHSENSE (Kolobov et al., 2010a) in the only prior work, by Kolobov et al. (2011). As SIXTHSENSE is a complex method and advanced dead-end pruning via heuristic functions is readily available in our framework, we did not re-implement SIXTHSENSE. Our discussion of cyclic problems in Section 16.3.2 below includes a detailed comparison of our results with those by Kolobov et al. (2011), on IPPC ExplodingBlocks which is the only domain Kolobov et al. considered.

Note that providing quality guarantees is an important property in this study. For this reason, and for the sake of clarity, we do not compare against unbounded suboptimal approaches, such as using an algorithm with a discounted criterion or assigning large finite penalties to dead ends (Teichteil-Königsbuch et al., 2011; Kolobov et al., 2012a).

LRTDP is a randomized algorithm. To test the robustness of LRTDP, we executed selected configurations 10 times, varying the RNG seed. The performance between the different runs was almost indistinguishable (standard deviation of runtime is  $< 1$  second in most cases, and peak memory usage  $< 2$  MB). For the sake of simplicity, we selected one of the RNG seeds arbitrarily, and we keep it fixed throughout all the following experiments.

## 16.2. Benchmark Suite

Our aim being to comprehensively explore the relevant problem space, we designed a broad suite of benchmarks, 1641 instances in total, based on domains from the International Probabilistic Planning Competitions (IPPC), resource-constrained planning (RCP), and network penetration testing (pentesting).

**IPPC** From the IPPC, we selected those PPDDL domains in STRIPS format, or with moderate non-STRIPS constructs, easily compilable into STRIPS. This resulted in 12 domains from IPPC'04 – IPPC'08.

**Canadian RCP Domains** For resource-constrained planning, we adopted the NoMystery, Rovers, and TPP benchmarks by Nakhost et al. (2012), more precisely those suites with a single consumed resource (fuel, energy, money), which correspond to limited-budget planning. To make the benchmarks feasible for optimal probabilistic planning, we had to reduce their size parameters (number of locations etc). We scaled all parameters with the same number  $< 1$ , chosen to get instances at the borderline of feasibility for VI. We created probabilistic versions by adding uncertainty about the underlying road map, akin to the *Canadian Traveller's Problem* (Papadimitriou and Yannakakis, 1991), each road segment being present with a given probability (this is encoded through a separate, probabilistic, action attempting a segment for the first time, akin to Example 10.1). For simplicity, we set that probability to 0.8 throughout.

**Pentesting** For pentesting, the general objective is – using *exploits* – to compromise computers in a network, one after another, until specific targets are reached (or no action is available). We modified the POMDP generator by Sarraute et al. (2012), which itself is based on a test scenario used at Core Security<sup>2</sup>

<sup>2</sup><http://www.coresecurity.com/>

to output PPDDL encodings of Hoffmann’s (2015) *attack-asset MDP* pentesting models. In these models, the network configuration is known and fixed, and each exploit is callable once and succeeds (or fails) with some probability. The generator uses a network consisting of an exposed part, a sensitive part, and a user part. It allows to scale the numbers  $h$  of hosts and  $E$  of exploits. Sarraute et al.’s (2012) POMDP model and solver (Kurniawati et al., 2008) scale to  $H = 6, E = 10$ .<sup>3</sup> For our benchmarks, we fixed  $H = E$  for simplicity. We scaled the instances from  $6 \dots 20$  without budget limit, and from  $10 \dots 24$  with budget limit.

From each of the above benchmark tasks  $\Pi$  (except the pentesting ones for which we already generated a separate limited-budget version anyway), we obtained several limited-budget benchmarks, as follows. We set outcome costs to 1 where not otherwise specified. We determined the minimum budget  $b_{\min}$  required to achieve non-0 goal probability. For the resource-constrained benchmarks,  $b_{\min}$  is given by the generator itself, as the minimum amount of resource required to reach the goal in the deterministic domain version. For all other benchmarks, we ran FD with  $A^*$  using the LMcut heuristic (Helmert and Domshlak, 2009) on the all-outcomes determinization of  $\Pi$ . If this failed, we skipped  $\Pi$ , otherwise we read  $b_{\min}$  off the cost of the optimal plan. We created several limited-budget tasks  $\Pi_{\mathfrak{C}}$ , differing in their *constrainedness level*  $\mathfrak{C}$ . Namely, following Nakhost et al. (2012), we set the initially available budget in  $\Pi_{\mathfrak{C}}$  to  $b(\mathcal{I}_b) := \mathfrak{C} * b_{\min}$ , so that  $\mathfrak{C}$  is the factor by which the available budget exceeds the minimum needed (to be able to reach the goal at all). We let  $\mathfrak{C}$  range in  $\{1.0, 1.2, \dots, 2.0\}$ .

### 16.3. Evaluating the State of the Art in Goal-Probability Planning

We evaluate the algorithm design space spanned by the parameters (a) – (c) from above. We consider all three goal-probability objectives, AtLeastProb, ApproxProb, and MaxProb. For AtLeastProb, we let  $\theta$  range in  $\{0.1, 0.2, \dots, 1.0\}$  ( $\theta = 0$  is pointless). For ApproxProb, we let  $\delta$  range in  $\{0.0, 0.1, \dots, 0.9\}$  ( $\delta = 1$  is pointless). We next discuss the results for the acyclic benchmarks, where a FRET-like analysis is generally not needed. Afterwards, we consider the cyclic part.

#### 16.3.1. Acyclic Planning

The acyclic part of our benchmark suite comprises the budget-limited benchmarks (all action outcomes have a non-zero cost), pentesting with and without budget limit (in either case, exploits can be used just once), as well as IPPC TriangleTireworld (moves are unidirectional). We consider the three objectives MaxProb, AtLeastProb, and ApproxProb. We run the search algorithm variants: VI, GOALPROB-EXHAO\*, GOALPROB-AO\*, GOALPROB-LRTDP, 14 GOALPROB-DFHS variants, and GOALPROB-IDUAL. We do not run GOALPROB-EXHDFS, since the simple bottom-up update procedure of GOALPROB-EXHAO\* is sufficient in acyclic planning, and GOALPROB-EXHAO\* by default explores the state space in a depth-first manner. In the following, we will omit the “GOALPROB-” prefix in algorithm names. Keep in mind though that our algorithms differ from the original ones, in particular in terms of early termination which depends on the objective MaxProb, AtLeastProb, or ApproxProb.

To study the termination benefits of the lower bound, we will switch it on and off where applicable, enabling/disabling the corresponding early termination conditions. Where  $\mathcal{A}$  denotes one of our heuristic

<sup>3</sup>For modeling/solving the entire network, that is. With their domain-dependent decomposition algorithm “4AL”, trading accuracy for performance, Sarraute et al. scale up much further.

	# Total
<b>i. Search Algorithms &amp; Pruning in MaxProb</b>	96
Search algorithms: • VI; • ExhAO*; • AO* <sub> U</sub> ; • LRTDP <sub> U</sub> ; • DFHS <sub> U</sub> (×14); • IDUAL; and • VI on DB	
Dead-end pruning: • disabled; • LMcut; • PDB; and • M&S with $N \in \{50k, \infty\}$	
Tie-breaking: default	
<b>ii. AtLeastProb and ApproxProb Objective Parameters</b>	17
Search algorithms: • VI; • ExhAO*; • AO* <sub> U</sub> ; • AO* <sub> LU</sub> ; • LRTDP <sub> U</sub> ; • LRTDP <sub> LU</sub> ; • DFHS <sub> U</sub> (×1, best from i.); • DFHS <sub> LU</sub> (×1); and • IDUAL	
Dead-end pruning: PDB	
Tie-breaking: default	
<b>iii. Tie-Breaking Strategies on AtLeastProb and ApproxProb</b>	57
Search algorithms: • VI; • ExhAO*; • AO* <sub> U</sub> ; • AO* <sub> LU</sub> ; • LRTDP <sub> U</sub> ; • LRTDP <sub> LU</sub> ; • DFHS <sub> U</sub> (×1); • DFHS <sub> LU</sub> (×1); and • IDUAL	
Dead-end pruning: PDB	
Tie-breaking: <i>all</i> (• 1; • 4; • 4; • 5; • 3; • 4; • 3; • 4; and • 1)	

Table 16.1.: Overview of experiments on the acyclic benchmark part. In ii. and iii., the heuristic search configurations are doubled because AtLeastProb vs. ApproxProb result in different algorithm configurations (using different termination criteria).

search algorithms, we denote by  $\mathcal{A}|_U$  and  $\mathcal{A}|_{LU}$  the variants of  $\mathcal{A}$  maintaining only  $V^U$  respectively both  $V^U$  and  $V^L$ .

Each search algorithm instance is run with up to 5 tie-breaking strategies, as described in Chapter 13.

For dead-end pruning / pruning against the remaining budget, we run LMcut (Helmert and Domshlak, 2009), a *canonical PDB heuristic* over patterns generated through a hill-climbing procedure (Haslum et al., 2007), and merge-and-shrink (M&S) with the state-of-the-art shrinking strategies based on bisimulation and an abstraction-size bound  $N$  (Nissim et al., 2011; Helmert et al., 2014); we show data for  $N = \infty$  and  $N = 50k$  (we also tried  $N \in \{10k, 100k, 200k\}$  which resulted in similar behavior). We also run variants without dead-end pruning. We use the deterministic-bisimulation (DB) reduced state space only for VI: once (and if) a bisimulation is successfully computed, the quotient MDP is easily solved by that simplest algorithm. Given DB, we do not require any dead-end pruning because all dead ends are already removed from the reduced state space.

Overall, this yields 1791 different possible algorithm configurations. We do not actually test all these configurations, of course, as not all of them are interesting, or needed to make the essential observations. We instead organize our experiment in terms of three parts i. – iii., each focusing on a particular issue of interest. Consider Table 16.1, which gives an overview of the configurations considered in each experiment.

The design of the experiments is as follows:

- i. We first evaluate different search algorithms and dead-end pruning methods on MaxProb, fixing the tie-breaking strategy to default.

We omit here all  $A|_{LU}$ , because for MaxProb heuristic search, maintaining  $V^L$  is redundant (early termination is dominated by regular termination).

- ii. We next fix the best-performing dead-end pruning method, and analyze search algorithm performance in AtLeastProb and ApproxProb as a function of the parameter  $\theta$  respectively  $\delta$ .

We again fix the tie-breaking strategy to default here, leaving their examination to iii.

- iii. We finally vary the tie-breaking strategy, keeping otherwise the setting of experiment ii.

We will conclude our discussion with additional data illustrating typical anytime behavior. Each part of the experiment is described in a separate sub-section in what follows.

### Search Algorithms & Pruning Methods in MaxProb

Tables 16.2 and 16.3 show coverage data, i.e., the number of benchmark instances for which MaxProb was solved within the given time and memory limits.

Consider first Table 16.2, which shows the results for a representative selection of  $DFHS|_U$  configurations. A comparison of all DFHS parameter combinations is available in Appendix A.2. Overall, there are only little performance differences between the different variants. The termination parameter has the largest impact. The LABEL variants dominate their VI counterparts throughout. Among the VI configurations, the additional forward updates (FW) yield a slight but consistent advantage, while the cutoff options are more detrimental than helpful. This is so because the former tends to decrease the number of VI calls by spotting inconsistency early; while the latter options consistently result in more, presumably mostly redundant, DFHS iterations. In contrast, for LABEL termination, the benefit of the additional forward updates disappears almost entirely. Yet, the TIP variants (cutting off the exploration at tip states) have a slight edge over disabling cutoffs completely. This difference mainly comes from Blocksworld-b, ExplodingBlocks-b, and TriangleTireworld-b, where the TIP configurations solve more instances, while in Elevators-b the TIP configurations perform slightly worse than with cutoffs disabled. The other cutoff option (INC), stopping the exploration at  $\epsilon$ -inconsistent states, is detrimental in almost all domains, most notably in Blocksworld-b, Elevators-b, and Zenotravel-b. Given the relative close overall performance, and since the  $DFHS|_U$  configuration LABEL/BW/TIP yields the overall highest coverage results, we will use it as the representative of the DFHS family in the remaining discussion. Note that this DFHS variant differs from ILAO\* only in the termination method, LABEL vs. VI. For the sake of simplicity, we will be henceforth referring to it as *Labeled ILAO\**, short LILAO\*.

Table 16.3 shows the coverage results for the remaining search algorithms and pruning functions. Due to space reasons, IDUAL is represented only in combination with PDB, yet, qualitatively, the differences between the different heuristic functions for IDUAL are similar to the other search algorithms. Of the pruning methods, PDB clearly stands out. For every search algorithm, it yields the by far best overall coverage. LMcute has a substantial advantage only in ExplodingBlocks-b. M&S has an advantage in Schedule-b. Note that, for M&S with  $N = \infty$ , overall coverage is worse than for using no pruning at all. This is due to the prohibitive overhead, in some domains, of computing a bisimulation on the determinized state space. Running VI directly on the deterministic-bisimulated state space (“VI on DB”) performs slightly worse

Domain	#	DFHS  <sub>U</sub>																	
		VI									LABEL								
		BW			FW			BW			FW			TIP ∪ INC					
		NONE	TIP		NONE	INC		NONE	TIP		NONE	INC		NONE	INC				
		-	LM	PDB	-	LM	PDB	-	LM	PDB	-	LM	PDB	-	LM	PDB	-	LM	PDB
IPPC Benchmarks																			
TriTire	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
IPPC Benchmarks with Budget Limit																			
Blocksw-b	180	54	64	99	54	64	100	54	64	99	54	64	112	54	64	109	54	64	102
Boxw-b	18	0	<b>3</b>	<b>3</b>	0	<b>3</b>	0	0	<b>3</b>	<b>3</b>	0	<b>3</b>	<b>3</b>	0	<b>3</b>	<b>3</b>	0	<b>3</b>	<b>3</b>
Drive-b	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90
Elevators-b	90	80	86	84	76	84	81	83	87	87	77	86	81	89	<b>90</b>	<b>90</b>	83	89	86
ExpBloc-b	150	64	118	99	62	119	95	64	119	99	63	118	97	64	123	100	63	<b>126</b>	101
Random-b	72	37	45	<b>53</b>	37	45	<b>53</b>	37	45	<b>53</b>	37	45	<b>53</b>	37	45	<b>53</b>	37	45	<b>53</b>
RecTire-b	36	24	<b>36</b>	<b>36</b>	23	<b>36</b>	<b>36</b>	25	<b>36</b>	<b>36</b>	27	<b>36</b>	<b>36</b>	25	<b>36</b>	<b>36</b>	27	<b>36</b>	<b>36</b>
Schedule-b	138	60	<b>65</b>	60	60	<b>65</b>	60	60	<b>65</b>	60	60	62	60	62	<b>65</b>	62	62	<b>65</b>	62
SeaResc-b	90	72	75	83	71	76	83	72	75	83	72	76	83	73	75	<b>84</b>	71	76	<b>84</b>
Tirew-b	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90
TriTire-b	60	52	56	57	49	55	56	52	57	57	52	56	56	52	57	57	51	57	<b>59</b>
Zenotra-b	78	36	<b>42</b>	<b>42</b>	31	<b>42</b>	<b>42</b>	37	<b>42</b>	<b>42</b>	25	37	39	<b>42</b>	<b>42</b>	<b>42</b>	<b>42</b>	<b>42</b>	<b>42</b>
Σ IPPC-b	1092	659	770	796	643	769	786	664	773	799	647	763	777	678	780	816	668	783	<b>818</b>
Canadian RCP Benchmarks with Budget Limit																			
NoMyst-b	60	13	45	52	12	43	49	15	46	52	15	45	52	24	52	<b>57</b>	21	49	<b>57</b>
Rovers-b	60	51	58	58	47	58	57	51	59	58	51	59	58	55	<b>60</b>	59	51	<b>60</b>	<b>60</b>
TPP-b	60	25	45	50	21	43	46	26	47	54	25	46	51	33	52	<b>55</b>	29	49	54
Σ RCP-b	180	89	148	160	80	144	152	92	152	164	91	150	161	112	164	171	101	158	171
Pentesting Benchmarks																			
Pentest-b	90	58	67	68	57	67	68	60	68	68	53	63	63	61	68	<b>70</b>	61	68	69
Pentest	15	8	9	9	8	9	9	9	9	9	8	8	8	9	<b>10</b>	9	9	9	9
Σ Pentest	105	66	76	77	65	76	77	69	77	77	61	71	71	70	78	<b>79</b>	70	77	78

Table 16.2.: Acyclic planning. MaxProb coverage (number of tasks solved within time & memory limits) for a representative selection of DFHS configurations. Best results in **bold**. Domains “-b” modified with budget limit. “#”: number of instances. DFHS parameters are abbreviated as “VI”: labeling (LABEL) is disabled, termination is checked via VI; “LABEL”: labeling is enabled; “BW”: value updates only on the way back up of the exploration; “FW”: additionally doing value updates on the way down of exploration; “NONE” no cutoffs, exploration is terminated only at terminal and goal states; “TIP”: cutting off exploration at tip states (CUTOFF<sub>TIP</sub>), i.e., states that have not been expanded yet; “INC” cutting off exploration at inconsistent states (CUTOFF<sub>INCONSISTENT</sub>); and “TIP ∪ INC” using both CUTOFF<sub>TIP</sub> and CUTOFF<sub>INCONSISTENT</sub>. Recall that ILAO\* corresponds to entry VI/BW/TIP; HDP corresponds to LABEL/FW/INC. Dead-end pruning disabled: “-”. Otherwise, pruning against remaining budget on “-b” domains; based on  $h(s) = \infty$  on other domains. “LM”: LMcut; “PDB”: canonical PDB heuristic. Default tie-breaking strategy.

than using the bisimulation for dead-end pruning. This is an artifact of our implementation, where in the absence of enough bisimilar states, the storage of the probabilistic bisimulation-reduced state space may actually require more memory than VI using the bisimulation for dead-end pruning. A notable exception is TriangleTireworld. Far beyond the standard benchmarks in Table 16.3 (triangle-side length 20), VI on DB scales to side length 74 in both the original domain and the limited-budget version. For comparison, the hitherto best solver by far was PROB-PRP (Camacho et al., 2016), which scales to side length 70 on the original domain, and is optimal only for goal probability 1, i.e., in the presence of strong cyclic plans

Domain	#	VI					exhAO*					AO*  <sub>U</sub>					LRTDP  <sub>U</sub>					LILAO*  <sub>U</sub>					idual	VI	
		–	LM	PDB	M&S N	∞	–	LM	PDB	M&S N	∞	–	LM	PDB	M&S N	∞	–	LM	PDB	M&S N	∞	–	LM	PDB	M&S N	∞	PDB	on DB	
IPPC Benchmarks																													
TriTire	10	5	5	5	5	5	5	5	5	5	5	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	
IPPC Benchmarks with Budget Limit																													
Blocksw-b	180	54	64	94	61	54	54	64	93	56	54	54	64	92	56	54	54	64	100	56	54	54	64	112	57	54	76	54	
Boxw-b	18	0	3	3	0	0	0	3	0	0	0	0	3	0	0	0	0	3	3	0	0	0	0	3	3	0	0	0	0
Drive-b	90	90	90	90	90	48	90	90	90	90	48	90	90	90	90	48	90	90	90	90	48	90	90	90	90	48	90	48	
Elevators-b	90	90	90	90	90	35	90	90	90	90	35	88	90	90	89	35	86	89	88	87	35	83	89	86	85	35	50	34	
ExpBloc-b	150	58	84	80	76	58	56	83	79	75	58	61	116	93	79	58	65	127	97	84	58	63	126	101	82	58	83	58	
Random-b	72	34	33	40	40	36	36	41	51	47	36	36	44	52	48	36	37	45	53	47	36	37	45	53	48	36	50	34	
RecTire-b	36	36	36	36	36	36	36	36	36	36	36	36	36	36	36	36	30	36	36	36	36	25	36	36	36	36	34	36	
Schedule-b	138	59	59	60	64	63	59	59	60	62	61	60	65	60	72	63	62	65	62	71	63	62	65	62	72	63	60	51	
SeaResc-b	90	74	75	85	85	84	72	75	83	82	82	72	76	82	82	82	71	75	84	84	84	71	76	84	85	84	62	80	
Tirew-b	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	
TriTire-b	60	50	54	55	55	55	47	53	54	54	54	49	56	57	57	57	52	57	58	57	57	51	57	59	58	58	50	60	
Zenotra-b	78	42	42	42	42	42	42	42	42	42	42	42	42	42	42	42	34	41	41	40	41	42	42	42	42	42	20	42	
Σ IPPC-b	1092	677	720	765	729	601	672	726	768	724	596	678	772	784	741	601	671	782	802	742	602	668	783	818	745	604	665	587	
Canadian RCP Benchmarks with Budget Limit																													
NoMyst-b	60	23	46	54	52	50	21	43	50	49	48	21	45	53	50	50	22	49	57	54	52	21	49	57	56	52	27	49	
Rovers-b	60	54	58	57	58	58	52	55	56	55	56	55	58	58	58	59	51	59	59	59	60	51	60	60	59	60	39	58	
TPP-b	60	37	49	49	40	34	29	42	44	39	31	34	47	51	39	34	30	50	53	43	34	29	49	54	42	32	16	33	
Σ RCP-b	180	114	153	160	150	142	102	140	150	143	135	110	150	162	147	143	103	158	169	156	146	101	158	171	157	144	82	140	
Pentesting Benchmarks																													
Pentest-b	90	68	70	70	70	54	64	67	68	70	53	57	67	67	67	52	58	66	66	66	51	61	68	69	70	54	48	40	
Pentest	15	10	10	10	10	10	9	9	9	10	10	8	9	9	9	9	8	8	8	8	8	9	9	9	9	9	7	9	
Σ Pentest	105	78	80	80	80	64	73	76	77	80	63	65	76	76	76	61	66	74	74	74	59	70	77	78	79	63	55	49	

Table 16.3.: Acyclic planning. MaxProb coverage for remaining search algorithms, including also the overall best DFHS variant (LABEL/BW/TIP), called “LILAO\*”. “M&S”: pruning via merge-and-shrink; “N” with size bound  $N = 50k$ , “∞” without size bound. “VI on DB”: VI run on reduced (deterministic-bisimulated) state space. Other abbreviations as in Table 16.2. Default tie-breaking strategy.

– which holds for the original domain but not for the limited-budget version. (We could not actually run PROB-PRP on the limited-budget domain version, as PROB-PRP does not natively support a budget, and hard-coding the budget into PPDDL resulted in encodings too large to pre-process.)

ExhAO\* is better than VI only in case of early termination on  $V^L(s_T) = 1$ , i.e., when a full-certainty policy is found before visiting the entire state space. This happened only very seldom, and primarily in Random-b. ExhAO\* is otherwise dominated by VI. Equipped with non-trivial value initialization, LRTDP|<sub>U</sub> and LILAO\*|<sub>U</sub> outperform AO\*|<sub>U</sub>, the latter suffering from the need of storing parent pointers for its update procedure, leading to a larger memory footprint. For the trivial value initialization ( $V^U = 1$  everywhere), memory consumption also becomes frequently an issue for the other two algorithms, making AO\*’s overhead comparatively less important. Overall, LILAO\*|<sub>U</sub> performs slightly better than LRTDP|<sub>U</sub>. The difference mainly comes from Blocksworld-b, the Canadian RCP benchmarks, and pentesting, whereas LRTDP|<sub>U</sub> has a slight edge in Elevators-b. IDUAL is not competitive with any of the other search algorithms, the overhead generated by the LP solver calls is typically prohibitive, and outclassed by the much faster numeric computations done in all other methods.

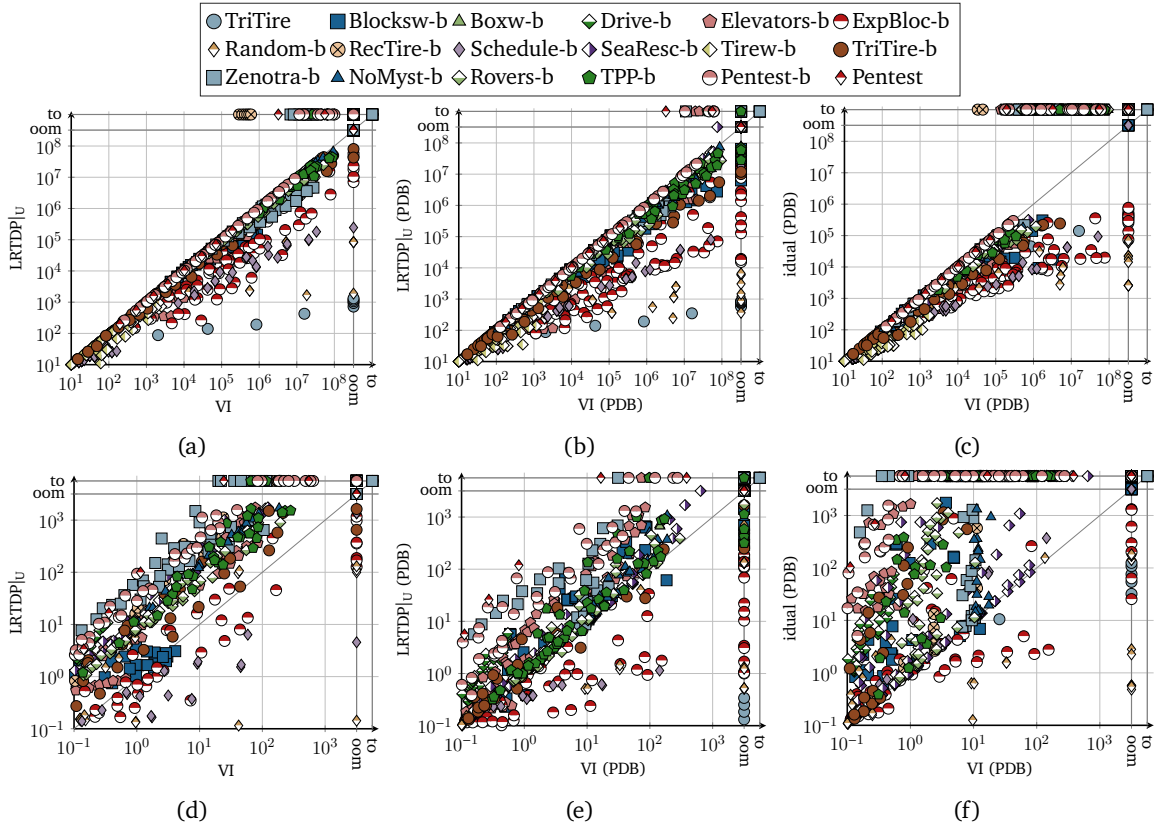


Figure 16.1.: Acyclic MaxProb planning. Per-instance comparison between VI ( $x$ -axes) and  $LRTDP|_U$  and  $IDUAL$  ( $y$ -axes), with and without pruning using PDB. Top: search space size (number of states visited). Bottom: runtime in seconds. Default tie-breaking strategy. “oom” represents out of memory; “to” out of time.

To gauge the efficiency of heuristic search for MaxProb, compare  $LRTDP|_U$  and VI in Table 16.3. With the trivial goal-probability initialization, the overall picture is mixed. While  $LRTDP|_U$  achieves higher coverage in 5 domains, it is worse in 9 other domains, and performs equally in the remaining 4 domains. In terms of total coverage, the per-domain differences cancel out, making  $LRTDP|_U$  and VI almost identical. Still, we find it remarkable that  $LRTDP|_U$  is able to improve upon VI at all, given the common perception that a good initial value estimator is required for heuristic search to be of any use. The advantage of heuristic search however becomes really apparent as the quality of the value initialization increases. Equipped with PDB,  $LRTDP|_U$  dominates VI in almost every domain. Moreover, comparing the respective coverage increases from the trivial bound (“-”) to PDB shows that  $LRTDP|_U$  is able to utilize the additional information much more effectively.

We next shed additional light on this by comparing search space sizes and runtime values. Tables 16.4 and 16.5 provide aggregate data, Figure 16.1 shows per-instance scatter plots for the comparison of VI vs.  $LRTDP|_U$ . Data for ExhAO\* is not shown as its coverage is dominated by VI (cf. Table 16.3), and the same goes for its runtime and search space. Moreover, we have omitted  $IDUAL$  from the tables, whose consideration would otherwise skew the commonly solved instance basis. A comparison between  $IDUAL$  and VI is available in Figure 16.1. We include the “ $\hookrightarrow$ (non-triv.)” rows in the tables to show behavior on the more interesting instances, where the averages are not biased by the many very small instances in most domains.

Domain	#	VI					AO*  <sub>U</sub>					LRTDP  <sub>U</sub>					LILAO*  <sub>U</sub>				
		–	LM	PDB	M&S		–	LM	PDB	M&S		–	LM	PDB	M&S		–	LM	PDB	M&S	
					N	∞				N	∞				N	∞				N	∞
IPPC Benchmarks																					
TriTire	5	3.37k	3.37k	3.37k	3.37k	3.37k	0.14	0.14	0.14	0.14	0.14	0.18	0.16	0.16	0.16	0.16	0.14	0.14	0.14	0.14	0.14
↳( <i>non-triv.</i> )	2	8.39k	8.39k	8.39k	8.39k	8.39k	0.22	0.22	0.22	0.22	0.22	0.31	0.27	0.27	0.27	0.27	0.23	0.22	0.22	0.22	0.22
IPPC Benchmarks with Budget Limit																					
Blocksw-b	54	0.15k	34.7	27.5	24.2	23.2	0.14k	28.5	22.1	19.2	18.4	40.5	7.47	6.43	4.76	4.43	37.3	5.98	5.28	3.79	3.47
↳( <i>non-triv.</i> )	10	0.57k	0.16k	0.12k	0.11k	0.11k	0.52k	0.13k	99.7	90.7	87.0	0.11k	24.5	20.2	17.1	15.5	97.8	17.9	15.2	12.6	11.0
Drive-b	48	1.73	1.11	1.47	0.83	0.79	1.71	0.99	1.37	0.74	0.7	1.67	0.99	1.32	0.74	0.69	1.66	0.94	1.29	0.69	0.65
Elevators-b	35	11.4	3.08	6.24	4.72	2.72	4.75	0.24	1.49	0.96	0.12	4.85	0.25	1.53	0.98	0.13	4.6	0.24	1.45	0.95	0.12
ExpBloc-b	53	3.76k	27.6	79.9	83.6	22.0	0.98k	0.54	1.36	3.03	0.41	0.4k	0.59	1.55	4.17	0.5	1.0k	0.56	1.51	4.57	0.43
↳( <i>non-triv.</i> )	15	13.0k	77.8	0.25k	0.27k	60.4	3.43k	1.19	3.52	9.46	0.86	1.38k	1.31	4.01	12.9	1.08	3.47k	1.21	3.89	14.1	0.88
Random-b	31	0.97k	0.18k	0.18k	0.17k	0.17k	12.6	2.1	2.09	2.07	2.07	12.8	3.22	3.2	3.19	3.19	12.8	2.25	2.23	2.22	2.22
↳( <i>non-triv.</i> )	3	9.6k	1.71k	1.69k	1.61k	1.61k	54.8	1.06	0.69	0.69	0.69	56.4	1.06	0.69	0.69	0.69	56.4	1.06	0.69	0.69	0.69
RecTire-b	25	22.7	2.51	0.96	0.96	0.96	22.3	2.25	0.72	0.72	0.72	22.6	2.27	0.76	0.76	0.76	22.6	2.24	0.72	0.72	0.72
↳( <i>non-triv.</i> )	8	55.2	6.06	2.29	2.29	2.29	54.8	5.41	1.69	1.69	1.69	55.0	5.47	1.79	1.79	1.79	54.9	5.38	1.68	1.68	1.68
Schedule-b	59	1.64k	0.4k	0.34k	0.33k	0.33k	55.1	2.99	1.72	1.35	1.27	13.2	3.02	2.3	2.09	2.05	18.5	2.33	1.7	1.36	1.28
↳( <i>non-triv.</i> )	11	8.65k	2.13k	1.8k	1.78k	1.77k	0.27k	14.7	8.73	6.76	6.36	62.2	14.8	11.6	10.6	10.3	86.7	11.3	8.61	6.82	6.41
SeaResc-b	71	3.28k	0.5k	0.32k	0.32k	0.32k	3.26k	0.4k	0.23k	0.24k	0.23k	3.26k	0.45k	0.27k	0.26k	0.26k	3.26k	0.32k	0.21k	0.21k	0.21k
↳( <i>non-triv.</i> )	31	7.42k	1.13k	0.71k	0.71k	0.71k	7.36k	0.88k	0.52k	0.53k	0.52k	7.38k	1.01k	0.6k	0.59k	0.59k	7.38k	0.72k	0.47k	0.47k	0.47k
Tirew-b	90	0.27	0.17	0.17	0.17	0.17	0.08	0.05	0.05	0.05	0.05	0.15	0.06	0.06	0.06	0.06	0.13	0.05	0.05	0.05	0.05
TriTire-b	49	5.77k	0.29k	0.29k	0.29k	0.29k	3.36k	47.6	47.6	47.6	47.6	2.61k	27.5	27.5	27.5	27.5	2.42k	27.1	27.1	27.1	27.1
↳( <i>non-triv.</i> )	15	18.7k	0.94k	0.94k	0.94k	0.94k	10.9k	0.15k	0.15k	0.15k	0.15k	8.42k	87.7	87.7	87.7	87.7	7.8k	86.8	86.8	86.8	86.8
Zenotra-b	34	4.13k	0.74k	0.61k	0.97k	0.44k	4.13k	0.73k	0.61k	0.97k	0.44k	1.31k	0.22k	0.21k	0.31k	0.14k	1.59k	0.27k	0.25k	0.37k	0.18k
↳( <i>non-triv.</i> )	26	5.34k	0.96k	0.78k	1.27k	0.58k	5.34k	0.96k	0.77k	1.26k	0.57k	1.67k	0.29k	0.25k	0.4k	0.18k	2.03k	0.34k	0.31k	0.48k	0.23k
Canadian RCP Benchmarks with Budget Limit																					
NoMyst-b	21	15.4k	0.16k	31.4	32.0	31.4	12.7k	0.12k	19.4	20.0	19.4	13.7k	0.12k	17.9	18.5	17.9	13.6k	0.12k	17.7	18.2	17.7
↳( <i>non-triv.</i> )	21	15.4k	0.16k	31.4	32.0	31.4	12.7k	0.12k	19.4	20.0	19.4	13.7k	0.12k	17.9	18.5	17.9	13.6k	0.12k	17.7	18.2	17.7
Rovers-b	51	3.38k	1.04k	1.3k	1.01k	0.9k	1.68k	0.3k	0.46k	0.29k	0.22k	2.3k	0.37k	0.57k	0.37k	0.27k	2.06k	0.32k	0.51k	0.32k	0.24k
↳( <i>non-triv.</i> )	30	5.67k	1.75k	2.2k	1.7k	1.53k	2.8k	0.49k	0.77k	0.49k	0.36k	3.85k	0.62k	0.96k	0.63k	0.46k	3.44k	0.53k	0.85k	0.54k	0.39k
TPP-b	19	8.78k	0.85k	0.89k	4.1k	0.43k	4.92k	0.36k	0.28k	2.3k	93.9	6.33k	0.39k	0.35k	2.85k	0.11k	5.77k	0.37k	0.29k	2.65k	95.1
↳( <i>non-triv.</i> )	16	10.4k	1.0k	1.05k	4.85k	0.51k	5.81k	0.42k	0.33k	2.71k	0.11k	7.47k	0.46k	0.41k	3.37k	0.13k	6.81k	0.43k	0.34k	3.13k	0.11k
Pentesting Benchmarks																					
Pentest-b	49	0.3k	0.16k	0.2k	0.17k	0.16k	0.3k	0.16k	0.2k	0.17k	0.16k	0.3k	0.16k	0.2k	0.17k	0.16k	0.3k	0.16k	0.2k	0.17k	0.16k
↳( <i>non-triv.</i> )	11	1.25k	0.68k	0.83k	0.71k	0.68k	1.24k	0.68k	0.83k	0.71k	0.68k	1.25k	0.68k	0.83k	0.71k	0.68k	1.25k	0.68k	0.83k	0.71k	0.68k
Pentest	8	38.9	34.7	38.9	34.7	34.7	38.9	34.7	38.9	34.7	34.7	38.9	34.7	38.9	34.7	34.7	38.9	34.7	38.9	34.7	34.7
↳( <i>non-triv.</i> )	1	0.19k	0.17k	0.19k	0.17k	0.17k	0.19k	0.17k	0.19k	0.17k	0.17k	0.19k	0.17k	0.19k	0.17k	0.17k	0.19k	0.17k	0.19k	0.17k	0.17k

Table 16.4.: Acyclic MaxProb planning. Per-domain average search space size (number of states visited) data in multiples of 1000. “#” gives the size of the instance basis, namely those instances solved by all shown configurations. “↳(*non-triv.*)” considers of those only instances not solved by VI in < 1 second. Rows with empty instance basis are skipped. “k” multiples of 1000. Default tie-breaking strategy.

A clear message from Table 16.4 and Figure 16.1 is that the heuristic search algorithms, apart from a few exceptions, visit much fewer states than VI does. Even with the trivial upper bound initialization, search spaces are reduced in all domains except RectangleTireworld-b, SearchAndRescue-b, and Pentest.

		VI					AO*  <sub>U</sub>					LRTDP  <sub>U</sub>					LILAO*  <sub>U</sub>				
Domain	#	–	LM	PDB	M&S N	∞	–	LM	PDB	M&S N	∞	–	LM	PDB	M&S N	∞	–	LM	PDB	M&S N	∞
IPPC Benchmarks																					
TriTire	5	5.08	30.1	5.41	10.1	9.79	0.0	0.01	0.03	0.03	0.03	0.0	0.01	0.03	0.03	0.03	0.01	0.01	0.04	0.03	0.03
$\mathfrak{b}(\text{non-triv.})$	2	12.7	75.0	13.5	25.1	24.4	0.01	0.01	0.05	0.06	0.06	0.01	0.01	0.05	0.06	0.06	0.01	0.02	0.06	0.07	0.07
IPPC Benchmarks with Budget Limit																					
Blocksw-b	54	0.6	1.04	0.12	0.6	0.45	5.64	1.53	0.6	0.96	0.8	1.12	0.29	0.11	0.56	0.41	0.62	0.21	0.09	0.52	0.4
$\mathfrak{b}(\text{non-triv.})$	10	2.42	4.74	0.39	1.89	1.39	22.8	7.1	2.63	3.71	3.08	2.36	0.92	0.24	1.61	1.15	1.45	0.58	0.16	1.49	1.09
Drive-b	48	0.01	0.02	0.02	1.31	29.2	0.04	0.03	0.04	1.3	29.1	0.04	0.02	0.04	1.27	28.7	0.04	0.02	0.03	1.25	28.4
Elevators-b	35	0.03	0.02	0.06	2.24	46.7	0.04	0.01	0.06	2.22	47.4	0.1	0.01	0.07	2.11	46.5	0.2	0.01	0.09	2.11	46.0
ExpBloc-b	53	7.6	0.27	1.47	8.82	54.3	15.2	0.01	1.37	8.11	53.9	10.2	0.01	1.38	7.88	52.1	17.7	0.02	1.38	7.81	52.4
$\mathfrak{b}(\text{non-triv.})$	15	26.3	0.77	0.87	24.9	158.1	53.4	0.03	0.49	22.5	157.1	35.6	0.03	0.49	21.9	151.0	61.9	0.03	0.49	21.7	152.5
Random-b	31	4.26	18.2	0.53	10.3	17.7	6.47	0.54	0.21	8.19	14.1	7.26	0.56	0.21	7.64	14.5	7.5	0.53	0.19	7.28	14.1
$\mathfrak{b}(\text{non-triv.})$	3	42.0	175.5	4.15	95.6	171.9	31.2	0.84	0.86	74.4	135.5	67.6	0.84	0.9	69.0	140.0	71.4	0.84	0.85	65.4	135.2
RecTire-b	25	1.69	1.19	0.38	0.28	0.28	3.71	1.34	0.4	0.29	0.29	57.9	1.37	0.4	0.29	0.29	187.4	1.33	0.39	0.28	0.29
$\mathfrak{b}(\text{non-triv.})$	8	4.73	3.45	0.93	0.71	0.72	10.5	3.89	0.98	0.76	0.75	174.3	3.98	0.98	0.74	0.75	553.5	3.85	0.96	0.73	0.74
Schedule-b	59	3.21	21.0	0.92	7.56	48.5	0.67	0.24	0.18	5.93	45.0	0.31	0.21	0.18	5.84	44.6	0.37	0.19	0.19	5.74	44.2
$\mathfrak{b}(\text{non-triv.})$	11	17.0	111.8	4.39	35.0	256.7	3.36	1.19	0.42	26.6	237.9	1.47	1.04	0.46	26.1	235.7	1.76	0.91	0.49	25.6	233.5
SeaResc-b	71	11.0	30.6	9.28	2.28	2.3	109.3	25.7	11.1	4.0	3.87	173.7	31.8	10.7	3.68	3.72	109.2	21.1	9.83	2.8	2.81
$\mathfrak{b}(\text{non-triv.})$	31	24.9	69.8	19.8	5.11	5.14	248.7	58.5	23.8	8.97	8.67	395.1	72.3	22.9	8.23	8.34	248.2	47.9	21.0	6.23	6.27
Tirew-b	90	0.01	0.01	0.03	0.01	0.01	0.01	0.01	0.03	0.01	0.01	0.01	0.01	0.03	0.01	0.01	0.01	0.01	0.03	0.01	0.01
TriTire-b	49	12.1	4.04	0.53	1.71	1.56	55.9	1.5	0.56	0.81	0.82	65.8	1.12	0.5	0.65	0.68	72.9	0.98	0.37	0.53	0.54
$\mathfrak{b}(\text{non-triv.})$	15	39.2	13.1	1.62	5.51	5.03	181.5	4.86	1.72	2.6	2.63	212.9	3.59	1.5	2.05	2.15	235.9	3.16	1.08	1.67	1.69
Zenotra-b	34	13.0	69.9	7.03	5.52	9.74	104.4	79.9	16.0	20.7	14.5	297.8	32.3	22.6	25.2	15.1	62.2	27.1	10.1	10.4	11.4
$\mathfrak{b}(\text{non-triv.})$	26	16.9	91.3	8.52	6.64	11.6	135.2	104.3	19.8	26.5	17.7	382.2	42.0	26.6	32.4	18.5	79.4	35.3	12.2	13.0	13.7
Canadian RCP Benchmarks with Budget Limit																					
NoMyst-b	21	27.4	2.04	11.9	0.58	0.74	128.9	1.84	12.3	0.6	0.76	383.6	2.07	12.2	0.63	0.77	509.0	2.48	12.4	0.71	0.83
$\mathfrak{b}(\text{non-triv.})$	21	27.4	2.04	11.9	0.58	0.74	128.9	1.84	12.3	0.6	0.76	383.6	2.07	12.2	0.63	0.77	509.0	2.48	12.4	0.71	0.83
Rovers-b	51	12.8	6.46	5.89	3.97	5.02	28.0	3.31	6.03	3.07	3.61	95.3	5.63	9.44	5.36	5.09	108.7	6.26	10.9	6.29	5.69
$\mathfrak{b}(\text{non-triv.})$	30	21.6	10.8	8.83	6.62	8.4	46.8	5.5	9.02	5.07	5.99	160.1	9.44	14.8	8.97	8.5	182.2	10.5	17.2	10.5	9.49
TPP-b	19	21.8	8.15	2.49	13.4	187.2	88.9	4.96	2.56	25.1	184.6	285.1	6.79	3.79	59.8	183.9	312.0	8.27	4.38	77.9	182.7
$\mathfrak{b}(\text{non-triv.})$	16	25.8	9.62	2.72	15.5	216.7	105.2	5.85	2.81	29.3	213.7	337.5	8.02	4.27	70.5	213.2	369.3	9.78	4.96	91.9	211.7
Pentesting Benchmarks																					
Pentest-b	49	1.48	1.86	0.67	7.49	113.1	62.8	11.5	13.3	17.5	122.4	83.5	21.5	26.0	27.1	130.4	15.3	4.16	3.47	9.39	112.9
$\mathfrak{b}(\text{non-triv.})$	11	6.21	7.83	2.75	12.8	180.2	268.5	48.8	56.4	55.6	222.1	358.5	91.1	110.0	97.2	261.7	64.7	17.6	14.6	22.0	184.8
Pentest	8	0.22	0.24	0.17	0.2	0.2	9.56	3.61	5.38	3.54	3.58	15.0	10.4	19.0	10.5	10.5	1.27	0.77	0.98	0.72	0.76
$\mathfrak{b}(\text{non-triv.})$	1	1.15	1.23	0.82	0.97	0.97	56.8	20.7	31.3	20.1	20.3	99.5	64.7	119.0	65.3	64.8	7.15	4.16	5.2	3.88	4.17

Table 16.5.: Acyclic MaxProb planning. Per-domain average runtime (in seconds) over commonly solved instances. Same setup and presentation as in Table 16.4.

Figure 16.1a gives a picture of the impact of heuristic search, exemplified again through LRTDP<sub>|U</sub>. Observe that LRTDP<sub>|U</sub>'s search space is 1 order of magnitude smaller than that by VI in many instances, and larger gains (up to 4 orders of magnitude) also occur in rare cases. As portrayed in Figure 16.1b, when providing heuristic search additional information through dead-end detection, the differences become even larger.

These observations have not been made in this clarity before. While Kolobov et al. also report LRTDP to

beat VI on MaxProb, they consider only a single domain; they do not experiment with trivially initialized  $V^U$ ; and they do not use dead-end pruning in VI, so that LRTDP already benefits from a smaller state space, and the impact of heuristic search remains unclear.

Even though the search space of the heuristic search algorithms is in many cases only a small fraction of the whole (dead-end pruned) state space, this is not necessarily reflected in runtime. On those instances solved by VI, it is typically fast, often faster than heuristic search and rarely outperformed significantly. This is despite having larger search spaces, i.e., heuristic search does visit less states but suffers from having to do more updates on these (recall that VI here updates each visited state exactly once). Significant runtime advantages over VI are obtained by heuristic search only in TriangleTireworld, ExplodingBlocks-b, and Random-b.

Comparing the heuristic search algorithms, the conclusions are more fine-grained but overall similar to what we concluded from coverage above. Like the other heuristic search algorithms, IDUAL visits orders of magnitudes fewer states than VI (cf. Figure 16.1c), yet that advantage is completely overshadowed by the computational overhead of solving the LPs (cf. Figure 16.1f). The remaining heuristic search algorithms perform very similarly overall, yet there are significant differences in some domains. The volatility between the algorithms is largest if no dead-end pruning is used. In almost every domain, one of the three shown algorithms stands out, either positively or negatively. The most striking examples are Blocksworld-b, where  $AO^*_U$  visits significantly more states than  $LRTDP_U$  and  $LILAO^*_U$ ; ExplodingBlocks-b where  $LRTDP_U$  excels; Zenotravel-b where, despite the smaller search space size,  $LRTDP_U$  is dominated by  $AO^*_U$ , which in turn is dominated by  $LILAO^*_U$ ; and the Canadian RCP benchmarks, on which  $LRTDP_U$  and  $LILAO^*_U$  struggle.  $AO^*_U$ 's runtime advantages stem from its update procedure, which propagates value changes more effectively, in total requiring fewer updates necessary until termination. Across the non-trivial commonly solved instances in the tables, the average number of updates done in  $AO^*_U$  is about 3 times smaller than that in  $LRTDP_U$ , and about 2.5 times smaller than in  $LILAO^*_U$ . This advantage is hidden in coverage due to the larger memory demand. The difference between the algorithms diminishes as value initialization becomes better. Comparing the PDB configurations,  $LILAO^*_U$  and  $LRTDP_U$  have similar search space sizes in almost all domains, yet  $LILAO^*_U$  is able to propagate the dead-end information more effectively. Over the non-trivial instances,  $LILAO^*_U$  needs only about half as much value updates as  $LRTDP_U$ , which is reflected on runtime.  $AO^*_U$  utilizes the heuristic not quite as well as  $LRTDP_U$  and  $LILAO^*_U$ .  $AO^*_U$ 's search space is almost consistently larger on the limited-budget IPPC benchmarks. Only on the RCP benchmarks,  $AO^*_U$  is able to keep its runtime lead.

The impact of dead-end pruning on VI is typically moderate. The gains for heuristic search are much more pronounced. Comparing across different dead-end pruning methods, although M&S with  $N = \infty$  clearly yields the largest search space reductions, and necessarily so as it recognizes *all* dead-ends, the overhead of the bisimulation computation outweighs the search space reduction in all but a few cases. In terms of pruning power, the different heuristics have strengths in different domains. LMcut is especially beneficial in Elevators-b and ExplodingBlocks-b; M&S with  $N = 50k$  holds the lead in Blocksworld-b and Schedule-b. Even though the PDB configurations stand out in terms of search space size in only TPP-b, overall it offers the best trade-off between accuracy and overhead, being less expensive to construct than M&S and more efficient to evaluate than LMcut, while the search space sizes are usually close to the best of LMcut and M&S.

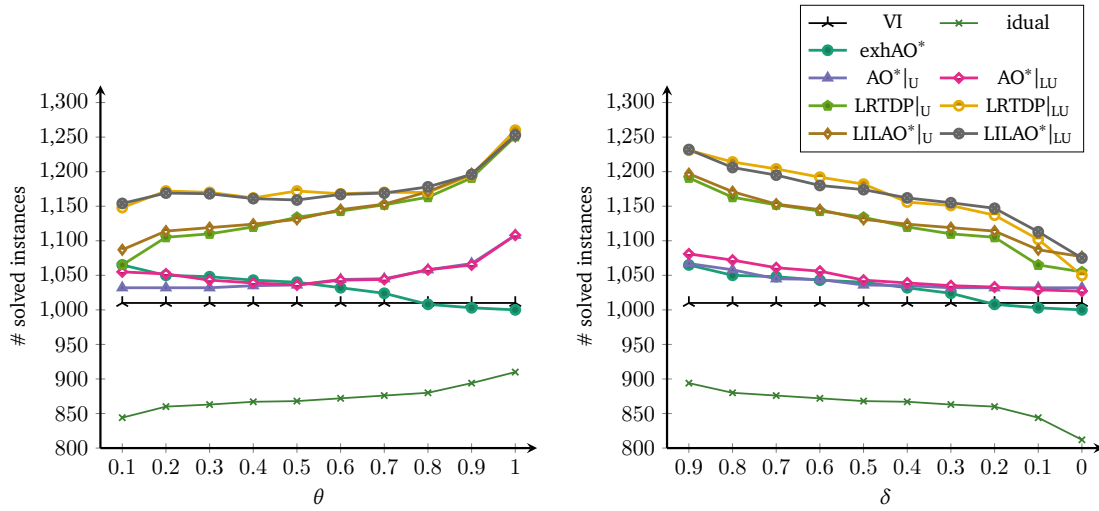


Figure 16.2.: Acyclic planning. Total coverage for (left) AtLeastProb as a function of  $\theta$ , and (right) for ApproxProb as a function of  $\delta$ . All configurations use the default tie-breaking strategy and PDB dead-end pruning.

#### AtLeastProb and ApproxProb Objective Parameter Analysis

We now turn to the weaker objectives, AtLeastProb and ApproxProb. We fix PDB as the (almost always most effective) dead-end pruning technique. We examine the power of early termination for different search algorithms and tie-breaking strategies. This is best viewed as a function of the goal-probability threshold  $\theta$  in AtLeastProb, and of the desired goal probability accuracy  $\delta$  in ApproxProb. VI forms a baseline independent of  $\theta$  ( $\delta$ ). Consider Figure 16.2.

For AtLeastProb (the left plot in Figure 16.2), in the interesting region of benchmark instances not feasible for VI yet sometimes feasible for the other search algorithms, one clear feature is the superiority of LRTDP and LILAO\*. ExhAO\* exhibits a strikingly strong behavior for small values of  $\theta$ , approaching (and in one case even surpassing) the performance of LRTDP|<sub>U</sub>. It is in particular also more effective than AO\*|<sub>LU</sub>. Evidently, the depth-first expansion strategy is quite effective for anytime behavior on  $V^L$  and thus for termination via  $V^L(s_I) \geq \theta$ . In general, for all algorithms, using  $V^L$  is a clear advantage for small  $\theta$ . For larger  $\theta$ , maintaining  $V^L$  can become a burden, yet  $V^U$  is of advantage due to early termination on  $V^U(s_I) < \theta$ .

The AtLeastProb performance behavior of algorithms using both bounds resemble an easy-hard-easy pattern. The spike at the left-hand side in Figure 16.2 (left), i.e., significantly worse performance for  $\theta = 0.1$  than for  $\theta = 0.2$ , is an outlier due to the Pentest domains. This is because, in contrast to typical probabilistic planning scenarios, in penetration testing the goal probability – the chance of a successful attack – are typically small, and indeed this is so in our benchmarks. Searches using an upper bound quickly obtain  $V^U(s_I) < 0.2$ , terminating early based on  $V^U(s_I) < \theta$  for  $\theta = 0.2$ . But it takes a long time to obtain  $V^U(s_I) < 0.1$ .

For ApproxProb (right plot in Figure 16.2), smaller values of  $\delta$  consistently result in worse performance. We see again the superiority of LRTDP and LILAO\*, and a competitive behavior of ExhAO\* compared to AO\*|<sub>LU</sub> in  $\delta$  regions allowing aggressive early termination. Again, the key to LRTDP and LILAO\*'s performance is their effective updates of  $V^L$ . Finally, we also see again the superiority of algorithms using both bounds over those that don't.

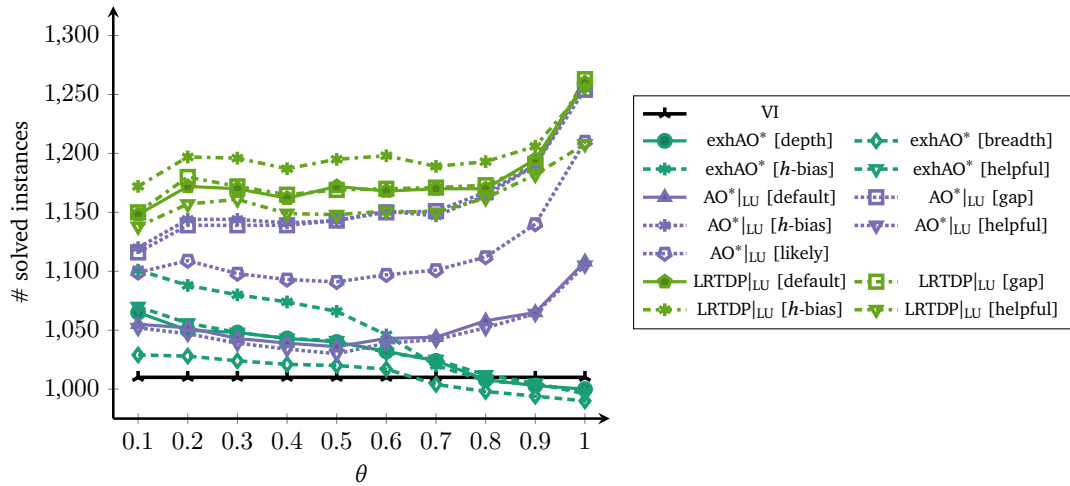


Figure 16.3.: Acyclic planning. Total coverage for AtLeastProb as a function of  $\theta$ , varying the tie-breaking strategy. Abbreviations: “depth” depth-first exploration; “breadth” breadth-first exploration; “ $h$ -bias”  $h$ -bias strategy; “helpful” helpful actions strategy; “gap” gap-bias strategy; “likely” most-likely outcome bias. All configurations use PDB dead-end pruning.

### Tie-Breaking Strategies

Figure 16.3 shows different tie-breaking strategies in AtLeastProb. The relative performance is the same in ApproxProb, so we do not include a separate figure for that.

For the sake of readability, we show only the most competitive base algorithms, ExhAO\*, AO\* and LRTDP using upper and lower bounds, as well as the VI baseline. We omit LILAO\*<sub>LU</sub> whose performance across the different tie-breaking strategies is identical to LRTDP<sub>LU</sub>. For ExhAO\*, we see that a depth-biased exploration is important, the breadth-first variant lacks behind consistently. Especially for small values of  $\theta$ , the depth-bias allows to raise  $V^L(s_I)$  much more effectively, fostering early termination. The helpful-actions strategy has no impact on the development of the bounds for any algorithm, but the additional overhead negatively (if at all) affects the results. The  $h$ -bias results in the by far biggest improvements over the default strategy across all the search algorithms, guiding the explorations to goal states quickly, and with that fostering the development of the  $V^L$  bounds. For AO\*<sub>LU</sub>, the gap-bias strategy is almost as beneficial as the  $h$ -bias strategy. The most-prob-outcome bias consistently improves coverage across the board as well, yet not quite as much as the other two strategies. For LRTDP<sub>LU</sub>, the gap-bias strategy resulted in almost identical behavior compared to the default strategy.

### An Illustration of Typical Anytime Behavior

To conclude our discussion of acyclic planning, Figure 16.4 exemplifies typical anytime behavior, i.e., the development of the  $V^L(s_I)$  and  $V^U(s_I)$  bounds on the initial state value, as a function of runtime, for LRTDP<sub>LU</sub> and ExhAO\*. The behavior for AO\*<sub>LU</sub> and LILAO\*<sub>LU</sub> is very similar to LRTDP<sub>LU</sub>.

The benefit of PDB pruning is evident. Observe that ExhAO\* is way more effective than LRTDP in quickly improving the lower bound. Indeed, the runs shown here find an optimal policy very quickly. Across the benchmarks solved by both ExhAO\* and LRTDP, omitting those where both took  $< 1$  second, in about 50% of cases ExhAO\* finds an optimal policy faster than LRTDP. On the downside, unless  $V^*(s_I) \geq \theta$ , ExhAO\* must explore the entire state space. In summary, heuristic search is much stronger in proving that

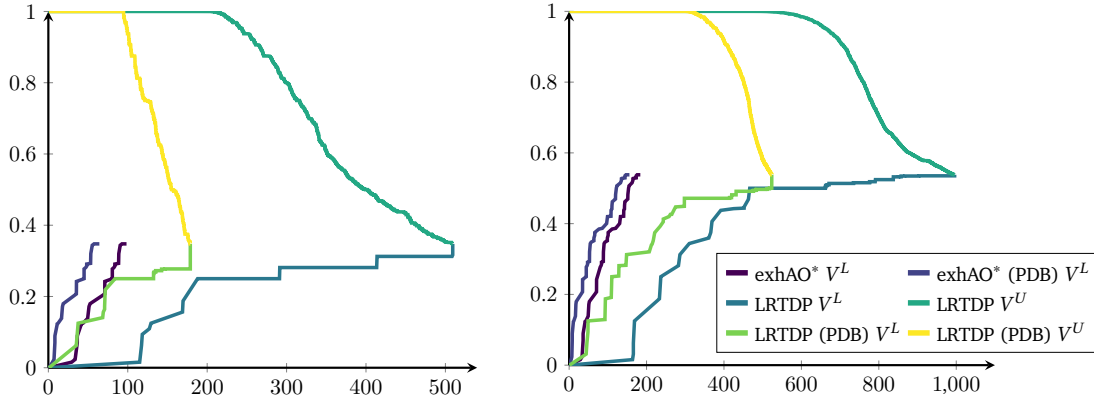


Figure 16.4.: Acyclic planning. Anytime behavior in  $\text{LRTDP}|_{\text{LU}}$  ( $V^U$  and  $V^L$ ) and  $\text{ExhAO}^*$  ( $V^L$  only), as a function of runtime (in seconds). Elevators instance 11, without pruning and with PDB pruning, for constrainedness level  $\mathfrak{C} = 1.4$  (left) respectively  $\mathfrak{C} = 1.8$  (right). Default tie-breaking strategy.

the maximum goal probability is found, but can be distracting for improving  $V^L$  quickly.

As both parts of Figure 16.4 use the same base instance but with different constrainedness levels  $\mathfrak{C}$ , we can also draw conclusions on the effect of surplus budget. With more budget, more actions can be applied before reaching terminal states. This adversely affects the upper bound (consistently across our experiments), which takes a much longer time to decrease. The lower bound, on the other hand, often increases more quickly with higher  $\mathfrak{C}$  as it is easier to find goal states.

### 16.3.2. Cyclic Planning

The cyclic part of our benchmark suite comprises the standard IPPC benchmarks (except TriangleTire-world), as well as NoMystery, Rovers, and TPP without budget limit.

In terms of algorithms, we evaluate LRTDP and DFHS in combination with both FRET variants. Furthermore, we consider the trap-aware (TA) LRTDP and DFHS variants, as well as  $\text{idUAL}$ , which support cyclic planning without dedicated FRET outer loop. We do not run  $\text{AO}^*$ , as it is restricted to acyclic state spaces, and for the same reason substitute  $\text{ExhAO}^*$  by  $\text{ExhDFS}$ . As baseline, we consider topological VI. We consider five dead-end pruning methods as before. We replace  $\text{LMcut}$  by  $h^{\text{FF}}$  (Hoffmann and Nebel, 2001), which identifies the exact same dead ends but often yields more informative cost-to-goal estimates (useful for tie-breaking). We did not use  $h^{\text{FF}}$  before because its cost-to-goal bounds (except for dead-end detection) are not guaranteed to be admissible, and thus not suited for pruning against the remaining budget. Each heuristic search algorithm is tested with up to 4 different tie-breaking strategies. We use the deterministic-bisimulation (DB) reduced state space again with VI. Given the deterministic bisimulation, additional dead-end pruning is not needed.

We consider all three goal-probability objectives. Running all parameter combinations would yield a total of 4791 possible configurations. As before, not all of these are interesting, and we instead organize our experiment in terms of parts focusing on issues of interest. We follow roughly the same structure as in the acyclic part. Specifically, we have parts i. on MaxProb, ii. on AtLeastProb and ApproxProb objective parameters, and iii. on tie-breaking strategies. We combine the discussion of ii. and iii. into a single subsection below, and integrate an illustration of anytime behavior alongside. Table 16.6 provides an overview

	# Total
<b>i. Search Algorithms &amp; Pruning in MaxProb</b>	241
Search algorithms: • VI; • ExhDFS; • LRTDP <sub> U</sub> (×3); • DFHS <sub> U</sub> (×14×3); • IDUAL; and • VI on DB	
Dead-end pruning: • disabled; • $h^{\text{FF}}$ ; • PDB; and • M&S with $N \in \{50k, \infty\}$	
Tie-breaking: default	
<b>ii. AtLeastProb and ApproxProb Objective Parameters</b>	29
Search algorithms: • VI; • ExhDFS; • LRTDP <sub> U</sub> (×3); • LRTDP <sub> LU</sub> (×3); • DFHS <sub> U</sub> (×1, best from i., ×3); • DFHS <sub> LU</sub> (×1×3); and • IDUAL	
Dead-end pruning: $h^{\text{FF}}$	
Tie-breaking: default	
<b>iii. Tie-Breaking Strategies on AtLeastProb and ApproxProb</b>	93
Search algorithms: • VI; • ExhDFS; • LRTDP <sub> U</sub> (×3); • LRTDP <sub> LU</sub> (×3); • DFHS <sub> U</sub> (×1×3); • DFHS <sub> LU</sub> (×1×3); and • IDUAL	
Dead-end pruning: $h^{\text{FF}}$	
Tie-breaking: <i>all</i> (• 1; • 3; • 3; • 4; • 3; • 4; and • 1)	

Table 16.6.: Overview of experiments on the cyclic benchmark part. LRTDP and DFHS are executed in three variants: wrapped in FRET- $V$  vs. wrapped in FRET- $\pi$  vs. using the trap-aware (TA) extensions. In ii. and iii., the heuristic search configurations are doubled because AtLeastProb vs. ApproxProb result in different algorithm configurations (using different termination criteria).

of tested configurations.

### Search Algorithms & Pruning Methods in MaxProb

We evaluate the trap-aware versions of LRTDP and DFHS separately at the end of this sub-section, and for the time being concentrate on the more fundamental difference between the two FRET variants. Table 16.7 and Table 16.8 provide MaxProb coverage results for the remaining algorithm configurations. More specifically, Table 16.7 shows the results for a selection of different DFHS instantiations. A comparison of all variants is available in Appendix A.2. For space reasons, we do not include results for M&S pruning. Moreover, as there were almost no coverage differences between the DFHS configurations for FRET- $V$ , we do not vary the FRET variant here. Table 16.8 complements Table 16.7, showing results for all pruning functions, the remaining search algorithms, and both FRET variants. For convenience, we also include the overall best-performing DFHS configuration, HDP. We omit the M&S results for IDUAL whose performance relative to the other dead-end detection methods is identical to the other heuristic search algorithms.

Domain	#	FRET- $\pi$ DFHS  $\cup$																	
		VI									LABEL								
		BW			FW			BW			FW			TIP $\cup$ INC					
		NONE	TIP	PDB	NONE	INC	PDB	NONE	TIP	PDB	NONE	INC	PDB	TIP $\cup$ INC	PDB				
		- $h^{FF}$	- $h^{FF}$	PDB	- $h^{FF}$	- $h^{FF}$	PDB	- $h^{FF}$	- $h^{FF}$	PDB	- $h^{FF}$	- $h^{FF}$	PDB	- $h^{FF}$	- $h^{FF}$	PDB	- $h^{FF}$	- $h^{FF}$	PDB
IPPC Benchmarks																			
Blocksw	30	9	<b>22</b>	17	9	16	14	9	<b>22</b>	18	9	<b>22</b>	18	9	16	14	9	<b>22</b>	18
Boxw	15	0	<b>7</b>	5	0	5	4	0	<b>7</b>	5	0	<b>7</b>	5	0	5	4	0	<b>7</b>	5
Drive	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
Elevators	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
ExpBloc	30	10	25	16	10	26	16	10	25	16	10	26	16	10	25	16	10	<b>28</b>	16
Random	15	5	<b>14</b>	11	5	13	11	5	<b>14</b>	11	5	<b>14</b>	11	5	<b>14</b>	11	5	<b>14</b>	11
RecTire	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14
Schedule	30	6	<b>10</b>	<b>10</b>	6	<b>10</b>	<b>10</b>	6	<b>10</b>	<b>10</b>	6	<b>10</b>	<b>10</b>	6	<b>10</b>	<b>10</b>	6	<b>10</b>	<b>10</b>
SeaResc	15	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
Tirew	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
Zenotra	30	4	<b>10</b>	8	4	8	8	4	<b>10</b>	9	4	<b>10</b>	9	4	<b>10</b>	9	4	<b>10</b>	9
$\Sigma$ IPPC	224	98	152	131	98	142	127	98	152	133	98	153	133	98	152	133	98	<b>156</b>	134
Canadian RCP Benchmarks																			
NoMyst	10	4	<b>5</b>	<b>5</b>	0	4	4	4	<b>5</b>	<b>5</b>	4	<b>5</b>	<b>5</b>	4	<b>5</b>	<b>5</b>	4	<b>5</b>	<b>5</b>
Rovers	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
TPP	10	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	<b>9</b>	<b>9</b>	8
$\Sigma$ RCP	30	22	23	23	18	22	22	22	23	23	22	23	23	22	23	23	22	<b>24</b>	<b>24</b>

Table 16.7.: Cyclic planning. MaxProb coverage for a representative selection of DFHS configurations; showing only the dominating FRET version, FRET- $\pi^U$ . Best results in **bold**. DFHS parameters abbreviated as in Table 16.2. ILAO\* corresponds to the configuration VI/BW/TIP; HDP corresponds to LABEL/FW/INC; and our new LILAO\* variant to LABEL/BW/TIP. Dead-end pruning variants: “-” none, else based on  $h = \infty$ . Default tie-breaking strategy.

Consider first the results for the different DFHS| $\cup$  variants in Table 16.7. Differences are even less pronounced than in the acyclic case. Yet, similarly to before, the LABEL based termination check has a slight advantage over termination by VI. In contrast to the acyclic case, cutting off the exploration at tip states is detrimental overall, with negative effects on coverage in Blocksworld, Boxworld, Random, Zenotravel, and NoMystery. It turns out that the four former domains actually contain no dead ends. In this case, one DFHS run until convergence does nothing but completely exploring a single policy; values do not change, and traps are removed only after convergence. The cutoffs merely increase the number of (mostly redundant) iterations until the policy is fully explored. Cutting off the exploration at  $\epsilon$ -inconsistent states does not suffer from the same problem, as there are no inconsistent states, so it never applies. On the domains with dead ends, the cutoffs can however still be useful. This is most notable in ExplodingBlocks. The additional FW updates have a very slim advantage over doing backward updates (BW) only; its main benefit being the support of cutoffs at  $\epsilon$ -inconsistent states. Given again the marginal differences between the different DFHS variants, we will consider a single representative in the remaining discussion. Contrary to the acyclic case, we choose HDP, which avoids the TIP cutoff bottleneck.

Consider now Table 16.8. Thanks to early termination, ExhDFS is able to improve coverage over VI in 4 out of the 14 domains. Running VI on the deterministic-bisimulation reduced state space was not really effective in any case. Akin to the acyclic case, both LRTDP| $\cup$  and the chosen DFHS| $\cup$  variant perform equally well; and IDUAL cannot really hold on to the performance of the other heuristic search algorithms. But, the most striking result here by far is that FRET- $\pi$  significantly outperforms its competitors. While

Domain	#	VI				exhDFS				FRET-V								FRET- $\pi$								idual				VI			
		$h^{\text{FF}}$		PDB		M&S		$h^{\text{FF}}$		PDB		M&S		$h^{\text{FF}}$		PDB		M&S		$h^{\text{FF}}$		PDB		M&S		$h^{\text{FF}}$		PDB			M&S		
		$N$	$\infty$	$N$	$\infty$	$N$	$\infty$	$N$	$\infty$	$N$	$\infty$	$N$	$\infty$	$N$	$\infty$	$N$	$\infty$	$N$	$\infty$	$N$	$\infty$	$N$	$\infty$	$N$	$\infty$	$N$	$\infty$	$N$	$\infty$		on DB		
IPPC Benchmarks																																	
Blocksw	30	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	22	18	12	9	9	22	18	12	9	9	9	9	9	9	
Boxw	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	5	0	0	0	7	5	0	0	0	0	0	0	0	
Drive	15	15	15	15	15	8	15	15	15	15	15	8	15	15	15	15	8	15	15	15	15	8	15	15	15	15	8	15	15	15	15	8	
Elevators	15	15	15	15	15	5	15	15	15	15	15	5	15	15	15	15	5	15	15	15	15	5	15	15	15	15	5	15	15	15	15	5	
ExpBloc	30	9	11	10	10	7	9	10	10	10	7	10	14	12	10	7	10	14	12	10	7	10	26	16	10	7	10	28	16	11	7	8	
Random	15	1	1	1	1	1	2	2	2	2	1	5	2	4	3	1	5	2	4	3	1	5	14	11	5	1	5	14	11	5	1	5	
RecTire	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	
Schedule	30	7	7	7	7	7	6	6	6	6	6	6	10	10	10	7	6	10	10	10	7	6	10	10	10	7	6	10	10	10	7	7	
SeaResc	15	6	6	6	6	6	6	6	6	6	6	6	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	6	6	6	6	3	
Tirew	15	14	14	14	14	12	15	15	15	15	12	15	15	15	15	12	15	15	15	15	12	15	15	15	15	12	15	15	15	15	12	15	
Zenotra	30	7	7	7	7	7	8	8	8	8	8	8	4	8	6	7	8	4	8	6	7	8	4	10	9	8	8	4	10	9	8	8	5
$\Sigma$ IPPC	224	97	99	98	98	76	99	100	100	100	76	98	107	105	103	76	98	107	105	103	76	98	153	133	109	76	98	156	134	111	77	94	
Canadian RCP Benchmarks																																	
NoMyst	10	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	4	5	5	5	5	4	5	5	5	5	4	5	5	5	5	0	
Rovers	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	8	
TPP	10	8	8	8	8	7	10	10	10	10	7	6	6	6	6	6	6	6	6	6	6	8	9	9	8	6	8	9	9	8	6	2	
$\Sigma$ RCP	30	23	23	23	23	22	25	25	25	25	22	21	21	21	21	21	20	21	21	21	21	22	24	24	23	21	22	24	24	23	21	10	

Table 16.8.: Cyclic planning. MaxProb coverage for the remaining search algorithms, varying the FRET variant and dead-end pruning method. Best results in **bold**. The overall best DFHS variant, HDP, is included for ease of reference. Abbreviations as in Table 16.7; “M&S” dead-end pruning via merge-and-shrink with size bound  $N = 50k$  and no size bound  $\infty$ . “on DB”: run on bisimulation-reduced state space. Default tie-breaking strategy.

this holds to a small extent even for the trivial value function initialization, its lead becomes substantial when dead-end pruning is enabled. FRET-V does not nearly benefit as much from the heuristic functions; regardless of the underlying heuristic search algorithm.

Among the different dead-end pruning methods,  $h^{\text{FF}}$  has a clear lead. That lead is small but consistent for the search algorithms other than the FRET- $\pi$  variants (with advantages primarily due to ExplodingBlocks). The lead is significant for the FRET- $\pi$  variants, where  $h^{\text{FF}}$  outclasses the other heuristics on multiple domains.  $h^{\text{FF}}$  is especially effective on Blocksworld, Boxworld, ExplodingBlocks, Random, and Zenotravel. Now, recall however that of those 5 domains, ExplodingBlocks is the only one with dead ends. Thus, the benefit of  $h^{\text{FF}}$  here is actually not attributed entirely to its stronger dead-end detection capabilities. Indeed, this brings into focus a FRET implementation detail that turns out to be particularly important. To facilitate the implementation of our tie-breaking strategies, we order states in our trap representation by increasing  $h$  value (according to the  $h$  used for dead-end detection). Trap-leaving transitions are processed following the order of the source states in the trap, biasing the (default) policy selection towards source states with smaller  $h$  values – thus states, deemed to be closest to the goal. The main advantage of  $h^{\text{FF}}$  lies in offering a particularly strong guidance during policy action selection while at the same time causing less overhead than its competitors. The “blind” configurations (not using any heuristic for dead-end pruning) in contrast order the trap states arbitrarily, so for the dead-end free domains, they reflect exactly the results of when this feature is disabled. On the domains with dead ends, the trap state order had only a marginal impact. On those domains, disabling the ordering feature yields exactly the same coverage results.

Domain	#	VI			FRET-V LRTDP U			FRET- $\pi$ LRTDP U			HDP U			idual								
		-	$h^{\text{FF}}$	$\frac{\mathbb{E}}{\mathbb{D}}$	M&S	N	$\infty$	-	$h^{\text{FF}}$	$\frac{\mathbb{E}}{\mathbb{D}}$	M&S	N	$\infty$	-	$h^{\text{FF}}$	$\frac{\mathbb{E}}{\mathbb{D}}$	M&S	N	$\infty$			
IPPC Benchmarks																						
Blocksw	9	57.8	57.8	57.8	57.8	57.8	57.8	15.6	57.8	57.8	57.8	57.8	15.6	0.55	0.64	0.48	0.48	9.12	9.12	9.12	9.12	9.12
$\hookrightarrow(\text{non-triv.})$	5	0.1k	0.1k	0.1k	0.1k	0.1k	0.1k	27.4	0.1k	0.1k	0.1k	0.1k	27.4	0.89	1.0	0.79	0.79	15.6	15.6	15.6	15.6	15.6
Drive	8	0.19	0.17	0.17	0.17	0.17	0.17	0.18	0.12	0.13	0.12	0.12	0.18	0.12	0.13	0.12	0.12	0.17	0.12	0.13	0.12	0.12
Elevators	5	0.91	0.91	0.91	0.91	0.91	0.91	0.3	0.8	0.69	0.79	0.79	0.3	0.27	0.26	0.23	0.23	0.41	0.41	0.41	0.41	0.41
ExpBloc	7	0.16k	25.8	43.9	0.13k	19.7	19.7	5.93	0.16	0.24	2.55	0.12	5.93	0.16	0.24	2.49	0.12	27.6	0.51	0.73	11.7	0.44
Random	1	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	0.92	0.92	0.92	0.92	3.18	3.18	3.18	3.18	3.18
RecFire	14	0.41	0.31	0.23	0.23	0.23	0.23	0.4	0.3	0.21	0.21	0.21	0.36	0.17	0.05	0.05	0.05	0.41	0.29	0.21	0.21	0.21
$\hookrightarrow(\text{non-triv.})$	4	1.07	0.84	0.61	0.61	0.61	0.61	1.07	0.84	0.59	0.59	0.59	0.93	0.45	0.09	0.09	0.09	1.06	0.81	0.57	0.57	0.57
Schedule	6	39.5	18.8	18.8	18.8	18.8	18.8	0.72	0.13	0.13	0.13	0.13	0.7	0.13	0.13	0.13	0.13	0.06	0.06	0.06	0.06	0.06
$\hookrightarrow(\text{non-triv.})$	1	0.18k	85.0	85.0	85.0	85.0	85.0	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.08	0.08	0.08	0.08	0.08
SeaResc	3	9.64	8.67	8.67	8.67	8.67	8.67	9.63	8.66	8.67	8.66	8.66	9.37	7.79	7.8	7.79	7.79	9.57	7.78	7.77	7.78	7.78
Tirew	12	3.36k	3.36k	3.36k	3.36k	3.36k	3.36k	0.71	90.3	90.5	90.3	90.3	0.71	0.22	0.37	0.22	0.22	0.17	0.23	0.47	0.23	0.23
$\hookrightarrow(\text{non-triv.})$	8	5.0k	5.0k	5.0k	5.0k	5.0k	5.0k	0.92	0.14k	0.14k	0.14k	0.14k	0.92	0.27	0.47	0.27	0.27	0.22	0.21	0.62	0.21	0.21
Zenotra	3	0.31k	0.31k	0.31k	0.31k	0.31k	0.31k	19.0	0.31k	0.31k	0.31k	0.31k	19.0	0.6	28.0	0.2	0.2	4.93	4.93	4.93	4.93	4.93
$\hookrightarrow(\text{non-triv.})$	2	0.46k	0.46k	0.46k	0.46k	0.46k	0.46k	28.6	0.46k	0.46k	0.46k	0.46k	28.6	0.89	42.0	0.31	0.31	7.39	7.39	7.39	7.39	7.39
Canadian RCP Benchmarks																						
Rovers	8	33.2	33.2	33.2	33.2	33.2	33.2	27.2	29.2	29.6	29.6	29.6	3.48	3.43	3.24	3.19	3.19	7.23	6.67	6.69	6.67	6.67
TPP	2	0.31k	0.31k	0.31k	0.31k	0.31k	0.31k	0.31k	0.31k	0.31k	0.31k	0.31k	16.5	9.1	9.54	9.66	9.66	18.4	9.58	9.58	9.58	9.58
$\hookrightarrow(\text{non-triv.})$	1	0.54k	0.54k	0.54k	0.54k	0.54k	0.54k	0.54k	0.54k	0.54k	0.54k	0.54k	3.59	2.31	2.31	2.31	2.31	4.7	2.43	2.43	2.43	2.43

Table 16.9.: Cyclic MaxProb planning. Per-domain average search space size (number of states visited) data in multiples of 1000. Similar setup and presentation as in Table 16.4: “#” gives the size of the instance basis. The default are all instances commonly solved by the shown configurations. “ $\hookrightarrow(\text{non-triv.})$ ” considers of those only instances not solved by VI in  $< 1$  second.

Domain	#	VI				FRET-V LRTDP U				FRET- $\pi$ LRTDP U				HDP U				idual								
		-	$h^{FF}$	$P_D$	M&S	-	$h^{FF}$	$P_D$	M&S	-	$h^{FF}$	$P_D$	M&S	-	$h^{FF}$	$P_D$	M&S	-	$h^{FF}$	$P_D$	M&S					
		N	$\infty$	N	$\infty$	N	$\infty$	N	$\infty$	N	$\infty$	N	$\infty$	N	$\infty$	N	$\infty$	N	$\infty$	N	$\infty$					
IPPC Benchmarks																										
Blocksw	9	0.69	1.47	0.76	2.21	2.66	57.5	1.82	0.84	2.14	2.69	44.8	0.02	0.05	1.33	1.66	34.3	0.02	0.05	1.21	1.67	17.9	17.9	18.5	18.6	19.0
$\hookrightarrow(non-triv.)$	5	1.24	2.62	1.32	3.54	4.34	103.5	3.25	1.46	3.47	4.45	80.6	0.03	0.05	2.0	2.6	61.8	0.03	0.06	1.77	2.62	32.1	32.1	33.0	32.9	33.5
Drive	8	0.01	0.01	0.02	1.8	41.0	0.01	0.01	0.02	1.68	37.0	0.01	0.01	0.02	1.6	37.6	0.01	0.01	0.03	1.6	36.2	0.03	0.02	0.03	1.74	41.4
Elevators	5	0.01	0.01	0.05	1.96	2.61	0.01	0.02	0.06	1.56	2.14	0.01	0.01	0.05	1.6	2.28	0.01	0.01	0.05	1.58	2.24	0.08	0.08	0.12	1.95	2.59
ExpBloc	7	0.39	0.23	1.32	8.41	273.3	0.28	0.01	1.27	6.84	250.8	0.26	0.01	1.27	6.62	255.0	0.11	0.01	1.21	6.58	248.5	55.8	0.02	1.31	11.3	279.3
Random	1	0.34	0.46	0.41	0.41	0.42	65.2	0.59	0.56	0.57	0.56	52.3	0.05	0.14	0.11	0.11	39.6	0.04	0.13	0.1	0.11	0.38	0.46	0.51	0.48	0.46
RecTire	14	1.26	1.85	2.11	1.92	1.88	20.1	1.75	2.03	1.85	1.83	15.3	1.55	2.01	1.84	1.83	15.6	1.52	2.0	1.8	1.83	21.5	1.93	2.14	1.99	1.96
$\hookrightarrow(non-triv.)$	4	4.09	6.06	6.75	6.21	6.1	69.0	5.74	6.46	6.0	5.93	52.6	5.05	6.4	5.98	5.95	53.7	4.94	6.39	5.84	5.93	74.0	6.31	6.8	6.46	6.36
Schedule	6	0.42	0.5	0.34	1.22	0.88	0.05	0.01	0.05	0.94	0.56	0.04	0.01	0.05	0.96	0.53	0.06	0.01	0.05	1.04	0.52	0.01	0.01	0.05	1.08	0.57
$\hookrightarrow(non-triv.)$	1	2.26	2.59	1.69	6.37	4.39	0.01	0.01	0.07	4.92	2.65	0.01	0.01	0.07	5.04	2.47	0.01	0.01	0.07	5.52	2.39	0.01	0.01	0.08	5.73	2.67
SeaResc	3	0.03	0.05	0.19	0.07	0.07	0.17	0.32	0.47	0.34	0.35	0.15	0.11	0.24	0.12	0.12	0.1	0.09	0.23	0.11	0.11	28.4	6.29	5.13	5.3	5.45
Tirew	12	9.79	25.5	10.2	19.2	62.1	0.01	0.82	0.5	2.74	36.3	0.01	0.01	0.03	2.21	36.5	0.01	0.01	0.03	2.19	35.6	0.01	0.01	0.05	2.48	39.9
$\hookrightarrow(non-triv.)$	8	14.6	38.1	15.2	28.5	92.6	0.01	1.23	0.74	3.89	54.1	0.01	0.01	0.03	3.09	54.4	0.01	0.01	0.03	3.06	53.0	0.01	0.01	0.06	3.46	59.5
Zenotra	3	1.99	9.13	2.11	4.03	23.1	31.6	10.3	66.1	5.08	25.1	19.3	0.04	37.0	1.71	20.4	14.9	0.04	37.2	1.84	20.6	7.1	7.35	7.43	8.33	26.7
$\hookrightarrow(non-triv.)$	2	2.98	13.7	3.15	6.04	34.7	47.5	15.5	99.1	7.62	37.7	28.9	0.05	55.5	2.56	30.6	22.3	0.05	55.8	2.75	30.8	10.6	11.0	11.1	12.5	40.1
Canadian RCP Benchmarks																										
Rovers	8	0.18	0.26	0.83	0.3	0.29	0.47	0.77	1.29	0.82	0.83	0.15	0.1	0.73	0.19	0.2	0.16	0.11	0.73	0.2	0.2	14.5	11.0	11.7	11.2	11.6
TTP	2	0.86	1.66	1.27	2.15	5.77	2.7	4.22	4.4	5.41	8.45	0.75	0.34	0.6	1.27	4.64	0.68	0.3	0.59	1.37	4.48	135.2	23.9	22.2	25.2	28.8
$\hookrightarrow(non-triv.)$	1	1.5	2.9	2.09	3.36	10.0	3.91	6.57	6.29	7.98	13.4	0.07	0.03	0.38	1.37	7.5	0.06	0.03	0.39	1.48	7.37	1.78	0.39	0.65	1.87	8.41

Table 16.10.: Cyclic MaxProb planning. Per-domain average runtime (in seconds) over commonly solved instances. Same setup and presentation as in Table 16.9.

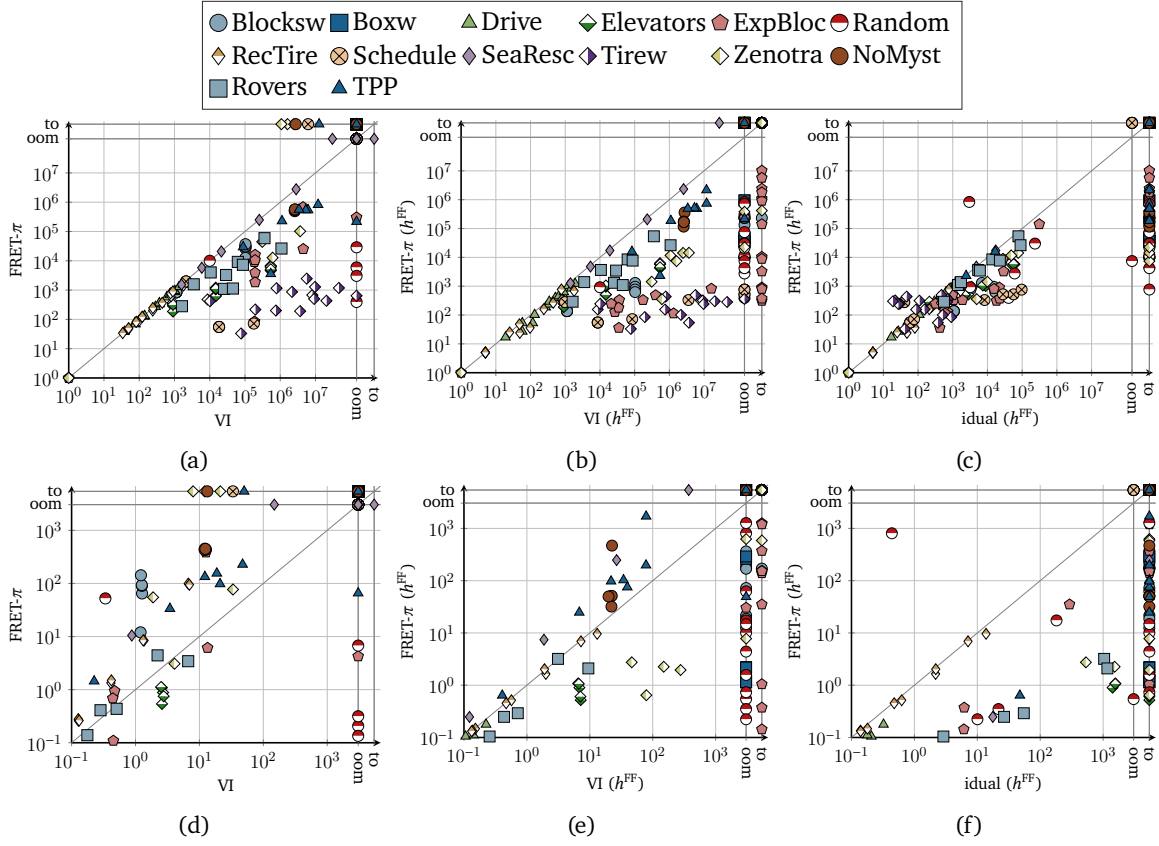


Figure 16.5.: Cyclic MaxProb planning. Per-instance comparison between FRET- $\pi$  with LRTDP $_{\text{U}}$  ( $y$ -axes) and VI respectively IDUAL ( $x$ -axes), with and without pruning using  $h^{\text{FF}}$ . Top: search space size (number of states visited). Bottom: runtime in seconds. Default tie-breaking strategy. “oom” represents out of memory; “to” out of time.

As before, we shed additional light on the coverage results through search space size and runtime data. Figure 16.5 compares the search space sizes for LRTDP $_{\text{U}}$  with FRET- $\pi$  vs. VI and IDUAL. As evident from part (a) and (b) of this figure, the non-trivial value initialization using  $h^{\text{FF}}$  is useful, but gains of up to 3 orders of magnitude over VI are possible even without it.

Table 16.9 provides aggregate search space size and Table 16.10 aggregate runtime data. No data is shown for the configuration using FRET- $V$  with HDP $_{\text{U}}$  as, like coverage, that data is almost identical to that of FRET- $V$  with LRTDP $_{\text{U}}$ : the search space sizes are exactly the same, and runtimes differ only by a few seconds. Table 16.9 confirms the general superiority of FRET- $\pi$  also in terms of search space size. It visits significantly less states than its competitors, and this consistently across the benchmark set. Closest to the performance of FRET- $\pi$  is IDUAL. Yet, as shown in Figure 16.5c, IDUAL still frequently considers orders of magnitude more states. The search space size advantage is typically, but not always, reflected on runtime. A notable exception is NoMystery (not shown in the table due to IDUAL not solving any instance, but visible in Figures 16.5d and 16.5e), where FRET- $\pi$  requires excessively many iterations to remove all traps, resulting in a substantial slow-down, despite the smaller search space.

Taking aside the aforementioned impact on policy action selection, the effect of the heuristics in terms of dead-end pruning is considerably smaller than in the acyclic case. This can be clearest seen for VI. Substantial search space reductions can be observed only in a single domain, ExplodingBlocks. In all other

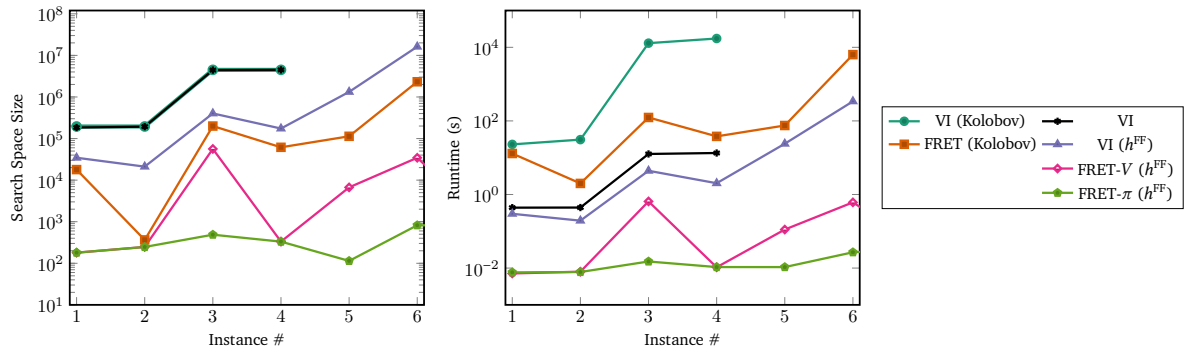


Figure 16.6.: Cyclic MaxProb planning. Comparison between FRET and VI for IPPC’08 ExplodingBlocks, as shown by Kolobov et al. (2011). Left: search space size (number of states visited). Right: runtime in seconds. Both are shown as a function of the IPPC instance index. Different variants included for comparison. The data for the “Kolobov” entries is taken from their paper (the code being not available anymore), hence the runtime comparison is modulo the different computational platforms, and should be treated with care. All shown FRET configurations (including Kolobov’s variant) use LRTDP. Default tie-breaking strategy.

domains, improvements are usually moderate, or even non-existing. The evaluation and construction overhead becomes the dominating factor in the comparison between the different heuristic functions.

ExplodingBlocks also happens to be the single domain Kolobov et al. (2011) experimented with. Figure 16.6 provides a detailed comparison to Kolobov et al.’s data, which is the only state of the art measure provided by previous work. We use here the exact runtime and search space size data reported by Kolobov et al., as their source code is not available anymore.

Kolobov et al. ran VI with no pruning vs. FRET-V using LRTDP with pruning based on SIXTHSENSE (Kolobov et al., 2010a). They observed a coverage of 4 for the former and of 6 for the latter, identical with our results for VI without pruning; our FRET-V configuration using LRTDP with  $h^{FF}$  solves one more instance (note that the results reported in Table 16.8 are aggregated over the IPPC’06 and IPPC’08 versions). To give more details, Figure 16.6 shows the number of states visited, and the total runtime, in terms of plots over IPPC instance index as done by Kolobov et al. (2011).

Consider search space size, the left plot in Figure 16.6. Kolobov’s VI and our VI without pruning visit the exact same number of states. The substantially better performance of VI with  $h^{FF}$  dead-end pruning shows that the omission of Kolobov et al.’s (2011) study, using dead-end pruning in FRET but not in VI, can possibly obfuscate the conclusions regarding the effect of heuristic search vs. the effect of the state pruning itself. Namely, equipped with  $h^{FF}$  pruning, VI becomes almost as effective as Kolobov’s FRET variant using the even stronger dead-end detector SIXTHSENSE. In terms of runtime, the picture on the right-hand side of Figure 16.6, it appears to be even more effective, but SIXTHSENSE’s information sources are also more time-intensive than  $h^{FF}$ . Moreover, keep in mind the different computational environments, so this comparison must be taken with a grain of salt. Moving back to our FRET-V variant, the picture however again changes. Being somewhat more effective than Kolobov’s variant (both in terms of search space size and runtime data), our FRET-V implementation clearly outperforms VI using the same pruning. All in all, given the clarity of FRET- $\pi$ ’s advantage in terms of both metrics, what we can certainly take away is that this variant of FRET substantially improves over the previous state of the art.

As promised, we finally inspect the trap-aware (TA) variants of LRTDP and DFHS. Figure 16.7 shows the data. We compare TA-LRTDP<sub>|U</sub> and TA-HDP<sub>|U</sub> to LRTDP respectively HDP wrapped in FRET- $\pi$ , without and

Domain	#	LRTDP <sub>U</sub>				HDP <sub>U</sub>			
		—		$h^{\text{FF}}$		—		$h^{\text{FF}}$	
		FRET	TA	FRET	TA	FRET	TA	FRET	TA
IPPC Benchmarks									
Blocksw	30	9	9	<b>22</b>	<b>22</b>	9	9	<b>22</b>	<b>22</b>
Boxw	15	0	2	7	9	0	0	7	<b>12</b>
Drive	15	15	15	15	15	15	15	15	15
Elevators	15	15	15	15	15	15	15	15	15
ExpBloc	30	10	10	26	<b>28</b>	10	10	<b>28</b>	<b>28</b>
Random	15	5	5	14	<b>15</b>	5	5	14	<b>15</b>
RecTire	14	14	14	14	14	14	14	14	14
Schedule	30	6	6	<b>10</b>	<b>10</b>	6	6	<b>10</b>	<b>10</b>
SeaResc	15	5	5	5	6	5	5	6	<b>6</b>
Tirew	15	15	15	15	15	15	15	15	15
Zenotra	30	4	5	10	<b>14</b>	4	5	10	12
Σ IPPC	224	98	101	153	163	98	99	156	<b>164</b>
Canadian RCP Benchmarks									
NoMyst	10	4	5	5	5	4	5	5	5
Rovers	10	10	10	10	10	10	10	10	10
TPP	10	8	<b>10</b>	9	<b>10</b>	8	<b>10</b>	9	<b>10</b>
Σ RCP	30	22	<b>25</b>	24	<b>25</b>	22	<b>25</b>	24	<b>25</b>

(a)

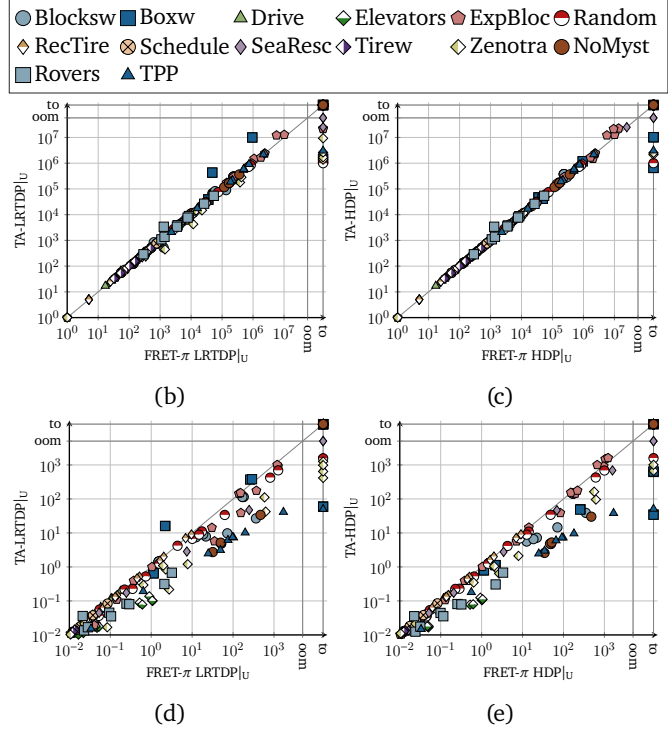


Figure 16.7.: Cyclic MaxProb planning. Comparison between FRET- $\pi$  and the trap-aware (TA) variants of LRTDP and HDP. (a) provides MaxProb coverage data. Best results in **bold**. “FRET” heuristic search wrapped in FRET- $\pi$ , “TA” trap-aware variants without FRET outer loop. Right: per-instance comparison between the FRET- $\pi$  ( $x$ -axes) and trap-aware ( $y$ -axes) variants, using  $h^{\text{FF}}$  pruning. (b) and (c): search space size (number of states visited). (d) and (e): runtime (in seconds). Default tie-breaking strategy. “oom” represents out of memory; “to” out of time.

with ( $h^{\text{FF}}$ ) pruning. In terms of coverage, Figure 16.7a, the TA variants further improve the already strong FRET- $\pi$  baselines, doing so consistently, with significant improvements in Boxworld, ExplodingBlocks (for LRTDP), and Zenotravel. The plots on the right-hand side Figure 16.7 elucidate the reasons for the latter via search space size and runtime comparisons. Consider search space size, Figures 16.7b and 16.7c. The FRET- $\pi$  and TA variants are almost indistinguishable, which makes sense given that the latter differs from the former basically only in interleaving trap analysis with heuristic search instead of keeping the two processes separate. That this integration is however beneficial can be clearly seen in Figures 16.7d and 16.7e. Avoiding running heuristic search from scratch after every trap analysis step, and with that the redundant work associated with each such call, has a tremendous effect on runtime. Runtime reductions of 1 order of magnitude are common, and this goes up to 2 orders of magnitude in some cases. The runtime differences correlate directly with the number of FRET- $\pi$  iterations.

#### AtLeastProb and ApproxProb Objective Parameter Analysis, Tie-Breaking Strategies, and Anytime Behavior

For the weaker objectives AtLeastProb and ApproxProb, as before we examine coverage as a function of  $\theta$  respectively  $\delta$ . Figures 16.8a and 16.8b show the data. Figures 16.8c and 16.8d compare the different tie-breaking strategies in AtLeastProb for the most competitive algorithms from parts (a) and (b). We do not show results for HDP, which performs almost identically to LRTDP.

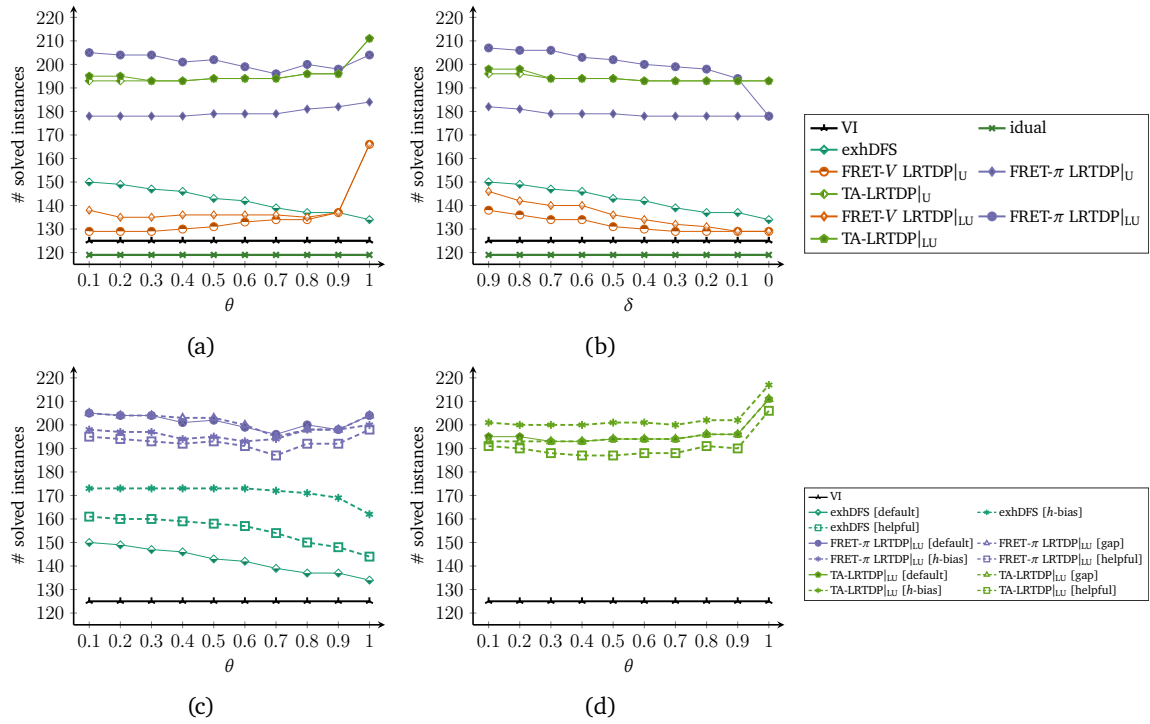


Figure 16.8.: Cyclic planning. Total coverage as a function of the objective parameter value for different goal-probability objectives and algorithm configurations. Comparison of different search algorithms, using the default tie-breaking strategy, on (a) AtLeastProb and (b) ApproxProb. (c) and (d) compare different tie-breaking strategies on AtLeastProb. Abbreviations as in Figure 16.3. All configurations using  $h^{\text{FF}}$  pruning.

The overall behavior in Figure 16.8 is similar to that for the acyclic case. ExhDFS has a surprisingly strong performance. While not entirely reaching the performance of FRET- $\pi$ , it is able to beat FRET-V for the  $\theta$  values up to 0.9, due to aggressive early termination; and ExhDFS maintains a lead over the VI baseline across the board. In the range from 0.9 to 1, FRET-V is much stronger due to negative early termination on the goal-probability upper bound. FRET- $\pi$  and the TA variant perform similarly, with a slight edge for FRET- $\pi$ , which tends to be more effective on raising the lower bound. When maintaining both an upper and a lower bound, the FRET configurations again exhibit an easy-hard-easy pattern due to the advantages of early termination. That this pattern is somewhat less pronounced for the FRET- $\pi$  variants is an artifact of the benchmark set, combined performance loss at some point in the scaling. In each domain, there is an instance number  $x$  so that, below  $x$ , FRET- $\pi$  solves MaxProb, while above  $x$  neither  $V^L(s_I)$  nor  $V^U(s_I)$  can be improved at all, remaining 0 respectively 1 up to the time/memory limit.

The effect of the tie-breaking strategies is almost identical to the acyclic case. ExhDFS benefits the most from non-default tie-breaking. Biasing the DFS exploration towards states with smaller  $h$  values has proved particularly effective in quickly increasing the lower bound, fostering early termination. The helpful-actions bias is still beneficial, yet to a smaller extent, providing less fine granular guidance. As opposed to the acyclic case, we did not experiment with the breadth-first exploration bias, which is incompatible with the ExhDFS search algorithm. The  $h$ -bias is also beneficial for TA-LRTDP. The other tie-breaking strategies have only a marginal effect on the development of the bounds in LRTDP in either variant. Due to the additional overhead, they negatively affect the performance, if at all.

Figure 16.9 exemplifies anytime behavior of ExhDFS and LRTDP $_{|LU}$  with FRET- $\pi$  for ExplodingBlocks. The

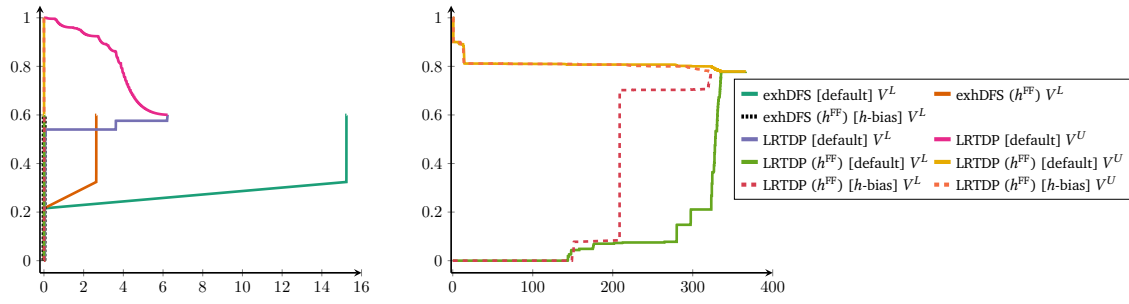


Figure 16.9.: Cyclic planning. Anytime behavior of ExhDFS and  $\text{LRTDP}|_{\text{LU}}$  with FRET- $\pi$ , with and without pruning, and using the default and  $h$ -bias tie-breaking strategies. Left: IPPC'08 ExplodingBlocks instance 4. Right: instance 15.

right plot considers the largest instance feasible when using pruning. The left plot considers the second-largest instance feasible without pruning: on the largest one feasible without pruning, the maximum goal probability is 1 so the anytime curve for  $V^U$  is not interesting. We included plots for both default and  $h$ -bias tie-breaking to show the impact of that strategy on the development of  $V^L$  bound (we did not vary the tie-breaking strategy in Figure 16.4 for acyclic planning, where the difference between the strategies was negligible and thus not interesting). In the instance on the right-hand side of Figure 16.9, ExhDFS timed out before raising its  $V^L$  bound once.

## 16.4. Occupation-Measure and Operator-Counting Heuristics for Goal Probability

We finally compare the goal-probability heuristics from Chapter 14 to dead-end pruning. We primarily focus on MaxProb analysis. We also experimented with the AtLeastProb and ApproxProb objectives, for which the relative behavior of the different configurations was the same. We next describe the experiment setup, and specify what heuristic variants we run exactly. We then discuss the results.

### 16.4.1. Experiment Setup

We consider cyclic as well as acyclic planning as before. Note, however, that none of our goal-probability heuristics provide native support of an explicit budget. We diverge in the following from our previous experiments on budget-limited planning in that we no longer consider the budget to be an explicit algorithm parameter, and instead enforce it at the level of the PPDDL model files, following an encoding akin to the original resource-constrained planning benchmarks (Nakhost et al., 2012).

We consider two goal-probability occupation-measure heuristics:  $H_{\text{atomic}}^{\text{gpm}}$  over the atomic projections, the variant corresponding to Trevizan et al.'s (2017b) expected-cost heuristic  $h^{\text{pom}}$ ; and, to see how the heuristic behaves for larger projections  $H_{\text{pairs}}^{\text{gpm}}$  over the projections onto all pairs of variables. Moreover, we consider two instances of the goal-probability operator-counting framework:  $H_{\text{pseq}}^{\text{gpo}}$  using only the probabilistic state equation, and  $H_{\text{pseq+lm}}^{\text{gpo}}$  augmenting the state equation by landmark constraints. We do not consider landmark constraints in isolation, which does not make sense for goal-probability objectives; these would simply result in the trivial goal-probability bound 1 for all states.

The landmarks for the landmark constraints are generated anew for every state via LMcut (Helmert and

Domshlak, 2009).<sup>4</sup> When LMcut returns  $\infty$ , i.e., identifies a dead end, we prune the state. In order to estimate the gains from the landmark constraints over just the additional dead-end pruning capability, we additionally tested the goal-probability operator-counting variant, using the probabilistic state equation without landmark constraints, in combination with dead-end pruning through LMcut. As the landmarks are not needed for the latter configuration, we instead use, as in our previous experiments,  $h^{\text{FF}}$  (Hoffmann and Nebel, 2001) which identifies the exact same dead ends as LMcut while being often cheaper to compute.

We compare all these heuristics to the best-performing pruning methods from Section 16.3: dead-end pruning via  $h^{\text{FF}}$  (Hoffmann and Nebel, 2001) and PDB (Haslum et al., 2007). We include the deterministic state-equation heuristic  $h^{\text{seq}}$  (Bonet, 2013) as an additional dead-end pruning baseline.

We run all heuristics and pruning methods with LRTDP, LILAO\*, and IDUAL. On the cyclic benchmarks, we use the TA variants of LRTDP and LILAO\*.<sup>5</sup> We also include results for FRET-V, which may benefit especially from the (potentially) more accurate value-function initialization. Moreover, we consider a variant of Trevizan et al.’s (2017b)  $i^2\text{dual}$ , an enhanced combination of IDUAL and the projection occupation-measure heuristic over atomic projections, which embeds the heuristic computation directly into IDUAL’s LPs. Trevizan et al. showed in their experiments that this embedding creates a synergistic interplay between heuristic and search, typically resulting in a much stronger performance than using the heuristic in IDUAL via separate heuristic LPs.

To briefly summarize this integration, recall that IDUAL’s LP objective function encodes the “steady-state probability distribution” over frontier and goal states that is induced by the policy underlying IDUAL’s occupation-measure LP variables. The goal states are weighted by their exact  $V^*$ -value, 1, while the frontier states are weighted according to the provided goal-probability heuristic  $H$ . In other words, IDUAL’s objective function simply represents the expected value of the policy that its LP variables represent, as per the current frontier states and the provided  $H$  bounds. Now, observe that given any set of probability-weighted states, the expected  $H_{\text{atomic}}^{\text{gpm}}$  value can be computed via just a single LP (omitting  $\text{atomic}$  in the following). To do so, one only needs to aggregate the given probability weights, computing for each variable-value pair (fact) the sum of the probability weights of the states that contain that fact. The fact probability weights must then only be plugged into the  $H^{\text{gpm}}$  LP by changing accordingly the bounds of the goal (14.1d) and flow constraints (14.1c). From this, the integration of  $H^{\text{gpm}}$  into IDUAL is straightforward. Both their LPs are merged. The set of states for which the expected heuristic value is to be computed are IDUAL’s current frontier states. The probability weights associated with any one of them is given by the sum over IDUAL’s occupation-measure LP variables representing a transition going into that frontier state; and these accordingly yield  $H^{\text{gpm}}$ ’s constraint bounds. Finally, the frontier-state part of IDUAL’s objective function is replaced by  $H^{\text{gpm}}$ ’s goal-probability LP variable  $v_G$ , i.e., the variable that stores the expected goal-probability estimate. For simplicity, we stick to the name  $i^2\text{dual}$  to denote this integration. However, keep in mind that our version differs from the one by Trevizan et al. (2017b), as we are considering goal-probability maximization rather than expected-cost minimization. Lastly, notice that this integration is actually not bound to the atomic occupation-measure projection heuristic, but in principle works for any set of projections. However, we do not further explore this idea here.

<sup>4</sup>Given that the landmarks for different states are not necessarily related, the resulting goal-probability operator-counting heuristic may not be monotone. We circumvent this issue by replacing the Bellman update operator by  $(\hat{B}V)(s) := \min\{V(s), (BV)(s)\}$ . Notice that when starting from any upper bound, the sequence of value functions produced by the modified  $\hat{B}$  operator is again guaranteed to be monotonically decreasing (which is what we need).

<sup>5</sup>To avoid comparability issues stemming from the policy tie-breaking choices being affected by the trap state order (cf. Section 16.3.2), we order the states in our trap representation by increasing  $h^{\text{FF}}$  value in all configurations.

Domain	#	FRET-V LRTDP									LRTDP									idual										
		prune			gpom			gpoc			prune			gpom			gpoc			prune			gpom			gpoc				
		–	$h^{\text{FF}}$	PDB	seq	atomic	pairs	pseq	pseq+lm	pseq/ $h^{\text{FF}}$	–	$h^{\text{FF}}$	PDB	seq	atomic	pairs	pseq	pseq+lm	pseq/ $h^{\text{FF}}$	–	$h^{\text{FF}}$	PDB	seq	atomic	pairs	pseq	pseq+lm	pseq/ $h^{\text{FF}}$	i <sup>2</sup> dual	
Cyclic IPPC Benchmarks																														
Blocksw	30	9	9	9	9	9	4	9	9	9	9	22	22	22	19	9	23	19	22	9	9	9	9	9	4	9	9	9	25	
Boxw	30	0	0	0	0	0	0	0	0	0	2	9	10	7	7	0	7	7	7	0	0	0	0	0	0	0	0	0	0	
Drive	15	15	15	15	15	15	8	15	15	15	15	15	15	15	15	8	15	15	15	15	15	15	15	15	15	15	15	15	15	
Elevators	15	15	15	15	15	15	10	15	15	15	15	15	15	15	15	15	15	15	15	15	13	13	13	13	13	10	13	13	13	
ExpBloc	30	10	14	12	10	10	9	10	12	14	10	28	16	10	10	10	10	24	25	8	18	11	9	8	10	9	18	18	9	
Random	15	5	2	4	2	2	1	2	2	2	5	15	11	14	12	2	14	13	14	5	5	5	5	4	1	5	5	5	10	
RecTire	14	14	14	14	14	13	10	14	14	14	14	14	14	14	13	10	14	14	14	14	14	14	14	14	13	10	14	14	13	
Schedule	30	6	10	10	7	10	7	10	10	10	6	10	10	6	10	7	10	10	10	7	10	10	6	10	6	10	10	10	7	
SeaResc	15	5	5	5	5	5	3	5	4	5	5	6	6	5	5	3	5	5	6	3	3	3	3	3	3	3	3	3	3	
Tirew	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	14	15	15	15	15	
Zenotra	30	4	8	6	8	6	1	8	5	8	5	14	15	12	10	3	12	9	12	5	5	5	5	5	1	5	4	5	8	
Σ IPPC	239	98	107	105	100	100	68	103	101	107	101	163	149	135	131	82	140	146	155	94	107	100	94	95	67	98	106	107	120	
Canadian RCP Benchmarks without Budget Limit																														
NoMyst	10	5	5	5	5	5	0	5	5	5	5	5	5	5	0	5	5	5	5	0	0	0	0	0	0	0	0	0	0	
Rovers	10	10	10	10	10	10	8	10	10	10	10	10	10	10	10	10	10	10	10	8	10	10	8	8	8	8	10	10	8	
TPP	10	6	6	6	6	6	1	6	6	6	10	10	10	10	10	2	10	10	10	2	2	2	2	2	2	2	2	2	2	
Σ RCP	30	21	21	21	21	21	9	21	21	21	25	25	25	25	25	12	25	25	25	10	12	12	10	10	10	10	12	12	10	
Acyclic IPPC Benchmarks																														
TriTire	10										10	10	10	10	10	6	10	10	10	10	10	10	10	10	10	3	10	10	10	10
IPPC Benchmarks with Compiled Budget Limit																														
Blocksw-b	180										54	54	61	63	51	0	54	54	54	54	54	54	54	64	54	0	54	54	54	23
Drive-b	90										90	90	90	82	56	20	82	76	85	89	90	90	81	56	20	85	76	86	62	
Elevators-b	90										79	75	78	62	46	5	61	58	61	42	42	43	45	42	3	42	47	42	44	
ExpBloc-b	150										62	83	103	71	57	6	64	85	87	44	81	84	66	53	6	58	85	85	47	
Random-b	72										37	36	44	50	48	11	48	48	48	38	38	43	48	41	11	42	42	43	45	
RecTire-b	36										18	18	18	12	10	1	16	13	16	13	18	18	12	9	1	16	13	16	12	
Schedule-b	138										62	60	62	60	56	36	60	60	60	58	59	59	60	56	36	59	59	58	44	
SeaResc-b	90										71	59	84	49	38	6	52	49	56	30	34	60	34	33	6	34	35	35	17	
Tirew-b	90										90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	
TriTire-b	60										49	43	57	46	29	15	37	35	38	29	32	48	46	29	15	34	34	34	25	
Zenotra-b	78										42	21	42	12	1	0	7	6	9	3	7	25	9	0	0	3	5	6	0	
Σ IPPC-b	1092										654	629	729	597	482	190	571	574	604	490	545	614	555	463	188	517	540	549	409	
Canadian RCP Benchmarks with Compiled Budget Limit																														
NoMyst-b	60										20	10	53	0	0	0	0	3	6	0	2	25	0	0	0	0	2	2	0	
Rovers-b	60										51	49	58	37	18	0	38	41	45	9	20	35	14	10	0	11	21	20	1	
TPP-b	60										26	18	47	3	1	0	2	3	4	1	2	15	2	1	0	1	2	2	0	
Σ RCP-b	180										97	77	158	40	19	0	40	47	55	10	24	75	16	11	0	12	25	24	1	
Pentesting Benchmarks																														
Pentest-b	90										61	58	68	55	46	1	57	58	58	37	47	48	48	44	1	50	50	50	28	
Pentest	15										9	9	9	9	9	8	9	9	9	7	7	7	7	8	8	8	8	8	5	
Σ Pentest	105										70	67	77	64	55	9	66	67	67	44	54	55	55	52	9	58	58	58	33	

Table 16.11.: MaxProb coverage on cyclic (top) and acyclic (bottom) benchmarks. Best results in **bold**. FRET-V results not shown for the acyclic part, where FRET is generally not needed. “LRTDP” on the cyclic part shows the results of TA-LRTDP. Comparison of different goal-probability heuristics: “–” trivial value initialization (1 everywhere) and no pruning; “prune” dead-end pruning by either  $h^{\text{FF}}$ , PDB, as before, or “seq” the deterministic state-equation heuristic; “gpom” goal-probability occupation-measure heuristic over the atomic projections “atomic”, and over the projections onto all pairs of variables “pairs”; “gpoc” goal-probability operator-counting heuristic, all configurations use the probabilistic state-equation as basis, “pseq” only that, “pseq+lm” with landmarks constraints using landmarks from LMcut, “pseq/ $h^{\text{FF}}$ ” combines pseq with dead-end pruning by  $h^{\text{FF}}$ . “i<sup>2</sup>dual” embedding of  $H_{\text{atomic}}^{\text{gpoc}}$  directly into IDUAL.

### 16.4.2. Discussion

Table 16.11 provides MaxProb coverage data for the cyclic (top) and acyclic (bottom) benchmarks. For space reasons, we do not show results for LILAO\*, whose performance is overall close to that of LRTDP (cf. Section 16.3). As in our previous experiments, FRET-V LRTDP is completely outclassed by the TA-LRTDP variant. We hence focus on the latter in the following.

Before we dive into the heuristics' evaluation, we need to insert a general remark regarding the budget compilation. Note that LRTDP without pruning nor heuristic solves considerably less instances than the explicit-budget variant in Table 16.3. While the benchmarks are semantically equivalent, the budget compilation inflates the syntactic planning task size, which (a) induces a considerable overhead in *grounding*, i.e., in processing the budget-extended PPDDL model files, and furthermore (b) results in a slow down of frequently used operations like the transition computation due to a significantly increased number of actions. Moreover, notice that the difference between the dead-end pruning configurations in Table 16.11 and the budget-pruning ones in Table 16.3 is even larger. This indicates that the explicit knowledge of the budget is exploited more effectively for pruning by the latter configurations compared to the completely automatically computed dead-end information when the budget is not explicitly provided.

Consider now the advanced goal-probability heuristics in Table 16.11. In terms of raw coverage, they are not able to compete with dead-end pruning. The overhead associated with the heuristic computations could typically not be compensated by more informative bounds. The heuristic computation becomes in particular a bottleneck in the budget-limited part. In 35% of the cyclic benchmark instances and about 70% of the acyclic instances, the computation of  $H_{\text{pairs}}^{\text{gpom}}$  for the initial state did not even finish within the 30 minutes time limit. While in significantly less cases, the computation of  $H_{\text{atomic}}^{\text{gpom}}$  and the two probabilistic operator-counting heuristics for the initial state did still exceed the time limit in 5% and 2% of the acyclic instances respectively. Coverage differences between the different heuristics directly reflect the magnitude of their overhead.

The most surprising result from Table 16.11 is  $i^2\text{dual}$ . While on the acyclic part  $i^2\text{dual}$  also suffers from exceedingly expensive LP solver calls, on the cyclic IPPC benchmarks, the embedding of  $H_{\text{atomic}}^{\text{gpom}}$  into IDUAL lifts the (hopeless) performance of IDUAL to the level of the overall best configuration, TA-LRTDP with  $h^{\text{FF}}$  pruning, on multiple domains. Note in particular that  $i^2\text{dual}$  achieves substantially better coverage results than the IDUAL configuration with separate  $H_{\text{atomic}}^{\text{gpom}}$ . We emphasize that the synergy between IDUAL and  $H_{\text{atomic}}^{\text{gpom}}$  in  $i^2\text{dual}$  goes beyond a more effective use of heuristic information. The most drastic coverage improvements are in Blocksworld, Elevators, Random, and Zenotravel. As previously noted, Blocksworld, Random, and Zenotravel are dead-end free, so the goal-probability bounds make no difference here (are 1 throughout). We speculate that the integration of  $H_{\text{atomic}}^{\text{gpom}}$  into IDUAL's LPs guides the LP solver by introducing a distinction between states for which an optimal policy is already available in IDUAL's search space, and states for which this is not the case. Presumably, the LP solver prefers visiting the former states only, as a non-zero probability of reaching IDUAL's frontier states entails finding a non-0 solution to the embedded  $H_{\text{atomic}}^{\text{gpom}}$  LP, and thus requires additional computations.

Even though at the end of the day, none of the tested goal-probability heuristics payed off, in some domains they still yielded much better search guidance compared to dead-end pruning. Table 16.12 presents per-domain search space size statistics averaged over the per search algorithm commonly solved instances. We omitted the dead-end free domains. As when using  $H_{\text{pairs}}^{\text{gpom}}$ , it is typically only possible to finish the smallest instances, we additionally provide in Table 16.13 data aggregating the results over the instances commonly solved by the shown configuration, omitting  $H_{\text{pairs}}^{\text{gpom}}$  and its solved instances. (We also omitted  $H_{\text{atomic}}^{\text{gpom}}$  from this comparison, see next.)

Domain	LRTDP										idual										
	prune					gpom					gpoc										
	#	-	$h^{\text{FF}}$	PDB	seq	atomic	pairs	pseq	pseq+lm	pseq/ $h^{\text{FF}}$	#	-	$h^{\text{FF}}$	PDB	seq	atomic	pairs	pseq	pseq+lm	pseq/ $h^{\text{FF}}$	$t^2_{\text{dual}}$
Cyclic IPPC Benchmarks																					
Drive	8	0.17k	0.13k	0.13k	0.17k	0.14k	<b>0.11k</b>	0.14k	0.12k	0.13k	8	0.17k	0.12k	0.13k	0.17k	0.13k	<b>0.11k</b>	0.13k	0.12k	0.12k	0.16k
ExpBloc	10	0.16m	<b>0.21k</b>	7.54k	52.73k	52.73k	0.39k	52.73k	<b>0.21k</b>	<b>0.21k</b>	8	38.96k	<b>0.68k</b>	0.89k	23.16k	64.28k	0.75k	42.5k	<b>0.68k</b>	<b>0.68k</b>	4.8k
RecFire	10	0.13k	51.4	<b>28.9</b>	48	48	48	48	40.9	51.4	10	0.15k	87.8	<b>60.5</b>	87.8	87.8	87.8	87.8	69.8	87.8	0.15k
Schedule	6	0.7k	<b>0.13k</b>	<b>0.13k</b>	0.7k	<b>0.13k</b>	<b>0.13k</b>	<b>0.13k</b>	<b>0.13k</b>	<b>0.13k</b>	6	56.67	<b>55.67</b>	<b>55.67</b>	56.67	<b>55.67</b>	<b>55.67</b>	<b>55.67</b>	<b>55.67</b>	<b>55.67</b>	0.25k
SeaResc	3	9.37k	<b>7.79k</b>	7.8k	9.37k	9.37k	7.8k	9.37k	7.8k	<b>7.79k</b>	3	9.57k	7.78k	<b>7.77k</b>	9.56k	9.57k	7.78k	9.57k	7.79k	7.78k	9.6k
Tirew	15	0.72k	<b>0.24k</b>	0.38k	0.72k	0.71k	0.69k	0.71k	<b>0.24k</b>	<b>0.24k</b>	14	0.23k	0.25k	0.47k	0.23k	0.23k	0.23k	0.23k	0.25k	0.25k	<b>0.14k</b>
Canadian RCP Benchmarks without Budget Limit																					
Rovers	10	11.46k	11.06k	10.79k	11.39k	11.39k	11.39k	11.39k	<b>8.63k</b>	11.06k	8	7.23k	6.67k	6.69k	7.23k	7.23k	7.23k	7.23k	4.65k	6.67k	<b>2.79k</b>
TPP	2	21.43k	9.83k	10.08k	18.08k	18.08k	18.08k	18.08k	<b>4.86k</b>	9.83k	2	18.41k	9.58k	9.58k	18.41k	18.41k	18.79k	18.41k	<b>5.36k</b>	9.58k	14.51k
Acyclic IPPC Benchmarks																					
TriFire	6	0.17k	0.17k	0.17k	0.17k	0.17k	0.17k	0.17k	0.17k	0.17k	3	3.26k	2.91k	2.91k	3.26k	3.26k	3.26k	3.26k	2.91k	2.91k	<b>0.64k</b>
IPPC Benchmarks with Compiled Budget Limit																					
Drive-b	20	0.23k	0.17k	<b>86.1</b>	0.13k	0.17k	92.8	0.17k	0.16k	0.16k	20	0.23k	0.17k	<b>86.1</b>	0.13k	0.17k	92	0.17k	0.16k	0.16k	0.22k
Elevators-b	5	0.36k	0.36k	65.6	0.18k	0.18k	<b>52.8</b>	0.18k	0.1k	0.18k	3	0.47k	0.44k	94.67	0.23k	0.26k	<b>72.33</b>	0.26k	0.16k	0.26k	0.19k
ExpBloc-b	6	0.57k	0.29k	64.33	0.11k	0.11k	<b>63.17</b>	0.11k	69	97.33	6	0.97k	0.34k	<b>0.11k</b>	0.18k	0.18k	<b>0.11k</b>	0.18k	0.12k	0.15k	0.12k
Random-b	11	0.42k	0.42k	0.42k	0.42k	<b>0.11k</b>	<b>0.11k</b>	<b>0.11k</b>	<b>0.11k</b>	<b>0.11k</b>	11	0.42k	0.41k	0.41k	0.41k	<b>0.11k</b>	<b>0.11k</b>	<b>0.11k</b>	<b>0.11k</b>	<b>0.11k</b>	<b>0.11k</b>
RecFire-b	1	0.81k	85	<b>36</b>	85	85	<b>36</b>	85	85	85	1	0.81k	85	<b>36</b>	85	85	<b>36</b>	85	85	85	0.33k
Schedule-b	36	19.36	19.36	13.97	14.69	18.92	<b>13.53</b>	18.92	18.75	18.92	36	24.25	19.92	13.86	15.42	18.97	<b>13.28</b>	18.97	18.5	18.97	24.08
SeaResc-b	6	2.01k	1.8k	<b>0.21k</b>	1.73k	1.9k	0.6k	1.9k	1.5k	1.73k	6	1.98k	1.78k	<b>0.2k</b>	1.68k	1.85k	0.53k	1.85k	1.42k	1.73k	1.89k
Tirew-b	90	0.14k	0.13k	54.33	86.31	81.91	<b>43.07</b>	81.91	74.31	75.59	90	0.13k	0.11k	43.83	76.11	71.8	<b>35.7</b>	71.56	64.41	65.03	0.13k
TriFire-b	15	0.3k	0.29k	45.47	60.67	0.21k	<b>32.93</b>	0.21k	0.21k	0.21k	15	0.33k	0.3k	44.07	61.07	0.2k	<b>31.67</b>	0.2k	0.19k	0.19k	0.24k
Pentesting Benchmarks																					
Pentest-b	1	0.24k	0.15k	<b>0.1k</b>	0.13k	0.13k	0.12k	0.13k	0.12k	0.13k	1	0.24k	0.15k	<b>0.1k</b>	0.13k	0.13k	0.12k	0.13k	0.12k	0.13k	0.18k
Pentest	8	38.9k	34.75k	34.75k	34.85k	24.84k	<b>24.13k</b>	24.84k	24.33k	24.28k	5	3.25k	2.91k	2.91k	2.91k	2.12k	<b>2.07k</b>	2.12k	2.11k	2.08k	3.25k

Table 16.12.: Per-domain average MaxProb search space size (number of visited states). Dead-end free domains are omitted. We aggregate the results for both search algorithms separately, considering for each search algorithm only instances commonly solved by all heuristic/pruning instantiations. “#” shows the size of the instance basis. Results between LRTDP and idual cannot be compared directly due the aggregation over different instance sets. Best results among both search algorithms in **bold**. Abbreviations as in Table 16.11. “k” multiples of  $10^3$ , “m” of  $10^6$ .

Domain	#	-	LRTDP							#	-	idual						
			prune				gpoc					prune				gpoc		
			$h^{\text{FF}}$	PDB	seq	pseq	pseq+lm	pseq/ $h^{\text{FF}}$	$h^{\text{FF}}$			PDB	seq	pseq	pseq+lm	pseq/ $h^{\text{FF}}$	i <sup>2</sup> dual	
Cyclic IPPC Benchmarks																		
Drive	7	1.08k	0.73k	0.78k	1.1k	0.8k	<b>0.71k</b>	0.73k	7	1.07k	0.72k	0.77k	1.07k	0.8k	<b>0.7k</b>	0.72k	1.01k	
RecTire	4	0.92k	0.42k	<b>92.25</b>	0.68k	0.68k	0.65k	0.42k	3	0.82k	0.52k	<b>0.24k</b>	0.52k	0.52k	0.49k	0.52k	0.81k	
SeaResc	2	1.51m	<b>1.27m</b>	<b>1.27m</b>	1.51m	1.51m	<b>1.27m</b>	<b>1.27m</b>	1	2.92k	0.72k	6.57k	2.92k	2.92k	0.72k	0.72k	<b>0.11k</b>	
Tirew																		
Canadian RCP Benchmarks without Budget Limit																		
NoMyst	5	0.57m	0.2m	0.19m	0.21m	0.21m	<b>0.16m</b>	0.2m										
TPP	8	1.73m	1.03m	1.05m	1.63m	1.63m	<b>0.42m</b>	1.03m										
Acyclic IPPC Benchmarks																		
TriTire	4	0.45k	<b>0.43k</b>	<b>0.43k</b>	0.45k	0.45k	<b>0.43k</b>	<b>0.43k</b>	7	0.22m	0.31m	0.31m	0.22m	0.22m	0.31m	0.31m	<b>6.37k</b>	
IPPC Benchmarks with Compiled Budget Limit																		
Blocksw-b	54	37.33k	36.73k	<b>3.47k</b>	5.28k	16.62k	16.25k	16.64k	23	11.15k	10.99k	<b>0.68k</b>	1.49k	8.28k	8.19k	8.26k	9.36k	
Drive-b	56	10.0k	8.21k	7.46k	<b>4.54k</b>	8.31k	8.05k	8.07k	42	5.14k	4.13k	3.47k	<b>2.08k</b>	4.12k	3.97k	3.98k	4.93k	
Elevators-b	53	55.42k	55.31k	<b>41.6k</b>	44.4k	53.52k	45.97k	53.4k	38	9.43k	7.7k	3.78k	5.76k	6.39k	3.71k	6.48k	<b>1.21k</b>	
ExpBloc-b	54	1.92m	16.16k	1.28k	43.36k	85.56k	<b>0.92k</b>	1.08k	35	34.52k	1.28k	<b>0.79k</b>	4.51k	5.26k	0.86k	0.89k	7.96k	
Random-b	25	28.59k	28.63k	2.93k	2.81k	2.35k	<b>2.23k</b>	2.35k	22	21.76k	51.65k	4.75k	6.5k	6.99k	4.06k	6.99k	<b>1.88k</b>	
RecTire-b	11	2.94k	0.27k	<b>0.14k</b>	0.27k	0.27k	0.27k	0.27k	11	2.78k	0.31k	<b>0.16k</b>	0.32k	0.32k	0.3k	0.31k	0.98k	
Schedule-b	24	54.33k	40.51k	12.74k	<b>8.44k</b>	37.53k	33.71k	37.54k	8	4.3k	2.85k	<b>1.05k</b>	1.26k	2.09k	2.06k	2.14k	6.48k	
SeaResc-b	43	0.27m	0.25m	<b>33.38k</b>	0.27m	0.27m	0.17m	0.25m	11	9.29k	8.15k	<b>2.25k</b>	8.7k	8.95k	6.29k	7.93k	9.34k	
TriTire-b	20	0.1m	97.31k	<b>5.0k</b>	6.43k	70.55k	68.46k	68.53k	10	10.98k	10.29k	<b>0.67k</b>	0.92k	6.82k	6.61k	6.61k	8.53k	
Zenotra-b	6	0.13m	67.44k	<b>14.27k</b>	0.1m	0.12m	67.32k	67.31k										
Canadian RCP Benchmarks with Compiled Budget Limit																		
Rovers-b	37	0.33m	0.19m	<b>59.87k</b>	0.3m	0.34m	0.19m	0.19m	1	11.88k	4.67k	<b>0.44k</b>	10.64k	11.51k	4.56k	4.59k	12.35k	
TPP-b	2	0.12m	78.31k	<b>4.25k</b>	47.15k	87.1k	71.67k	75.8k										
Pentesting Benchmarks																		
Pentest-b	54	0.54m	0.21m	<b>0.13m</b>	<b>0.13m</b>	0.16m	0.16m	0.16m	27	15.36k	3.89k	<b>2.55k</b>	2.6k	3.02k	2.96k	2.99k	4.82k	
Pentest	1	3.23m	2.87m	2.87m	2.87m	1.84m	<b>1.83m</b>	<b>1.83m</b>										

Table 16.13.: Per-domain average MaxProb search space size. We aggregate the results for both search algorithms separately, considering for each search algorithm only instances commonly solved by the shown heuristic/pruning instantiations *but not solved by*  $H_{\text{pairs}}^{\text{gpom}}$ . Otherwise same setup as in Table 16.12 and abbreviations as in Table 16.11. Best results among the individual search algorithms in **bold**. Results between the different search algorithms are not directly comparable due to the aggregation over different instance sets.

In general, the search space size statistics for  $H_{\text{atomic}}^{\text{gpom}}$  and  $H_{\text{pseq}}^{\text{gpoc}}$  are identical for LRTDP, and almost identical for IDUAL. In fact, in most cases,  $H_{\text{atomic}}^{\text{gpom}}$  and  $H_{\text{pseq}}^{\text{gpoc}}$  resulted in the exact same goal-probability estimates. If not, the difference was in the order of  $10^{-16}$  to  $10^{-15}$ , i.e., the most-likely cause being floating-point precision errors. The little differences however still result in slight variations in IDUAL’s objective function. This may affect the found LP solution, and therewith change IDUAL’s state expansion part. Similar to Trevizan et al.’s (2017b) conjecture in the expected-cost setting, we surmise that both heuristics are theoretically identical. Compared to  $H_{\text{atomic}}^{\text{gpom}}$ ,  $H_{\text{pseq}}^{\text{gpoc}}$ ’s LP is much smaller, which explains the coverage difference.

Comparing  $H_{\text{pseq}}^{\text{gpoc}}$  with  $h^{\text{seq}}$  dead-end pruning, the latter frequently achieves considerably smaller search spaces. This is due to its stronger dead-end detection capability:  $h^{\text{seq}}$  may return  $\infty$  (identifies a dead

end), although  $H_{\text{pseq}}^{\text{gpoC}}$  returns a non-zero goal-probability estimate. To understand why this can happen, consider the following example task with initial state  $s_I = \{x \mapsto 0, y \mapsto 0\}$ , goal  $\{x \mapsto 1, y \mapsto 1\}$ , and two deterministic actions both with precondition  $\{x \mapsto 0, y \mapsto 0\}$ , one with effect  $\{x \mapsto 1, y \mapsto 2\}$ , and vice versa one with effect  $\{x \mapsto 2, y \mapsto 1\}$ . The LP corresponding to  $h^{\text{seq}}(s_I)$  is infeasible, i.e.,  $h^{\text{seq}}(s_I) = \infty$ , because both actions consume the initial state’s facts, which are available just once, while at the same time both actions need to be executed once to produce every goal fact at least once. In contrast, setting the actions’ probabilistic operator-counts to  $\frac{1}{2}$  yields a feasible solution to the  $H_{\text{pseq}}^{\text{gpoC}}(s_I)$  LP; the initial state’s facts are consumed  $\frac{1}{2} + \frac{1}{2} = 1$ , and  $H_{\text{pseq}}^{\text{gpoC}}$  no longer insists in achieving all goal facts with full certainty. As both goal facts are achieved  $\frac{1}{2}$  times, thus  $H_{\text{pseq}}^{\text{gpoC}}(s_I) = \frac{1}{2}$ . Conversely, however, by taking into account probabilistic information,  $H_{\text{pseq}}^{\text{gpoC}}$  is able to return estimates  $< 1$  for non dead ends, or dead ends not recognized by  $h^{\text{seq}}$ . This has proved beneficial in Pentest without budget limit.

Consider the effect of the landmark constraints. Using both  $H_{\text{pseq}}^{\text{gpoC}}$  (without landmarks) and  $h^{\text{FF}}$  dead-end detection for search guidance is almost equivalent to choosing for each instance the best of the two standalone configurations. In some cases, the combination creates synergistic effects, showing in much smaller search spaces compared to the individual components. This is most striking in ExplodingBlocks-b. Comparing this  $H_{\text{pseq}}^{\text{gpoC}}$  dead-end pruning combination with the  $H_{\text{pseq+lm}}^{\text{gpoC}}$ , the latter frequently yields much smaller state space sizes. Recall that the only difference between the two are the landmark constraints in the latter’s operator-counting heuristic. This hence shows that the additional landmark constraints can indeed be very beneficial in obtaining tighter goal-probability bounds. In particular, notice that  $H_{\text{pseq+lm}}^{\text{gpoC}}$  yields the by far smallest search spaces on the cyclic Rovers and TPP domains.

Lastly, notice that although being the computationally most expensive heuristic in our comparison,  $H_{\text{pairs}}^{\text{gpom}}$  also resulted in the smallest search spaces almost throughout the benchmark set, the only exception being Canadian RCP domains.

# 17. Discussion

We close this part by briefly summarizing the presented material. Moreover, we discuss related and possible future work as relevant to this part of the thesis.

## 17.1. Summary

Optimal goal-probability analysis in probabilistic planning is a notoriously hard problem, to the extent that the amount of work addressing it is limited. In this part, we provided a comprehensive review of the theoretical foundation. We examined weaker objectives that permit bounded sub-optimal solutions, and developed a large design algorithm space featuring known, adapted, and new algorithms addressing these problems. Our experimental evaluation clarified the empirical state of the art.

We considered explicit state-space search methods in the form of value iteration and heuristic search. A major advantage of heuristic search is its potential to find an optimal solution without visiting the whole state space. This is accomplished by guiding the value-update procedure along transitions *greedy* on the current value function. Sub-optimal regions of the state space are implicitly ignored. We adapted different well-known, as well as proposed a new family of heuristic search algorithms, equipped them with early termination conditions exploiting the weaker goal-probability objectives, and we showed their correctness under appropriate conditions.

To guarantee optimality, heuristic search requires an *admissible* initialization of the values, i.e., upper bounds on the states' goal probabilities. Yet, due to subtleties in the context of goal-probability analysis, this can *trap* heuristic search into cycles, preventing it from converging to an optimal solution. In order to remedy the situation, Kolobov et al. (2011) have proposed FRET, an iteration of multiple heuristic searches, interleaved with trap identification and removal steps. We revisited FRET in the form introduced by Kolobov et al., and introduced a new variant. Both versions differ in the state space subgraph considered for the trap analysis; the former explores *all* options greedy on the values provided by heuristic search, while the latter restricts the analysis onto the policy returned by the search. We proved correctness of both variants by reformulating, in terms of standard MDP notions, and extending the arguments provided by Kolobov et al. Furthermore, we showed that the analysis conducted by this new FRET variant can be viewed as a natural extension of solved-labeling mechanisms readily implemented by various heuristic search algorithms. As a result, we obtained new algorithm variants that natively support goal-probability analysis for MDPs in general, without the need of any FRET outer-loop. While our exposition focused on goal-probability analysis specifically, the necessary changes can straightforwardly be extended to the more general class of GSPs (Kolobov et al., 2011).

Our experiments showed that heuristic search can be beneficial by itself, even if starting from the trivial goal-probability upper bound 1 everywhere. To further improve the effectiveness, we explored two strands of approaches. On the one hand, we considered goal-probability preserving state space reduction techniques, based on the well-known notion of *bisimulation* and via dead-end pruning. More specifically,

we showed how to use the merge-and-shrink framework from classical planning as a means to effectively constructing a *probabilistic bisimulation*. The distinction between bisimilar states can be dropped without sacrificing correctness. Secondly, we utilized classical planning heuristics in combination with the *all-outcomes determinization* to identify *dead ends*. By treating such as states as being terminal, one avoids the construction of state-space parts irrelevant for the goal-probability objectives. Moreover, dead-end detection yields a non-trivial value initialization, which may be exploited by heuristic search for even more pruning. For budget-limited planning, we used admissible classical planning heuristics to identify and prune states with insufficient remaining budget. On the side, we observed that the landmarks compilation as per Domshlak and Mirkis (2015) reducing the remaining budget is, on its own, equivalent to pruning against the remaining budget with a standard admissible landmark heuristic in terms of dead-end detection power. The proposed state space reduction methods are not limited to heuristic search specifically, but can be useful in all search algorithms.

On the other hand, we considered the derivation of *quantitative* goal-probability estimates, going beyond pure dead-end detection. To this end, we revised Trevizan et al.'s (2017b) *occupation-measure* and *operator-counting* expected-cost heuristics. Besides changes necessary as per the goal-probability context, we extended the occupation-measure heuristics to arbitrary *projections*. We provided a generic definition of probabilistic operator-counting heuristics, which more closely resembles the original definition from classical planning (Pommerening et al., 2014), while avoiding the detour via the all-outcomes determinization. To instantiate this framework, we re-interpreted the *state equation* (Bonet, 2013) for goal-probability analysis, and showed how to map landmarks into probabilistic operator-counting constraints. We proved that the proposed heuristics monotonically upper bound goal probability, as needed for the correctness of heuristic search.

We investigated the behavior of the resulting algorithm design space on a large benchmark suite we designed for that purpose. In summary, we observed that heuristic search yields substantial benefits, when combined with dead-end pruning. The ability of *early termination* significantly improves the performance on the weaker objectives. Our FRET variant is much better suited for the goal-probability objective, and its native embedding into the heuristic search algorithms further reduces the FRET-related overhead. Bisimulation reduction yields an optimal MaxProb solver that excels in TriangleTireworld. The occupation-measure and operator-counting heuristics could not entirely keep their promises. While they provided informative goal-probability bounds in some cases, this increase in informativeness could not compensate for the computational cost associated with the heuristic computations.

## 17.2. Related Work

Teichteil-Königsbuch et al. (2010) and Camacho et al. (2016) consider goal-probability maximization, but do not aim at guaranteeing optimality. MaxProb also partly underlies the International Probabilistic Planning Competitions (Younes et al., 2005; Bryce and Buffet, 2008), when planners are evaluated by how often they reach the goal in online policy execution. The time limit in the IPPC setting mixes MaxProb with a bias towards policies reaching the goal quickly. This also relates to the proposals by Teichteil-Königsbuch (2012) and Kolobov et al. (2012a) asking for the cheapest policy among those maximizing goal probability, and to the proposal by Chatterjee et al. (2016) asking for the cheapest policy ensuring that a target state is reached almost surely in a partially observable setting.

Limited-budget planning has been studied in various forms before. Our probabilistic variant was previously considered by Hou et al. (2014). It differs from *constrained MDPs* (Altman, 1999) in carrying the

remaining budget locally with each state, instead of globally enforcing a limit on the total expected cost of a policy. With uniform action outcome cost, it simplifies to a finite-horizon setting, or step-bounded reachability analysis. In a deterministic setting, budget-limited planning appeared in an *oversubscription* context, the objective being to maximize the reward from achieved (soft) goals subject to the budget (Domshlak and Mirkis, 2015). In classical-planning it relates to *resource-constrained planning* (e.g., Haslum and Geffner, 2001; Nakhost et al., 2012; Coles et al., 2013). There also exist work on probabilistic planning with resources (Marecki and Tambe, 2008; Meuleau et al., 2009; Coles, 2012), but these usually deal with stochastic and continuous resource consumption.

The analysis of reachability properties in systems exhibiting stochastic behavior is a fundamental concern in quantitative model checking. Goal-probability analysis directly relates to the verification of PCTL (Probabilistic Computation Tree Logic) formulae (Hansson and Jonsson, 1994) on systems modeled as MDPs (e.g., Bianco and de Alfaro, 1995; de Alfaro, 1997; Baier and Kwiatkowska, 1998). PCTL is an extension of Clarke et al.’s (1986) branching-time logic CTL by operators bounding the likelihood of wanted (or unwanted) system execution paths, and as such links closely to our AtLeastProb objective. The most common approach to validate a given PCTL formula is using VI to compute the maximal (or minimal) probability of satisfying the corresponding reachability property, i.e., solving MaxProb, and comparing the result against the formula’s probability bounds afterwards. While this principally constitutes one particular point in the space of algorithms that we explored, probabilistic model checkers typically feature additional improvements to VI to cope with the state explosion problem. A variant especially worth mentioning is *symbolic VI* (Kwiatkowska et al., 2002), which, motivated by earlier work on model checking non-stochastic systems (e.g., McMillan, 1993), exploits symbolic data structures to compactly represent the state space and the value function. VI can be implemented with operations that run in time and space polynomial in the size of the data structure. This may yield a significant advantage, as the symbolic representation has the potential of being exponentially more compact than the represented system.

Brázdil et al. (2014) considered MDP heuristic search in the form of RTDP (Barto et al., 1995) and BRTDP (McMahan et al., 2005) for verifying PCTL properties. To prevent search from getting trapped in sub-optimal fixed points, they identify and remove traps on-the-fly during the search. Contrary to FRET, they need to run only a single iteration of heuristic search. In contrast to our trap-aware heuristic search adaptations, trap identification and removal steps are not executed as necessary upon encountering a trap, but periodically during (B)RTDP’s trials, once every some predetermined number of steps. Moreover, their trap analysis, and consequently the correctness argument, differs from FRET and our trap-aware adaptations in terms of the MDP subgraph considered. Klauck and Hermanns (2021) presented an extension of LRTDP, and explored its combination with FRET- $\pi$  for model checking a broad range of different MDP objectives, including MaxProb, its opposite “MinProb”, as well as various other reward-centered objectives.

The fundamental relationship between goal-probability analysis in probabilistic planning and the verification of reachability properties in quantitative model checking, on the one hand, and the to a large extent independent development of algorithmic improvements, on the other hand, ask for systematic comparison of the state of the art from both communities. As mentioned in Section 1.5.2, we (Klauck et al., 2018; Klauck et al., 2020) did so via automatic translations between PPDDL (Younes et al., 2005) and Jani (Budde et al., 2017), a model specification language widely supported by probabilistic model checkers. In an extensive empirical comparison between PROBABILISTIC FAST DOWNWARD and various state-of-the-art probabilistic model checkers, we could show that the heuristic search approaches available in the former, as well as the VI variants available in the latter tools can be superior in different situations. In particular, heuristic search outperforms VI, and its variants, on some standard model-checking benchmarks, while at the same time, symbolic VI excels in some planning benchmarks.

### 17.3. Future Work

There are many promising future directions, of which we would like to emphasize:

**Symbolic heuristic search algorithms** Similar to symbolic VI, there have been attempts in creating symbolic MDP heuristic search algorithms (Teichteil-Königsbuch, 2005). However, these variants have to our knowledge not yet been applied to goal-probability analysis, and appropriate extensions, such as symbolic FRET, still need to be developed.

**State-space reduction via dominance relations** Goal probability can only be higher in dominating states, raising the opportunity to prune dominated regions and/or transfer upper/lower bounds across states. State dominance is ubiquitous in limited-budget planning (and resource-constrained planning). More general dominance relations have been shown to exist and to be automatically computable in a classical planning context (Torralba and Hoffmann, 2015). The transfer of these techniques to the probabilistic setting is a promising line of future work.

A big open research challenge remains the development of new, more precise or more efficient, approaches to compute goal-probability bounds for heuristic search. We want to point out three options that appear particularly promising:

**Goal-probability potential heuristics** Given the relation between potential heuristics and LP heuristics observed in classical planning (Pommerening et al., 2015), can we extend this connection to the probabilistic setting, and with that obtain admissible goal-probability potential heuristics? Let us briefly recall Chapter 7. Potential heuristics form simple linear combinations of real-valued state-feature weights. As Pommerening et al. showed, the feasible solutions to the dual of the classical-planning state-equation LP characterize exactly the weights corresponding to admissible facts potential heuristics. Interestingly, this result can be straightforwardly extended to our probabilistic setting, i.e., one can show that the dual of the probabilistic state-equation LP yields fact potential heuristics that monotonically upper bound goal probability (cf. Appendix C.4). However, what about potential heuristics using more complex state features? In particular, Pommerening et al. (2017) showed that admissible classical-planning fact-pair potential heuristics can still be characterized exactly through an LP encoding polynomial in the size of the task. In Chapter 7, we have analyzed the use of the compilation  $\Pi^C$  to construct admissible classical-planning potential heuristics over arbitrary conjunction sets. Can these techniques be adapted to compute conjunction potential heuristics that admissibly bound goal probability? Complications arise, e.g., from the transition normal form (TNF) assumption (the standard TNF transformation from classical planning (Pommerening and Helmert, 2015), introducing auxiliary preconditions, no longer works for probabilistic planning tasks, given that not all action outcomes must affect the same variables and therefore can prevail that auxiliary precondition).

**Optimal “outcome-probability partitioning” for goal-probability heuristics** We recently noted that syntactic projections can also yield informative goal-probability upper bounds without surrounding LP construction (Klößner et al., 2021). A major advantage over the projection occupation-measure heuristic is that the goal-probability bounds can be computed upfront, before search, through a single application of value iteration on the syntactic projection, simplifying the heuristic calls during search to simple table lookups. We showed that the individual estimates of such projection heuristics can always be combined admissibly by taking the minimum, and furthermore, identified conditions where this is even guaranteed when

taking the *product* (which dominates the minimum). Such multiplicative projection heuristic ensembles empirically compare favorably to the projection occupation-measure heuristic. In the context of expected-cost MDPs, it is known that the projection occupation-measure expected-cost heuristic computes an *optimal cost partitioning*, i.e., computes the best possible admissible additive combination of the considered expected-cost projections (Klößner et al., 2022a). In the goal-probability setting, the relation between occupation-measure heuristics and (multiplicative) projection heuristic ensembles is however not so clear anymore. In fact, it turns out that both heuristic classes are in general incomparable (we provide examples in Appendix C.5). This raises a series of questions. First and foremost, can we find a notion that, analogously to cost partitioning for expected cost, captures the best possible admissible multiplicative combination of goal-probability heuristics? Is it possible to efficiently compute this best combination? And how does the goal-probability occupation-measure heuristic relate to this notion?

**Goal-probability bounds via state-space abstractions** In decades of research on classical planning heuristics, various other, more flexible, classes of state *abstractions* have been explored (e.g., Helmert et al., 2014; Seipp and Helmert, 2018). Applying them to the stochastic setting is promising, in principle, but comes with its own difficulties. State abstractions can be viewed as equivalence relations  $\sim$  between states; its semantics being captured in terms of the *abstract state space*, i.e., a transition structure over  $\sim$ 's equivalence classes that retains the original transition behavior. A particularity arising in the probabilistic context is that the abstract state space of abstractions in general can no longer be expressed as an MDP: if  $s \sim t$  where  $\mathcal{P}(s, a, s') \neq \mathcal{P}(t, a, s')$  for some  $s'$ , then the transition probability from the equivalence class  $[S]_{\sim} \supseteq \{s, t\}$  via  $a$  may no longer be unique. *Bounded-parameter MDPs* (Givan et al., 2000) allow to specify transition probabilities as intervals, yet are susceptible to pathologies, where distinguishing between more states can result in worse heuristic estimates (Tagorti et al., 2013). An alternative view on MDP abstractions are stochastic two-player games, in which one player resolves the state-action choices (as in MDPs), while the other player decides the transition probabilities for the chosen state-action pair (Kwiatkowska et al., 2006). By accordingly instantiating both players, one can infer lower and upper bounds on the minimal and maximal goal probabilities. Beyond the exploration of the space of potential abstract state space representations, a major open topic is how to actually construct “good” abstractions, i.e., how to efficiently find some  $\sim$  that distinguishes only so many states so that the heuristic computation remains feasible, yet that distinguishes enough states to obtain informative bounds. A possible starting point may be given by the vast research on abstraction methods in the context of quantitative model checking (e.g., D’Argenio et al., 2001; Kwiatkowska et al., 2006; Hermanns et al., 2008).



## Part IV.

# Conflict-Driven Learning in Probabilistic-Planning State-Space Search

Dead-end detection is an integral part of guiding MDP heuristic search for goal-probability analysis. For forward state-space search in classical planning, we presented a conflict-driven learning approach that has the ability to produce accurate dead-end predictors during the search by utilizing information that becomes available because of the search. This essentially relied on two ingredients (1) the *identification* of conflicts, i.e., dead ends encountered in search that the predictor failed to recognize, and (2) techniques to explain the situation at those conflicts so to *refine* the predictions, preventing the consideration of similar dead ends in the remainder of the search. Here, we lift those ideas towards learning to recognize dead ends during MDP state-space search. We show how to identify conflicts in the conglomerate of MDP state-space search algorithms developed in Part III, doing so in a sound and complete manner. For dead-end prediction and refinement, we leverage once again the interface to classical planning via the all-outcomes determinization. The techniques developed in Part II then plug in directly. Our experiments demonstrate that the attained dead-end learning methods can indeed be an effective means for MaxProb analysis. Even when using nothing but the learned dead-end information for pruning, the performance is on par with, and sometimes even stronger than, that of state-of-the-art static dead-end detectors.



# 18. Conflict-Driven Learning in MDP State-Space Search for Goal-Probability Analysis

Applying the conflict-driven learning methodology as introduced in classical planning to MDP state-space search is in principle straightforward. Regarding conflict explanation and refinement, thanks to the all-outcomes determinization (Bonet and Geffner, 2005; Jimenez et al., 2006), the techniques developed in Part II work plug and play. There only remains the question of how to identify conflicts. But, for this, thanks to the Bellman updates, the heuristic search algorithms seem to provide a solution out of the box: whenever the maintained goal-probability upper bound of some unrecognized dead-end state  $s$  becomes 0, we know that  $s$  is a dead end, and hence have found a conflict. Unfortunately, this comes with some issues. Information available in the search graph, i.e., in the state-space subgraph unveiled by search so far, is not directly reflected on the value function. This would hence delay the identification of dead ends. Yet recall from Part II that quick conflict identification is key for conflict-driven learning to be effective. Moreover, value updates are not exact, i.e., with commonly used stopping conditions like  $\epsilon$ -convergence, the values of some states may never become small enough proving them to be dead ends, thus further hampering the dead-end learning potential. Lastly, in practice, there would be a chance of incorrectly classifying a state as dead end due to inaccuracies in the floating-point number representation.

To avoid all these difficulties, we fall back to the procedures presented in Chapter 5, analyzing the search graph directly. Soundness, i.e., that no solvable state is ever misclassified as conflict, is guaranteed by construction. By carefully initiating the analyses, one can furthermore guarantee completeness, i.e., to discover all currently known conflicts in the identification pass.

This chapter is structured as follows. In Section 18.1, we briefly revisit the relevant notions. In Section 18.2, we provide a formal characterization of what it means for a conflict to become known, considering a reasonably generic view of MDP state-space search. Section 18.3 integrates the conflict identification method from Chapter 5 into different MDP heuristic search algorithms, and shows how to make best use of the learned dead-end information in those algorithms. Section 18.4 designs depth-first search and VI variants, which similarly to our classical-planning depth-first search variant, have a structure predetermined for the conflict-driven learning approach. Finally, Section 18.5 evaluates the impact of dead-end learning on the different algorithms for goal-probability analysis, considering the cyclic and acyclic benchmarks from Chapter 16, and using a selection of the dead-end detection and refinement methods from Part II.

## 18.1. Preliminaries

Let  $\Pi$  be a probabilistic planning task with state space  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$ . We assume to be provided with an unsolvability detector  $\mathcal{U}$  for  $\mathcal{M}$ , i.e., a function  $\mathcal{U}: \mathcal{S} \mapsto \{0, \infty\}$  such that  $\mathcal{U}(s) = \infty$  only if  $s$  is a dead end in  $\mathcal{M}$ . Recall from Chapter 15 that such a  $\mathcal{U}$  can be obtained from any classical planning unsolvability detector via the all-outcomes determinization  $\Pi_D$ . Furthermore, we assume that there is a refinement operation available, which, given one or multiple unrecognized dead ends, i.e., dead ends  $s$

where  $u(s) = 0$ , updates  $u$  into  $u'$  such that (1)  $u'$  generalizes  $u$ , i.e.,  $u'$  still recognizes the dead ends recognized by  $u$ , and (2)  $u'(s) = \infty$  for the given states  $s$ , making  $u'$  in fact strictly more general. Recall from Proposition 15.1 that the dead ends of a probabilistic planning task  $\Pi$  are exactly the dead ends of the all-outcomes determinization  $\Pi_D$ . Hence, in particular, the conflict refinement procedures from Part II combined with the all-outcomes determinization perfectly suite the aforementioned requirements. Finally, recall that a set of states  $S$  satisfies the  $u$  recognized-neighbors property in the all-outcomes determinization if, for the neighbor states  $T = \text{Succ}[\Theta^{\Pi_D}](S) \setminus S$ , it holds that  $u(t) = \infty$ , for all  $t \in T$ . Observe that this holds if and only if  $S$  satisfies the  $u$  recognized-neighbors property in  $\mathcal{M}$ , as  $\text{Succ}[\Theta^{\Pi_D}](S) = \text{Succ}[\mathcal{M}](S)$  holds by the definition of  $\Pi_D$ .

## 18.2. Conflict Characterization

The search algorithms from Chapters 11 and 12 have in common to incrementally build and to operate on an MDP subgraph  $\hat{\mathcal{M}} = \langle \hat{S}, \mathcal{A}, \hat{P}, s_I, \hat{S}_* \rangle$  of  $\mathcal{M}$ . We refer as the *tip states* of  $\hat{\mathcal{M}}$  to all (non-pruned) states  $tip \subseteq \hat{S}$ , whose successors and transitions have not yet been inserted into  $\hat{\mathcal{M}}$ .

The general idea now is the same as in classical planning. We use  $u$  for pruning dead-ends during the exploration and expansion of  $\hat{\mathcal{M}}$ . At which point  $u$  is evaluated exactly does not matter for the following discussion, we specify the options below. It suffices to note that at any point in time, every state  $s \in \hat{S}$  falls into one of the following categories: (i)  $s \in tip$ , (ii)  $s \in \hat{S}_*$ , (iii)  $\text{Succ}[\mathcal{M}](s) = \text{Succ}[\hat{\mathcal{M}}](s)$ , or (iv)  $u(s) = \infty$ . Every dead end in  $\hat{\mathcal{M}}$  that is not recognized by  $u$  is a *conflict*. We attempt to learn from conflicts, refining  $u$ , so to recognize and prune similar dead ends in the remainder of the search. Note that these extensions do not affect the correctness guarantees of any of the algorithms as per the same arguments justifying why dead-end pruning is safe (cf. Chapter 15). It now only remains to specify how to know exactly whether a state in  $\hat{\mathcal{M}}$  is a conflict.

Following the concepts from Chapter 5, a dead end  $s \in \hat{S}$  becomes *known* when  $\hat{\mathcal{M}}$  has unveiled enough of the state space  $\mathcal{M}$  to know for sure that no goal state can be reached from  $s$ . In other words,  $s$  becomes known when all future state expansions in search cannot make a goal state become reachable from  $s$ . Analogous to Chapter 5, the future search graphs can be captured formally as all MDPs that *coincide* with the present one on all non-tip states (those that were expanded); where two MDPs coincide on a set of states  $S$  if they agree on all  $S$ -leaving transitions. The complete definition is almost identical to the classical planning case, and omitted for the sake of brevity. The currently known dead ends are given by

**Definition 18.1** (Known Dead End). *Let  $s \in \hat{S}$  be any state visited so far.  $s$  is a **known dead end** given  $\hat{\mathcal{M}}$  if  $s$  is a dead end in every MDP  $\mathcal{M}'$  that coincides with  $\hat{\mathcal{M}}$  on  $\hat{S} \setminus tip$ .*

Like in the classical setting, the known dead ends can obviously not have any tip-state descendant in  $\hat{\mathcal{M}}$ . Yet, unlike in the classical setting, a state that cannot reach any tip state must not necessarily be a dead end. Capturing the known dead ends of  $\hat{\mathcal{M}}$  additionally entails explicitly validating goal unreachability:

**Proposition 18.1.** *At any point during MDP state-space search, the known dead ends are exactly those states  $s \in \hat{S}$  where  $\mathcal{R}[\hat{\mathcal{M}}](s) \cap (tip \cup \hat{S}_*) = \emptyset$ .*

This follows via the same arguments as given for Proposition 5.1.

Clearly, the definition of *known dead ends* is sound in the following sense:

**Proposition 18.2.** *If  $s$  is a known dead end given  $\hat{\mathcal{M}}$ , then  $s$  is a dead end in  $\mathcal{M}$ .*

*Proof.* Suppose for contradiction that  $s$  was not a dead end in  $\mathcal{M}$ . Let  $\sigma = \langle s_0 = s, a_1, s_1, \dots, a_n, s_n \rangle$  be a path from  $s$  to some goal state  $s_n \in \mathcal{S}_*$  in  $\mathcal{M}$ . If  $s_n \in \hat{\mathcal{S}}$ , then  $s_n \in \hat{\mathcal{S}}_*$ . Since  $s$  is a dead end in  $\hat{\mathcal{M}}$ ,  $\sigma$  can hence not be a path in  $\hat{\mathcal{M}}$ . Since none of the states are dead ends, it follows that  $u(s_i) = 0$  for all  $i$ . Then, of the four cases (i) – (iv) above, for states  $s_i$ ,  $0 \leq i < n$ , there remains only the possibility of (i)  $s_i \in \text{tip}$ , or (ii)  $\text{Succ}[\mathcal{M}](s) = \text{Succ}[\hat{\mathcal{M}}](s)$ . (ii) cannot hold for all of them. In conclusion, there exists at least one tip state that is reachable from  $s$ . But, this contradicts Proposition 18.1.  $\square$

### 18.3. Conflict-Driven Learning in MDP Heuristic Search

It remains to specify how to verify Proposition 18.1 efficiently, and how to leverage the learned dead-end information, i.e., when to evaluate  $u$ . We start with the latter.

#### 18.3.1. Utilizing the Learned Knowledge

To understand how to best utilize the learned dead-end information, let us briefly revisit the concepts of MDP heuristic search as per Chapter 12. Specifically, we consider those heuristic search algorithms operating on an admissible (upper-bounding) goal-probability approximation  $V^U$  (we do not consider IDUAL; and exhaustive anytime DFS is discussed in the next section). Let  $\pi^U$  be the corresponding greedy policy. Recall that  $\pi^U$  is closed in a state  $s \in \hat{\mathcal{S}}$ , if no tip state is reachable from  $s$  in the policy induced subgraph  $\hat{\mathcal{M}}^{\pi^U}$ . Furthermore, recall that  $V^U$  is called  $\epsilon$ -consistent w.r.t.  $\pi^U$  and  $s$ ,  $\epsilon \geq 0$ , if  $V^U$  is  $\epsilon$ -consistent in all states reachable from  $s$  in  $\hat{\mathcal{M}}^{\pi^U}$ . We say that heuristic search has **converged** on a state  $s \in \hat{\mathcal{S}}$  if  $\pi^U$  is closed in  $s$  and  $V^U$  is  $\epsilon$ -consistent w.r.t.  $\pi^U$  and  $s$ ;  $\epsilon$  being the termination parameter. Once converged on  $s$ ,  $V^U(s)$  and  $\pi^U(s)$  no longer change. Note that convergence is transitive, i.e., if search has converged on  $s$  and  $s'$  is reachable from  $s$  in  $\hat{\mathcal{M}}^{\pi^U}$ , then search has converged on  $s'$  as well. The goal of heuristic search is to converge on the initial state  $s_I$ . This is accomplished via series of policy graph  $\hat{\mathcal{M}}^{\pi^U}$  traversals, during which tip states (if reached) might be expanded, and  $V^U$  and  $\pi^U$  are updated. The search can be guided by using a goal-probability upper-bounding approximation  $H$  to initialize  $V^U$  and  $\pi^U$ .

Furthermore,  $H$  can be used for dead-end pruning. Namely when a state  $s$  is inserted into  $\hat{\mathcal{M}}$  for the first time,  $s$  would typically be noted down for later expansion by appending it to *tip*. If, however,  $H(s) = 0$ , then  $s$  must necessarily be a dead end, and the expansion can be safely skipped.

The unsolvability detector  $u$  used for learning can be incorporated at three different places: (1) at **generation time**, as just described, i.e., when  $s$  is inserted into  $\hat{\mathcal{M}}$ , not adding it to *tip* and initializing  $V^U(s) = 0$  if  $u(s) = \infty$ ; (2) at **expansion time**, i.e., when expanding a tip state  $s$ , pruning that expansion if  $u(s) = \infty$ ; and (3) at **reexpansion time**, i.e., when traversing other non-tip states  $s$  during the policy-graph explorations, removing the outgoing transitions of  $s$  from  $\hat{\mathcal{M}}$  if  $u(s) = \infty$ . Retesting  $u(s) = \infty$  in (b) and (c) makes sense in our setup, given that since the last evaluation of  $u(s)$ ,  $u$  could have been refined, causing  $s$  to now be recognized. (1) and (2) are similar to the adaptations in the classical planning searches (cf. Chapter 5). (3) has no correspondence in the classical searches, where every state is expanded at most once. (1), (2), and (3) together ensure **immediate convergence**, i.e., that heuristic search converges on every recognized dead end it visits during the policy-graph traversals, without further updates or explorations.

On acyclic MDPs, where  $\epsilon$ -consistency is not needed, heuristic search furthermore ensures **convergence completeness**: if heuristic search converges on a dead end  $s$ , then  $s$  must have become known. Note that

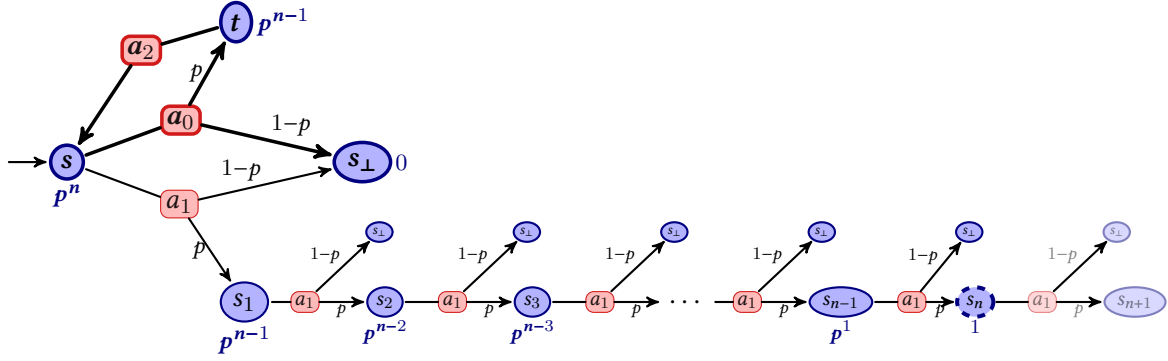


Figure 18.1.: Example MDP illustrating that  $\epsilon$ -consistency cutoffs may prevent some dead ends from ever becoming known. The SSP conditions are satisfied; no goal-probability trap analysis is needed. All states are dead ends.  $s_n$  is a tip state.  $s_{n+1}$  has not yet been inserted into  $\hat{\mathcal{M}}$ , and serves for illustration purposes only.  $s_\perp$  is duplicated for readability. States are annotated by their values under the  $V^U$  value function maintained by search. The corresponding greedy policy  $\pi^U$  is marked in bold.

this is important to guarantee that all (reachable) dead ends have a chance to become known at some point. Given that  $\pi^U$  no longer changes on converged states, and heuristic search only explores the  $\pi^U$  induced subgraph, tip states only reachable from converged states will not be expanded anymore. In particular, if a tip state is only reachable from a converged dead end, that dead end can no longer become known.

**Proposition 18.3.** *On acyclic MDPs with  $\epsilon = 0$ , when heuristic search converges on a dead end  $s$ , then  $s$  has become a known dead end in  $\hat{\mathcal{M}}$ .*

*Proof.* To show convergence completeness, suppose for contradiction that heuristic search converged on a dead end  $s$  that has not yet become known. With  $\epsilon = 0$  and as per the correctness of the heuristic search algorithm, it hence holds  $V^U(s) = 0$ . Given that  $s$  is not known,  $\hat{\mathcal{M}}$  must contain a path  $\langle s_0 = s, a_1, s_1, \dots, a_n, s_n \rangle$  such that  $s_n \in \text{tip}$  or  $s_n \in \hat{\mathcal{S}}^*$ . Since  $s$  is a dead end, the latter cannot be the case, so  $s_n \in \text{tip}$ . From the monotonicity of the value updates, it follows that  $V^U(s) \geq \prod_{i=1}^n \hat{\mathcal{P}}(s_{i-1}, a_i, s_i) V^U(s_n)$ . With  $s_n \in \text{tip}$ , it follows that  $V^U(s_n) = H(s_n) > 0$ . But this yields a contradiction to  $V^U(s) = 0$ .  $\square$

Although proving unsolvability is not the purpose of MDP heuristic search (which really is just a classical planning problem), notice that this also implies that, if the initial state  $s_I$  is unsolvable, once heuristic search has converged on  $s_I$ , it has become known. So, via the corresponding  $\mathcal{U}$  refinement, we can obtain an *unsolvability certificate* just as in the classical planning case.

Importantly, however, convergence completeness is not guaranteed on MDPs in general, where (a) heuristic search can converge on dead ends, that have not yet become known, due to goal-probability traps, and (b) with  $\epsilon > 0$ ,  $\epsilon$ -consistency may cause the exploration of  $\hat{\mathcal{M}}$  to be cut short. (a) is only a temporary issue, given that after the trap's removal, heuristic search must reestablish convergence, which allows the dead ends to become known. (b) may however prevent some dead ends from ever becoming known.

Figure 18.1 depicts an example. Note that all states are dead ends;  $s_\perp$  is the only one that is currently known. Consider the current policy  $\pi^U(s) = a_0$  and  $\pi^U(t) = a_2$  (which is greedy as per the depicted  $V^U$  values). Moreover, assume that  $\epsilon \geq p^n - p^{n-1}$ . Observe that  $V^U$  is  $\epsilon$ -consistent in all states visited by that policy, i.e., heuristic search has converged on  $s$  and  $t$ . As the tip state  $s_n$  can only be reached from  $s$  via  $a_1 \neq \pi^U(s)$ ,  $s_n$  will never be expanded. But then,  $s$  and  $t$  cannot become known. To remedy the situation,

we need a smaller  $\epsilon$  value. In the example,  $\epsilon < p^{n+1} - p^n$  would be sufficient to force heuristic search to also explore  $s_n$ . While there always exists a small enough  $\epsilon$  to prevent (b) from happening, these will typically be too small to be practical. Note that in the example, we could in fact make  $\epsilon$  arbitrarily small by scaling the path length  $n$ . Moreover, given that suitable  $\epsilon$  are task specific, and furthermore depend on the value function initialization  $H$ , it is not clear how to derive them in general. In practical regards, this means that there is a chance that some conflicts cannot be captured by the known dead end concept, which if the case, might limit the effect of conflict learning.

### 18.3.2. Conflict Identification

We finally describe how to identify the known dead ends during search. As per the algorithms in Chapter 12, we distinguish between acyclic and general MDPs.

#### Acyclic MDPs

Matters are particularly simple in the acyclic case, where a simple bottom-up labeling procedure suffices to collect all known dead ends. Specifically, we need to alter heuristic search at two places. First, when expanding a tip state  $t$ , we check whether all successors of  $t$  have been flagged known. If so, we trigger the refinement of  $\mathcal{U}$  on  $t$ , mark  $t$  as known, and repeat the test for  $t$ 's parent states in  $\hat{\mathcal{M}}$ . This process is repeated recursively until reaching a state some of whose successors were not yet marked known, or after flagging the initial state known. Secondly, we trigger the backward propagation process when a non-tip state is pruned during reexpansion.

As per Proposition 18.1, a dead end  $s$  becomes known exactly at the moment when all tip states are disconnected from  $s$ . This can happen because of two reasons: either search has expanded the last tip state  $t$  that was reachable from  $s$  in  $\hat{\mathcal{M}}$ ; or search has removed from  $\hat{\mathcal{M}}$  the outgoing transitions of some non-tip state  $t$  so that all paths from  $s$  to some tip state went through  $t$ . In both cases,  $s$  is an ancestor of  $t$ , and hence will be collected if the backward propagation process is started at  $t$ . If  $t$  was pruned during reexpansion, this holds by construction. Suppose that  $t$  was tip state. Assume as the invariant that prior to its expansion, we have identified and flagged all dead ends that were known at that time. Given that  $s$  can reach  $t$ , and  $s$  becomes known after the expansion of  $t$ , it follows that all successors of  $t$  must also be known dead ends after the expansion of  $t$ . Since  $\hat{\mathcal{M}}$  is acyclic, none of the successors of  $t$  can reach  $t$ , so these must have been already known dead ends prior to the expansion of  $t$ , and as per the invariant, are marked known. Thus,  $t$  is marked known, starting the propagation process. In conclusion

**Proposition 18.4.** *Upon the termination of the labeling procedure, the states marked known are exactly the known dead ends.*

Notice that if dead ends are pruned only based  $\mathcal{U}$ 's predictions, then as per the refinement calls, every dead end marked known is recognized by  $\mathcal{U}$ . Hence, the  $\mathcal{U}$  recognized-neighbors property is satisfied on every refinement:

**Proposition 18.5.** *Suppose that only  $\mathcal{U}$  is used for dead-end pruning. When a dead end  $s$  is marked known, then  $\{s\}$  satisfies the  $\mathcal{U}$  recognized-neighbors property.*

While the sketched conflict identification procedure can be integrated into any search algorithm, it fits particularly well the design of  $\text{AO}^*$ . The only change needed for  $\text{AO}^*$  is checking additionally in each  $\text{BackwardUpdate}(t)$  call (cf. Algorithm 12.3) whether the successors of  $t$  have been marked known, and if so also setting the solved label.  $\text{BackwardUpdate}(t)$  is called at the end of each expansion trace, as

required to start the propagation process. Moreover, the propagation of the solved labels ensures that the update function's backward pass reaches all known dead-end ancestors of  $t$ .

### General MDPs

In the presence of cycles, the bottom-up labeling procedure no longer guarantees to identify all known dead ends. Recall, however, that we already faced that problem in open & closed list based search (Section 5.1). And, it turns out that we can still leverage the same solution. All we need to do is to replace the condition from Proposition 5.1 by that from Proposition 18.1.

In summary, the procedure works as follows. Whenever expanding a tip state  $s$ , we run a lookahead search in  $\hat{M}$  starting from  $s$  and looking for a tip or goal state. If neither is found, then  $\mathcal{R}[\hat{M}](s) \cap (\text{tip} \cup \hat{S}_*) = \emptyset$ , i.e.,  $s$  is a known dead end. We refine  $\mathcal{U}$  on all states of  $\mathcal{R}[\hat{M}](s)$  not recognized so far. To then collect also the dead ends that have become known due to the expansion of  $s$ , we run backward propagation on the parents of all states of  $\mathcal{R}[\hat{M}](s)$ . To prevent the recursions from running into cycles, we label already identified dead ends, and skip recursive calls on labeled parents. Finally, by appropriately starting the backward propagation process, we ensure to also discover the dead ends that became known due to pruning a non-tip state during reexpansion. As per the arguments given in Section 5.1, the backward propagation process always terminates with all known dead ends being labeled:

**Proposition 18.6.** *Upon termination of the backward propagation procedure, the labeled states are exactly the known dead ends.*

Moreover, as per the proof arguments of Proposition 6.1, if only  $\mathcal{U}$  is used for dead-end pruning, then the  $\mathcal{U}$  recognized-neighbors property is satisfied on every refinement:

**Proposition 18.7.** *Suppose that only  $\mathcal{U}$  is used for dead-end pruning. Whenever a dead component  $\mathcal{R}[\hat{M}](s)$  is labeled during the propagation procedure, then  $\mathcal{R}[\hat{M}](s)$  satisfies the  $\mathcal{U}$  recognized-neighbors property.*

Recall that heuristic search as in LRTDP or DFHS requires certain adaptations to work on general goal-probability MDPs. To make sure that Proposition 18.6 remains satisfied, the conflict identification procedure may need to be integrated into those auxiliary processing steps as well. Specifically, this pertains to FRET- $V$ , which may expand  $\hat{M}$  beyond what was considered by its underlying heuristic search algorithm. FRET- $\pi$  and the trap-aware heuristic search variants do not require any further adaption besides calling the identification method during the heuristic searches.

Finally, notice that the conflict identification procedure also works out-of-the-box for IDUAL, given the similarity of its state-space exploration part to the other heuristic algorithms. Yet, we do not further explore this option here.

## 18.4. Conflict-Driven Learning in Topological VI & Anytime DFS

Exhaustive depth-first search has turned out to be especially effective in quickly making dead ends become known, which fosters learning, and therewith increases the potential for pruning. While the MDP heuristic searches discussed before also have a strong depth bias, their explorations are non-exhaustive due to their focus on the policy induced subgraphs. To leverage the full potential of depth-first search for dead-end learning, we equip our ExhDFS variant (Algorithm 12.13) with the ability to identify known dead ends. Topological VI (TVI, cf. Algorithm 11.1) can be adapted in the exact same manner.

The modifications follow our observations from Section 5.2 for DFS in classical planning. In a nutshell, we evaluate  $\mathcal{U}$  prior to every state expansion, skipping the expansion of recognized dead ends. To account for  $\mathcal{U}$  refinements, we reevaluate  $\mathcal{U}$  on  $s$  whenever the exploration backtracks to  $s$ , cutting off the expansion of the remaining successors when  $\mathcal{U}(s) = \infty$ . A dead end  $s$  becomes known exactly at the moment when search backtracks out of the maximal SCC  $S$  containing  $s$ . This is so because, the states from  $S$  have exactly the same descendants in  $\hat{\mathcal{M}}$ . In other words, as long as any one of  $S$  can still reach a tip state, none of the states of  $S$  can become known; while once no more tip states are reachable, search necessarily needs to backtrack out of  $S$ . In contrast to the classical case, though, not every SCC out of which we backtrack constitutes a known dead end. To check that no goal was reachable, we maintain and propagate goal-reachability flags during the depth-first exploration. When backtracking out of an SCC  $S$  from which a goal state was reachable, we run VI on  $S$  until  $\epsilon$ -convergence as before. Yet, when a dead-end SCC is identified, we skip running VI ( $V^L(s) = 0 = V^*(s)$  is satisfied as per the initialization), and instead refine  $\mathcal{U}$ . These are all changes.

Notice that due to its exhaustive state-space exploration, upon termination of topological VI, every reachable dead end must have become known (this does not necessarily hold for ExhDFS due to its early-termination criteria). This means that, if  $\mathcal{U}$  is the only method used for pruning dead ends and  $\mathcal{U}$  is transitive, the final  $\mathcal{U}$  recognizes all reachable dead ends, i.e.,  $\mathcal{U}$  is perfect on the reachable states. Moreover, observe that, both TVI and ExhDFS inherit the properties discussed in Section 5.2 for DFS in classical planning. In particular, they (1) deliver an unsolvability certificate upon termination when  $V^*(s_f) = 0$ . This holds even for cyclic MDPs and regardless of  $\epsilon$ , as opposed to the cyclic MDP heuristic search algorithms from before. Moreover, they guarantee (2) to immediately backjump to the shallowest non-refuted state; and (3) guarantee refinements on all  $\mathcal{U}$ -known dead ends prior to the next state expansion.

## 18.5. Experimental Evaluation

We evaluate MDP search with conflict-driven learning for optimal goal-probability analysis. We focus primarily on MaxProb. But, all observations equally apply to the weaker AtLeastProb and ApproxProb objectives. We next specify the general experiment setup, then discuss the results.

### 18.5.1. Experiment Setup

Our implementation is based on PROBABILISTIC FAST DOWNWARD from Part III. The source code is publicly available.<sup>1</sup> The experiments were run on a cluster of Intel Xeon E5-2660 machines running at 2.20 GHz, with time and memory cut-offs of 30 minutes and 4 GB.

We experiment with five search algorithms: topological VI; ExhDFS; AO\*; LILAO\* (ILAO\* using solved labels instead of VI for the termination test), as a representative of the DFHS algorithm family; and LRTDP. We use the default tie-breaking setups throughout. AO\* is restricted to acyclic MDPs. We use the trap-aware variants of LILAO\* and LRTDP on cyclic MDPs to handle goal-probability traps, which as shown in Part III, tends to be more effective than FRET. For  $\epsilon$ -convergence, we set  $\epsilon = 5 \cdot 10^{-5}$  as in our previous experiments.

We run every search algorithm with four different conflict learning methods, the most competitive ones from Part II: critical-path unsolvability detector  $\mathcal{U}^C$  using neighbors refinement; and  $\mathcal{U}$ -trap learning for three different  $\mathcal{U}$ : using no additional unsolvability detector, which we will refer to as the naive unsolvability detector  $\mathcal{U}^0$  that returns 0 for all states; the critical-path  $\mathcal{U}^1$  unsolvability detector (Haslum and

<sup>1</sup><https://doi.org/10.5281/zenodo.6992688>

Geffner, 2000); and the unsolvability PDB heuristic component  $\mathcal{U}^{\text{pdb}}$  of the UIPC’16 winning Aidos portfolio (Seipp et al., 2016). We compare the learning configurations to search without learning, using  $\mathcal{U}^1$  respectively  $\mathcal{U}^{\text{pdb}}$  for dead-end pruning. This covers the best performing pruning variants from our experiments from Part III. We additionally include results using no pruning at all, again denoted as  $\mathcal{U}^0$ ; and we run the heuristic search algorithms with Klößner et al.’s (2021) overall best *multiplicative goal-probability PDB heuristic*, using the *systematic patterns* with up to 3 variables (Pommerening et al., 2013).

We adopt the cyclic and acyclic benchmark suites from Part III, which encompass budget-limited and budget-unlimited variants of various IPPC, Canadian RCP, and pentesting benchmarks. We consider only domains with dead ends (this affects only the cyclic benchmarks), i.e., those domains where the conflict-driven learning approach could make sense. For the acyclic part, recall that the budget limit is *controlled* in that the initially available budget is set to  $\mathfrak{C} * \mathbf{b}_{\min}$ , where  $\mathbf{b}_{\min}$  is the minimal budget required to reach the goal at all, and  $\mathfrak{C} \in \{1.0, 1.2, 1.4, 1.6, 1.8, 2.0\}$  is the constrainedness level. The smaller  $\mathfrak{C}$  is, the more dead ends there will be. This makes  $\mathfrak{C}$  particularly interesting for our study. Given that the unsolvability detectors and refinement methods do not support an explicit budget, we adopt the benchmark variants from Section 16.4, which enforce the budget limit directly at the level of the PPDDL model files.

The remainder of this section is structured as follows. In Section 18.5.2, we consider MaxProb coverage results as means to compare the different conflict learning methods to each other and to the state of the art. Section 18.5.3 and 18.5.4 provide additional details on the performance of conflict-driven learning in MDP search. For the sake of comprehensibility, we focus in those part mostly on a single search algorithm, LILAO\*. We analyze the effectiveness of the different search algorithms in leveraging the conflict-driven approach in Section 18.5.5.

### 18.5.2. Coverage Analysis

Table 18.1 shows the coverage results for ExhDFS and LILAO\*. The results for VI are qualitatively similar to that for DFS; the other heuristic algorithms behave similarly to LILAO\*. We provide additional details on the differences between the different search algorithms in Section 18.5.5.

Consider first the non-learning variants. The results are in line with the experiments in Chapter 16. In summary, on the cyclic benchmark part, there are generally only little differences. The exhDFS configurations perform basically identically. For LILAO\*, differences are primarily due to ExplodingBlocks. In Schedule and SearchAndRescue, there is only the “blind”  $\mathcal{U}^0$  configuration which (negatively) stands out. The differences become larger on the acyclic benchmark part. Comparing the two unsolvability detectors,  $\mathcal{U}^1$  has the edge on the cyclic benchmarks (due to ExplodingBlocks), and  $\mathcal{U}^{\text{pdb}}$  is more effective on the acyclic benchmarks. Heuristic search based on an admissible value function (here represented by LILAO\*) outperforms search without such bound (exhDFS) consistently, achieving higher coverage in almost every domain. The difference grows the more informed the value function initialization (the pruning) gets. The most notable exception is budget-limited Random, where LILAO\* lacks behind exhDFS no matter of the pruning method. Comparing the multiplicative goal-probability PDB heuristic “PPDB” with PDB-based dead-end pruning “ $\mathcal{U}^{\text{pdb}}$ ” in LILAO\*, the more informative goal-probability bounds offered by the former pay off mainly on the Pentest benchmarks, and to a smaller extent in TriangleTireworld-b. On all other domains, dead-end pruning via  $\mathcal{U}^{\text{pdb}}$  performs as good, and often even better than PPDB. This is in line with Klößner et al.’s (2021) observation that for the goal-probability PDB heuristic to provide more information than mere dead-end detection, one typically needs to consider larger patterns, in particular larger than the size-3 patterns considered by PPDB; but generating such is out of scope of the systematic approach deployed by PPDB.

		exhDFS							LILAO*							
		w/o learning			w/ learning				w/o learning				w/ learning			
		$\mathcal{U}$ -trap							$\mathcal{U}$ -trap							
Domain	#	$u^0$	$u^1$	$u^{pdb}$	$u^C$	$u^0$	$u^1$	$u^{pdb}$	$u^0$	$u^1$	$u^{pdb}$	PPDB	$u^C$	$u^0$	$u^1$	$u^{pdb}$
Cyclic IPPC Benchmarks (with Dead Ends)																
Drive	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
ExplBlocksw	30	11	11	11	11	11	11	11	10	<b>28</b>	19	19	23	10	<b>28</b>	19
RectTireworld	14	<b>14</b>	<b>14</b>	<b>14</b>	<b>14</b>	12	<b>14</b>	<b>14</b>	<b>14</b>	<b>14</b>	<b>14</b>	<b>14</b>	<b>14</b>	12	<b>14</b>	<b>14</b>
Schedule	30	7	7	7	7	6	7	7	6	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	6	<b>10</b>	<b>10</b>
SearchRescue	15	5	<b>6</b>	5	<b>6</b>	<b>6</b>	<b>6</b>	5	5	<b>6</b>	<b>6</b>	<b>6</b>	5	5	5	5
Tireworld	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
$\Sigma$ IPPC	119	67	68	67	68	65	68	67	65	<b>88</b>	79	79	82	63	87	78
Canadian RCP Benchmarks without Budget Limit																
NoMystery	10	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
Rovers	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
TPP	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
$\Sigma$ RCP	30	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25
Acyclic IPPC Benchmarks (with Dead Ends)																
TriTireworld	10	7	7	7	7	7	7	7	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>
IPPC Benchmarks with Compiled Budget Limit																
Blocksworld-b	180	54	54	54	52	55	56	57	54	54	<b>59</b>	54	54	56	56	57
Boxworld-b	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Drive-b	90	89	<b>90</b>	<b>90</b>	73	<b>90</b>	<b>90</b>	<b>90</b>	<b>90</b>	<b>90</b>	<b>90</b>	<b>90</b>	80	<b>90</b>	<b>90</b>	<b>90</b>
Elevators-b	90	77	74	72	53	71	71	68	<b>79</b>	75	76	76	52	75	73	65
ExplBlocksw-b	150	62	71	77	83	85	84	84	62	90	101	86	88	84	<b>104</b>	97
Random-b	72	48	48	50	50	<b>51</b>	<b>51</b>	<b>51</b>	37	40	44	44	44	45	45	45
RectTireworld-b	36	<b>18</b>	12	<b>18</b>	12	12	12	12	<b>18</b>	13	<b>18</b>	<b>18</b>	13	12	12	12
Schedule-b	138	59	60	60	59	60	60	60	62	60	62	60	60	<b>70</b>	65	<b>70</b>
SearchRescue-b	90	59	63	73	71	77	75	75	70	69	83	76	76	<b>84</b>	83	<b>84</b>
Tireworld-b	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90
TriTireworld-b	60	41	42	51	51	52	51	51	49	47	57	<b>59</b>	54	57	54	57
Zenotravel-b	78	17	19	31	18	31	30	31	40	34	<b>42</b>	<b>42</b>	18	40	37	<b>42</b>
$\Sigma$ IPPC-b	1092	614	623	666	612	674	670	669	651	662	<b>722</b>	695	629	703	709	709
Canadian RCP Benchmarks with Compiled Budget Limit																
NoMystery-b	60	9	10	36	21	35	29	36	19	16	51	43	19	46	35	<b>52</b>
Rovers-b	60	46	48	52	46	55	52	52	50	49	57	51	46	<b>60</b>	57	59
TPP-b	60	17	18	32	14	36	27	35	24	23	45	40	13	42	39	<b>48</b>
$\Sigma$ RCP-b	180	72	76	120	81	126	108	123	93	88	153	134	78	148	131	<b>159</b>
Pentesting Benchmarks																
Pentest-b	90	57	62	67	63	68	67	67	61	63	68	<b>77</b>	63	69	68	68
Pentest	15	9	<b>10</b>	9	<b>10</b>	<b>10</b>	<b>10</b>	9	9	9	9	<b>10</b>	9	9	9	9
$\Sigma$ Pentest	105	66	72	76	73	78	77	76	70	72	77	<b>87</b>	72	78	77	77

Table 18.1.: MaxProb coverage (number of instances solved within time and memory limits) on cyclic (upper part) and acyclic benchmarks (lower part). Best results in **bold**. Abbreviations: “exhDFS” exhaustive anytime DFS; “LILAO\*” variant of ILAO\* replacing the VI termination check by maintaining solved labels, using TALILAO\* on the cyclic benchmarks; “w/o learning” baselines with dead-end learning disabled: “ $u^0$ ” using trivial goal-probability value initialization and no pruning; “ $u^1$ ” dead-end pruning via  $u^1$ ; “ $u^{pdb}$ ” dead-end pruning via the unsolvability PDB heuristic of AIDOS; “PPDB” orthogonal goal-probability PDB heuristic over the systematic patterns of size up to 3; “w/ learning” dead-end learning enabled, using: “ $u^C$ ” with neighbors refinement; “ $u$ -trap” combining dead-end pruning via  $u$  by  $u$ -trap dead-end learning, “ $u^0$ -trap” without additional unsolvability detector.

Consider now dead-end learning. On the cyclic part, the impact is again limited. For exhDFS, the learning configurations are (almost) indistinguishable from the non-learning ones. The same holds the  $\mathcal{U}$ -trap learning configurations of LILAO\*;  $\mathcal{U}^C$  learning turns out detrimental overall, with considerable coverage losses in ExplodingBlocks. The picture becomes clearer on the acyclic benchmark set. The relative performance between the different learning methods is the same for exhDFS and LILAO\*. Though, the impact of dead-end learning tends to be larger on exhDFS than on LILAO\*; we discuss the reasons below.

Learning again particularly excels on the budget-limited RCP domains.  $\mathcal{U}$ -trap learning can build upon its performance on the classical planning benchmark versions. For LILAO\*, all three  $\mathcal{U}$ -trap learning variants improve coverage across the budget-limited RCP set compared to the corresponding  $\mathcal{U}$ -based dead-end pruning baselines. The same holds for exhDFS and  $\mathcal{U}^0$ -trap and  $\mathcal{U}^1$ -trap learning, for  $\mathcal{U}^{\text{pdb}}$ -trap learning coverage is improved in TPP. The improvements are generally largest for  $\mathcal{U}^0$ -trap and  $\mathcal{U}^1$ -trap learning, unsurprisingly so, given the much stronger performance of  $\mathcal{U}^{\text{pdb}}$  pruning in the first place. Note though that the road-graph uncertainty makes the Canadian variants more complex than in the original classical planning equivalents: whether the available budget (resource) in a state is sufficient to accomplish the goal varies depending on the roads that are (un)available in that state. This appears to cause substantial problems to  $\mathcal{U}^C$  learning. In stark contrast to the performance on the classical planning domain variants,  $\mathcal{U}^C$  is mostly detrimental here. While the  $\mathcal{U}^C$  learning configurations can solve slightly more instances of NoMystery than the  $\mathcal{U}^1$ -pruning baselines, coverage decreases in Rovers and TPP, in the latter even substantially.

On the IPPC domains, the results are no longer in clear favor of dead-end learning.  $\mathcal{U}$ -trap learning still offers a strong performance in many domains, but it is also detrimental in some domains. Starting from the weaker baselines,  $\mathcal{U}^0$ -trap and  $\mathcal{U}^1$ -trap learning are able to improve coverage, relative to the baseline, in almost every domain.  $\mathcal{U}^{\text{pdb}}$ -trap learning performs not quite as well compared to the  $\mathcal{U}^{\text{pdb}}$  pruning only. Coverage improvements in up to 5 out of the 12 domains are contrasted by coverage losses in up to 4 domains, the latter dominating the former in total. However, overall, all three  $\mathcal{U}$ -trap learning variants are still competitive with  $\mathcal{U}^{\text{pdb}}$  pruning, the strongest configuration here. Comparing the three  $\mathcal{U}$ -trap configurations to each other, while they all perform equally overall, each one has its own advantages.  $\mathcal{U}^1$ -trap learning is especially effective on ExplodingBlocks-b, yet in the other domains  $\mathcal{U}^1$  rarely pays off compared to the naive  $\mathcal{U}^0$ -trap learning variant.  $\mathcal{U}^{\text{pdb}}$ -trap learning inherits the per-domain best performance of its  $\mathcal{U}^{\text{pdb}}$  dead-end pruning and  $\mathcal{U}^0$ -trap learning components. Finally,  $\mathcal{U}^C$  learning is again not really competitive. It improves over  $\mathcal{U}^1$  dead-end pruning in budget-limited Random, SearchAndRescue, and TriangleTireworld for both exhDFS and LILAO\*, and in ExplodingBlocks for exhDFS. At the same time, however, it leads to significant coverage drops on Drive, Elevators, and, for LILAO\*, on Zenotravel. The latter outweighs the benefits from the former.

### 18.5.3. Search Reduction

Figure 18.2 complements the coverage data from Table 18.1 by per-instance comparisons between the LILAO\* configurations with and without learning, showing results for search space size, i.e., number of states visited until termination, and total runtime. We show results for  $\mathcal{U}^C$ ,  $\mathcal{U}^0$ -trap, and  $\mathcal{U}^{\text{pdb}}$ -trap learning. The results for  $\mathcal{U}^1$ -trap learning are similar to that for  $\mathcal{U}^0$ -trap learning.

Consider first  $\mathcal{U}^C$ , Figures 18.2a and 18.2b. As evident from Figure 18.2a, there is clearly no lack of potential for conflict-driven learning in MDP heuristic search. The search space with  $\mathcal{U}^C$  learning is often orders of magnitude smaller than without learning (using just  $\mathcal{U}^1$  for dead-end pruning). This shows that (1) LILAO\* is well capable of identifying sufficiently many conflicts for the  $\mathcal{U}^C$  refinements, and that

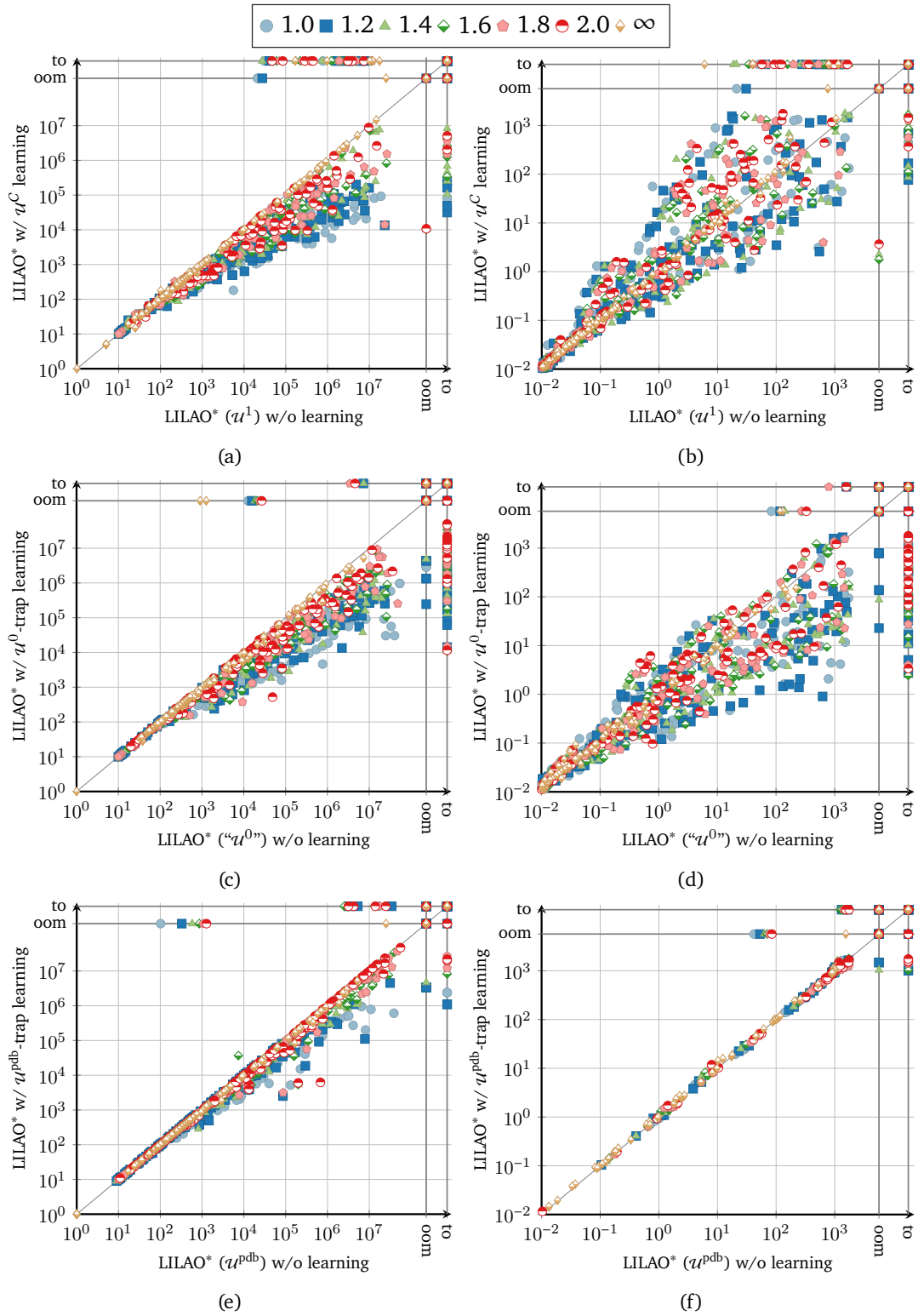


Figure 18.2.: Per-instance comparison of LILAO\* (TALILAO\* on cyclic benchmarks) without ( $x$ -axes) versus with ( $y$ -axes) dead-end learning for different conflict refinement methods. From top to bottom:  $\mathcal{U}^C$  with neighbors refinement;  $\mathcal{U}^0$ -trap learning; and  $\mathcal{U}^{\text{pdb}}$ -trap learning. Left: search space size, i.e., number of states  $|\hat{S}|$  visited until termination. Right: runtime in CPU seconds. Instances are distinguished along the constrainedness factor  $\mathfrak{C}$ , where for  $\mathfrak{C} = \infty$ , the budget is not limited. “oom” out of memory; “to” out of time.

(2) the  $\mathcal{U}^C$  refinements generalize to other states at sufficient scale, leading to additional pruning in the remainder of the search.  $\mathcal{U}^C$  learning can however not leverage this advantage due to the  $\mathcal{U}^C$  refinement and computation overhead. As shown by Figure 18.2b, despite the often much smaller search space, the runtime of  $\mathcal{U}^C$  learning is frequently even up to one order of magnitude higher than without learning. The search space size profile of  $\mathcal{U}^0$ -trap learning, Figure 18.2c, is similar to that of  $\mathcal{U}^C$  learning. But contrary to  $\mathcal{U}^C$ ,  $\mathcal{U}^0$ -trap learning is less heavy on runtime. As shown in Figure 18.2d, the search space reductions here typically translate directly into runtime reductions, which explains the coverage improvements in Table 18.1. Consider finally  $\mathcal{U}^{\text{pdb}}$ -trap learning, Figures 18.2e and 18.2f. The much stronger dead-end detection capabilities of the  $\mathcal{U}^{\text{pdb}}$  unsolvability detector leaves less potential for dead-end learning. While search space size is still reduced frequently, the scale of which is much smaller than for the other two learning variants. Search space reductions of up to 2 orders of magnitude are still possible. The effect on runtime is however often overshadowed by the PDB construction time.

To close the discussion of Figure 18.2, consider the different constrainedness levels  $\mathfrak{C}$ . The entry “ $\mathfrak{C} = \infty$ ” represents the budget-unlimited (including the cyclic) benchmarks. The plots show a clear trend: the more constrained the budget, the higher is the impact of dead-end learning. The reason simply is that smaller values of  $\mathfrak{C}$  lead to more dead ends, which makes conflict identification easier, and creates more pruning potential so that the learning effort can pay off. However, in line with the coverage results, the plots also show that dead-end learning has almost no effect on the budget-unlimited domains.

#### 18.5.4. Performance Analysis

Table 18.2 provides additional data supporting the analysis of the performance of conflict-driven learning. In order for it to work well, there must be three prerequisites satisfied: (1) conflicts must be identified, (2) the conflict refinements have to generalize to dead ends other than the identified conflicts, and (3) the scale of that generalization must outweigh the overhead associated with the conflict learning. The table grasps these prerequisites for each of the three conflict learning methods from Figure 18.2 in terms of (1) “# Id” the number instances on which at least one conflict was identified, so as the mean number of identified conflict components; (2) “Red S” the mean search space size reduction factor over the “Id” instances relative to the configuration without learning; and (3) “Red t” the mean runtime reduction factor, the gap between (2) and (3) roughly corresponding to the conflict learning overhead.

Consider the budget-unlimited benchmarks. Clearly, (1) is the main bottleneck for the shown dead-end learning configurations.  $\mathcal{U}^C$  and  $\mathcal{U}^{\text{pdb}}$ -trap learning do not even identify a single conflict in many domains. In the remaining domains only comparatively few conflicts are identified, not sufficient for the refinements to generalize. The same holds for  $\mathcal{U}^0$ -trap learning, with the exception of ExplodingBlocks, where conflicts are identified, and generalization does happen, but  $\mathcal{U}^0$ -trap learning is not effective as per (3).

The situation changes on the budget-limited benchmarks. For  $\mathcal{U}^C$  and  $\mathcal{U}^0$ -trap learning, (1) is no longer an issue; conflicts are identified in almost all instances of all domains. The amount of possible conflicts obviously relates to the quality of the unsolvability detector. This is reflected on the  $\mathcal{U}^{\text{pdb}}$ -trap learning data. In domains where  $\mathcal{U}^{\text{pdb}}$  already has a high dead-end detection accuracy, naturally, no or only few conflicts can be found. This particularly applies to the \*Tireworld-b domains, and to a smaller extent to Blocksworld-b.

Considering the data for (2) and (3), generalization happens in almost all domains where conflicts are identified. The most notable exceptions are RectangleTireworld-b and Tireworld-b for  $\mathcal{U}^C$  and  $\mathcal{U}^0$ -trap learning, Zenotravel-b and Pentest-b for  $\mathcal{U}^{\text{pdb}}$ -trap learning, and Drive-b for  $\mathcal{U}^C$  learning. Yet, despite strong gener-

		$\mathcal{U}^C$ learning Id & common $\mathcal{U}^1$						LILAO* $\mathcal{U}^0$ -trap learning Id & common $\mathcal{U}^0$						$\mathcal{U}^{\text{pdb}}$ -trap learning Id & common $\mathcal{U}^{\text{pdb}}$					
Domain	#	Cov	# Id	Confl	Red S	Red t	Cov	# Id	Confl	Red S	Red t	Cov	# Id	Confl	Red S	Red t			
Cyclic IPPC Benchmarks (with Dead Ends)																			
Drive	15	15	0				15	15	3.6	1.1	0.9	15	9	1.7	1.1	1.0			
ExplBlocksw	30	23	11	4.9	1.0	0.8	10	9	913.2	1.7	0.7	19	11	264.8	1.3	1.0			
RectTireworld	14	14	12	1.8	1.1	1.0	12	12	6.4	1.0	0.9	14	0						
Schedule	30	10	0				6	2	3.0	1.3	1.6	10	0						
SearchRescue	15	5	0				5	5	2.0	1.2	1.3	5	2	1.0	1.0	1.0			
Tireworld	15	15	0				15	14	9.9	1.1	0.8	15	0						
Canadian RCP Benchmarks without Budget Limit																			
NoMystery	10	5	0				5	5	7.0	1.0	0.9	5	0						
Rovers	10	10	0				10	9	2.1	1.0	0.9	10	0						
TPP	10	10	0				10	10	5.7	1.2	1.2	10	0						
Acyclic IPPC Benchmarks																			
TriTireworld	10	10	0				10	6	2.0	1.0	0.5	10	0						
IPPC Benchmarks with Compiled Budget Limit																			
Blocksworld-b	180	54	54	211.2	6.0	0.2	56	56	2.2k	2.7	0.5	57	4	10.9k	14.2	1.0			
Boxworld-b	18	0	0				0	0				0	0						
Drive-b	90	80	80	90.1	1.2	0.2	90	90	83.4	1.8	0.6	90	60	124.4	1.7	1.0			
Elevators-b	90	52	52	248.9	3.8	0.1	75	75	3.8k	3.1	0.9	65	65	1.2k	1.8	1.0			
ExplBlocksw-b	150	88	84	20.2	5.3	1.8	84	84	210.4	12.5	3.1	97	72	152.9	2.0	1.0			
Random-b	72	44	29	5.8	12.5	3.4	45	45	3.9	3.2	2.8	45	26	2.0	3.1	1.0			
RectTireworld-b	36	13	12	16.8	1.2	1.0	12	12	22.0	1.2	1.4	12	0						
Schedule-b	138	60	60	7.0	1.8	1.1	70	70	20.0	2.3	1.1	70	43	32.8	1.6	1.0			
SearchRescue-b	90	76	76	29.2	7.1	3.4	84	84	170.3	8.9	6.5	84	60	20.7	1.3	1.0			
Tireworld-b	90	90	73	4.5	1.1	0.9	90	90	13.6	1.0	0.7	90	0						
TriTireworld-b	60	54	54	67.3	7.0	3.0	57	57	268.8	5.2	1.7	57	0						
Zenotravel-b	78	18	18	435.8	6.8	0.2	40	40	37.0k	4.3	1.2	42	26	1.4k	1.0	1.0			
Canadian RCP Benchmarks with Compiled Budget Limit																			
NoMystery-b	60	19	19	569.6	90.0	1.9	46	46	11.1k	38.5	18.7	52	33	693.1	1.0	1.0			
Rovers-b	60	46	46	312.2	4.3	0.5	60	60	1.1k	5.9	9.5	59	59	391.6	1.6	1.0			
TPP-b	60	13	13	867.2	16.0	0.3	42	42	30.7k	7.1	4.4	48	48	4.7k	1.8	1.0			
Pentesting Benchmarks																			
Pentest-b	90	63	62	16.9	1.8	1.1	69	69	35.3	3.0	4.5	68	68	12.3	1.1	1.0			
Pentest	15	9	0				9	9	6.2	1.1	1.0	9	2	2.4	1.0	1.0			

Table 18.2.: Analysis of different conflict refinement methods in LILAO\* (TALILAO\* on cyclic benchmarks). Configurations abbreviated as in Table 18.1. “Cov” repeats MaxProb coverage for convenience; “# Id” shows the number of solved instances in which at least one conflict was identified; “Id & common” considers of the Id instances only those also solved by the corresponding baseline pruning configuration without learning (using the unsolvability detector as indicated in the table); “Confl” geometric mean of the number of conflict components identified, “k” multiples of 1000; “Red S” and “Red t” geometric mean search space size respectively runtime reduction factors with learning relative to without.

alization, conflict learning may still not be effective. The discrepancy between the theoretically possible performance improvement “Red S” (ignoring the overhead) and the actual improvement “Red t” is particularly high for  $\mathcal{U}^C$  learning. As per the reported data,  $\mathcal{U}^C$  learning is effective only in ExplodingBlocks-b,

		$\mathcal{U}^0$ -trap learning										
		Coverage improvements					Search space reduction					
Domain	#	VI	exhDFS	AO*	LILAO*	LRTDP	# Com.	VI	exhDFS	AO*	LILAO*	LRTDP
Cyclic IPPC Benchmarks (with Dead Ends)												
Drive	15	15 +0 -0	15 +0 -0		15 +0 -0	15 +0 -0	15	1.0	1.0		<b>1.1</b>	<b>1.1</b>
ExplBlocksw	30	9 +0 -0	11 +0 -0		10 +0 -0	10 +0 -0	9	2.7	<b>2.9</b>		1.6	2.3
RectTireworld	14	14 +0 -2	14 +0 -2		14 +0 -2	14 +0 -2	12	1.0	1.0		1.0	1.0
Schedule	30	7 +0 -1	7 +0 -1		6 +0 -0	6 +0 -0	6	<b>1.2</b>	1.0		1.1	1.1
SearchRescue	15	6 +0 -0	5 + <b>1</b> -0		5 +0 -0	5 +0 -0	5	1.1	1.1		<b>1.2</b>	<b>1.2</b>
Tireworld	15	14 +0 -0	15 +0 -0		15 +0 -0	15 +0 -0	14	1.0	1.0		1.1	<b>1.5</b>
$\Sigma$ IPPC	119	65 +0 -3	67 +1 -3		65 +0 -2	65 +0 -2						
Canadian RCP Benchmarks without Budget Limit												
NoMystery	10	5 +0 -0	5 +0 -0		5 +0 -0	5 +0 -0	5	1.0	1.0		1.0	1.0
Rovers	10	10 +0 -0	10 +0 -0		10 +0 -0	10 +0 -0	10	1.0	1.0		1.0	1.0
TPP	10	8 +0 -0	10 +0 -0		10 +0 -0	10 +0 -0	8	1.0	1.1		1.2	<b>1.3</b>
$\Sigma$ RCP	30	23 +0 -0	25 +0 -0		25 +0 -0	25 +0 -0						
Acyclic IPPC Benchmarks (with Dead Ends)												
TriTireworld	10	5 +0 -0	7 +0 -0	10 +0 -0	10 +0 -0	10 +0 -0	5	1.0	1.0	1.0	1.0	1.0
IPPC Benchmarks with Compiled Budget Limit												
Blocksworld-b	180	54 +1 -0	54 +1 -0	54 +0 -0	54 + <b>2</b> -0	54 + <b>2</b> -0	54	<b>3.5</b>	<b>3.5</b>	2.4	2.7	2.7
Boxworld-b	18	0 +0 -0	0 +0 -0	0 +0 -0	0 +0 -0	0 +0 -0						
Drive-b	90	90 +0 -0	89 + <b>1</b> -0	90 +0 -0	90 +0 -0	90 +0 -0	89	2.3	2.3	<b>2.5</b>	1.8	2.3
Elevators-b	90	72 +0 -6	77 +0 -6	85 +0-13	79 +0 -4	82 +0-11	66	2.5	3.0	3.0	3.0	<b>3.2</b>
ExplBlocksw-b	150	51+ <b>26</b> -0	62+23 -0	61+21 -0	62+22 -0	65+20 -0	51	<b>18.6</b>	10.5	8.4	7.9	8.3
Random-b	72	31 +7 -0	48 +3 -0	36 + <b>9</b> -0	37 +8 -0	37 +8 -0	31	<b>2.6</b>	1.7	1.9	2.1	2.1
RectTireworld-b	36	18 +0 -6	18 +0 -6	18 +0 -6	18 +0 -6	18 +0 -6	12	<b>7.7</b>	<b>7.7</b>	3.0	1.2	1.2
Schedule-b	138	52 +5 -0	59 +1 -0	60 +7 -0	62 + <b>8</b> -0	62 + <b>8</b> -0	52	<b>2.4</b>	1.8	1.9	1.7	1.6
SearchRescue-b	90	59+ <b>18</b> -0	59+ <b>18</b> -0	71+11 -0	70+14 -0	67+16 -0	59	8.8	8.8	6.7	7.3	<b>9.6</b>
Tireworld-b	90	90 +0 -0	90 +0 -0	90 +0 -0	90 +0 -0	90 +0 -0	90	<b>1.1</b>	<b>1.1</b>	1.0	1.0	<b>1.1</b>
TriTireworld-b	60	42+10 -0	41+ <b>11</b> -0	49 +7 -0	49 +8 -0	49 +8 -0	41	6.9	<b>7.1</b>	5.5	3.6	5.1
Zenotravel-b	78	18+13 -0	17+ <b>14</b> -0	42 +0 -4	40 +0 -0	35 +4 -0	17	<b>9.5</b>	<b>9.5</b>	6.9	4.4	4.9
$\Sigma$ IPPC-b	1092	577+ <b>80</b> -12	614+72-12	656+55-23	651+62-10	649+66-17						
Canadian RCP Benchmarks with Compiled Budget Limit												
NoMystery-b	60	9+26 -0	9+26 -0	21+24 -0	19+ <b>27</b> -0	20+ <b>27</b> -0	9	93.2	<b>94.7</b>	39.6	26.3	51.4
Rovers-b	60	46 +7 -0	46 +9 -0	55 +5 -0	50+ <b>10</b> -0	51 +8 -0	46	7.9	<b>9.9</b>	4.8	5.7	9.2
TPP-b	60	17+18 -0	17+ <b>19</b> -0	34+11 -1	24+18 -0	26+18 -0	17	15.2	<b>15.7</b>	4.9	6.2	10.6
$\Sigma$ RCP-b	180	72+51 -0	72+54 -0	110+40 -1	93+ <b>55</b> -0	97+53 -0						
Pentesting Benchmarks												
Pentest-b	90	58+10 -0	57+11 -0	56+ <b>12</b> -0	61 +8 -0	57 +9 -0	56	<b>4.0</b>	<b>4.0</b>	3.6	2.9	2.8
Pentest	15	10 +0 -0	9 + <b>1</b> -0	8 + <b>1</b> -0	9 +0 -0	9 +0 -1	8	1.1	1.1	1.1	1.1	1.1
$\Sigma$ Pentest	105	68+10 -0	66+12 -0	64+ <b>13</b> -0	70 +8 -0	66 +9 -1						

Table 18.3.: Comparing the impact of  $\mathcal{U}^0$ -trap learning on different search algorithms. On the cyclic benchmarks, AO\* cannot be applied, for LILAO\* and LRTDP we fall back to TALILAO\* and TALRTDP. Left: coverage improvement data for  $\mathcal{U}^0$ -trap learning relative to the  $\mathcal{U}^0$  baseline without learning, showing from left to right: total number of instances solved by the  $\mathcal{U}^0$  baseline; “+” number of instances solved with learning but not without; “-” number of instances not solved with learning but solved without. Right: geometric mean search space size reduction factors over commonly solved instances. “# Com.” shows the number of those instances. Largest improvements in the two categories respectively in **bold**.

Random-b, SearchAndRescue-b, TriangleTireworld-b, and NoMystery-b. With the exception of Exploding-Blocks-b, these are exactly the domains in which  $\mathcal{U}^C$  learning achieves higher coverage than  $\mathcal{U}^1$  pruning without learning.

The difference between “Red  $S$ ” and “Red  $t$ ” is generally is smaller for  $\mathcal{U}^0$ -trap learning, although in some cases the overhead can still be prohibitive. Notice that in some cases runtime reduction actually exceeds search space reduction, most notably Rovers-b and Pentest-b. This is so because reevaluating the unsolvability detector during the policy-graph traversals can reduce state reexpansions, and therewith reduce search effort without actually reducing the search space size.

Finally, note that despite generalization is happening to a notable extent on multiple domains,  $\mathcal{U}^{pdb}$ -trap learning has no effect on performance measure “Red  $t$ ”. This is so because the PDB construction usually takes a large fraction of the total runtime, which is the same with and without learning. Ignoring the construction, the overhead of the  $\mathcal{U}^{pdb}$ -trap refinements actually outweighs the search space reduction in almost all cases. The main strength of  $\mathcal{U}^{pdb}$ -trap learning lies in the combination of the performance of its two base components:  $\mathcal{U}^{pdb}$  pruning, and  $\mathcal{U}^0$ -trap learning. Contrary to the results in classical planning, synergistic effects between  $\mathcal{U}^{pdb}$  dead-end pruning and trap learning, i.e., instances solved by  $\mathcal{U}^{pdb}$ -trap learning, yet not by either of its components, happened only rarely: 1 instance of NoMystery-b, and 2 instances of TPP-b.

### 18.5.5. Search Algorithm Comparison

Table 18.3 compares the extent to which the different search algorithms can benefit from conflict-driven learning. We chose  $\mathcal{U}^0$ -trap learning as comparison basis because (a) it is competitive with the static dead-end detectors, and (b) dead-end learning is most essential for the performance. The table shows MaxProb coverage improvement data over the non-learning configurations as a total measure of performance gains. Moreover, it measures the effectiveness of the different search algorithms in utilizing the conflict-driven learning approach in terms of search space size reduction factors of with learning relative to without. The quicker a search algorithm identifies conflicts, and the more efficient it uses the gained information for pruning, the larger the search space reduction factor will be.

Consider the budget-unlimited benchmarks. As we have just seen, conflict-driven learning has generally only little effect as conflicts are not being identified at sufficient scale. While the previous analysis focused on MDP heuristic search as per LILAO\* specifically, Table 18.3 indicates that this is an artifact of the benchmarks, and does not stem from the inability of heuristic search to identify conflicts per se. In particular, recall that upon termination, VI will have identified all reachable conflicts. Yet, considerable search space reductions can be observed only in ExplodingBlocks. And this is indeed also the only domain on which conflicts are identified to a notable extent. On the other budget-unlimited domains, conflict identification in VI behaves similarly to the  $\mathcal{U}^0$ -trap learning results in LILAO\* from Table 18.2. Given the undersupply of conflicts in all algorithms, the impact of  $\mathcal{U}^0$ -trap learning on them is basically the same.

Differences become more pronounced on the budget-limited benchmarks. VI’s and exhDFS’s exhaustive explorations are advantageous in making dead ends quickly become known. This is reflected on coverage improvements to a certain extent. But, it really becomes evident on the search space reduction data, where VI and exhDFS dominate the heuristic search algorithms almost consistently, and partly even substantially. Though, notice that there actually a few cases where the opposite is the case, notably budget-unlimited Tireworld and TPP, and SearchRescue-b. This confirms once again that heuristic search is able utilize dead-end information more effectively; as via the non-trivial goal-probability value initialization, sub-optimal

regions of the state space can be identified, translating into the pruning of even non-dead-end states.

Comparing the heuristic search algorithms to each other, they perform equally well. LILAO\* tends to be least effective in identifying conflicts overall, though. Its tip-state cutoffs during the policy-graph traversals introduce a breadth-bias into the exploration, which is generally detrimental for conflict identification. While AO\* also cuts off its traversals once reaching a tip state, its explorations differ from LILAO\* in running down just a single policy execution trace versus exploring the entire policy-graph up to the current tip states. As such, AO\* is less affected from the breadth bias induced by the cutoffs. LRTDP has much stronger depth bias. Its trials go deep, and the subsequent calls to the `CheckAndMarkSolved` sub-procedure yield the necessary exploration for dead ends to become known. This is reflected on the data from Table 18.3.

## 18.6. Discussion

We briefly summarize the contents of this chapter, and discuss possible future works pertaining specifically to conflict-driven learning in the probabilistic context.

### 18.6.1. Summary and Related Work

We have shown how to learn to recognize dead ends during MDP state-space search. Regarding dead-end detection and conflict refinement, thanks to the all-outcomes determinization, the techniques developed in Part II apply unchanged. For conflict identification, the procedures from Chapter 5 work almost out of the box. We have equipped a range of different MDP search algorithms with these procedures, and discussed how to leverage the learned information. The conflict identification setup in all algorithms was proved to be sound and complete, i.e., to fulfill the prerequisites of the conflict refinements, and to utilize as much of the search's knowledge as possible for dead-end learning.

Our experiments demonstrated that conflict-driven learning can be an effective means for goal-probability analysis. Even when using nothing but the learned dead-end information for pruning, the performance is on par with that of state-of-the-art static unsolvability detectors. However, not all tested configurations worked equally well.  $\mathcal{U}^C$  learning almost always caused a prohibitive overhead, which was compensated only rarely, despite that the learned information generalized to a remarkable extent.  $\mathcal{U}$ -trap learning has been overall much more effective. Yet, like in our experiments on classical planning, the impact of learning significantly varies between different domains. Substantial improvements were generally achieved only if dead ends are plenty, causing sufficiently many conflicts to become known quickly, and providing enough pruning potential for the conflict refinements to pay off. This property is naturally provided by the budget-limited benchmarks, but not so much in the remaining domains.

Contrasting their counterparts in classical planning, conflict-driven learning has shown to work quite effectively in MDP heuristic search. Although the relative gains were notably smaller compared to VI and the anytime DFS variant that are geared for this kind of learning, significant performance improvements could still be observed on many domains. Part of the reason for this is certainly the goal-probability objective. Maximizing goal probability entails minimizing the likelihood of entering dead end states. So, their identification naturally happens as a byproduct of finding the MaxProb policy. Another distinguishing property is the strong depth-bias in their exploration, which fosters conflict identification, and therewith increases the potential of pruning.

Related work has already been extensively discussed in Section 9.2, but we want to nevertheless comment on one work that is particularly closely related: Kolobov et al.'s (2010b) work on SIXTHSENSE. Like we do,

SIXTHSENSE learns NoGoods, i.e., efficiently testable dead-end conditions, during MDP heuristic search. It differs from our approach in two fundamental points: (1) the invocation of NoGood generation; and (2) the fact that its NoGood learning procedure is inherently incomplete. More specifically, regarding (1), SIXTHSENSE starts the generation of new NoGoods at predetermined intervals during search instead of based on a systematic conflict analysis and discovery as we do here. Regarding (2), SIXTHSENSE validates the generated NoGoods against GRAPHPLAN's (Blum and Furst, 1997) underlying  $h^2$  (Haslum and Geffner, 2000) reachability approximation, and therewith cannot learn to recognize any dead end not recognized by  $h^2$ . In contrast, our learning techniques are able to represent every dead end.

### 18.6.2. Future Work

An interesting avenue for future work remains the application of our techniques to other MDP reachability problems. This particularly pertains to quantitative model-checking tasks. These could principally be handled already via available compilations into probabilistic planning (Klauck et al., 2020). The more interesting question in the long term, however, is obtaining variants of our techniques that integrate natively into existing model-checking tools. Beyond goal-probability analysis, our experiments also left open an evaluation on expected-cost objectives, for which dead-end detection, while less, can still be of vital importance. Notice that all our search and learning modifications work on these objectives out of the box.

In the context of expected-cost minimization on SSPs, it could be furthermore worthwhile to investigate the extension of “conflicts” to states that are unable to reach the goal with absolute certainty. As this condition is weaker than the dead-end requirement, it allows for more pruning. Pruning such states however still remains safe because they must not be visited by any optimal policy according to the SSP requirements.

An obvious next step in the context of goal-probability analysis is lifting the conflict-driven learning approach to learning *quantitative NoGoods*, i.e., given some  $0 < p < 1$ , can we learn to identify states that cannot reach the goal with more than  $p$  probability? Such information can be exploited directly in heuristic search as means to initialize the goal-probability value function: if a  $p$ -NoGood covers a state  $s$ , then clearly  $V^U(s) = p \geq V^*(s)$  is valid goal-probability upper bound. The learned information can be helpful even beyond initialization, as a newly learned NoGood may generalize to other visited states where  $V^U(s) > p$ . Spinning the idea further, such a learning procedure would yield the building blocks for generating *goal-probability certificates*, i.e., (hopefully) compact self-contained proofs of a state not being able to reach the goal with a probability higher than the provided threshold. On the one hand, this can serve as a proof that the returned policy is indeed optimal. On the other hand, such certificates can be of particular interest in a quantitative model checking context, delivering the arguments for why the system under investigation remains safe almost surely.

More generally speaking, quantitative NoGood learning can also be interpreted as learning and refining a goal-probability upper bound  $H$  online, during search;  $H$  providing the reasoning of why state  $s$  cannot reach the goal with probability higher than  $p = H(s)$ . Online heuristic refinement has already been used successfully in classical planning (e.g., Fickert and Hoffmann, 2017; Eifler and Fickert, 2018; Seipp, 2021). Translating the methodology of conflict-driven dead-end learning, conflicts becomes states  $s$  visited by search such that  $H(s) > V^*(s)$ . This then re-raises the questions of (1) how to identify conflicts, and (2) given a conflict  $s$ , how find a refinement  $H'$  of  $H$  such that  $H(s) > H'(s)$ ; while hopefully also  $H(t) > H'(t)$  for states  $t \neq s$ . An obvious characterization for (1) is given by the Bellman operator, i.e., a state is a conflict if  $V^U(s) > (BV^U)(s)$ . But how to test this efficiently, and possibly on all currently visited states? (2) is more problematic given the lack of goal-probability heuristics to even start from. The projection based heuristics as in Chapter 14, or as in probabilistic PDB heuristics (Klößner et al., 2021),

are not very well suited due to supporting only coarse refinements. A promising aspirant might again be given by the  $\Pi^C$  compilation (Haslum, 2016), with its support of representing selective fact conjunctions  $C$ . As we have seen in Chapter 7, by applying the classical-planning state-equation heuristic to  $\Pi^C$ , one can render the heuristic perfect in the limit. Can this result be generalized to the goal-probability state-equation heuristic from Chapter 14? Answering this question entails lifting  $\Pi^C$  to probabilistic planning tasks in the first place.

**Part V.**

**Conclusion**



# 19. Conclusion

The ability to analyze conflicts, and to learn from them avoiding similar mistakes in the future, is a fundamental algorithm ingredient in constraint satisfaction. But despite this success, it has received only little attention in state-space search so far. Transferred to state-space search, conflicts take the form of dead-end states, i.e., states starting from which it is not possible to reach the target states. Although not quite as quintessential as in constraint satisfaction, the ability to deal with such conflicts effectively remains important in many state-space search applications, such as planning with limited resources, game playing, model checking of safety properties, and for decision-making in uncertain environments.

In this work, we have demonstrated how to learn sound and generalizable knowledge from conflicts during state-space search in the context of classical and probabilistic planning.

In Part II, we equipped state-space search in classical planning with the ability to learn from conflicts. At the heart of our technique are unsolvability detectors, sound and efficient to test dead-end criteria. The overall principle is simple. Whenever search encounters a dead-end state  $s$  that was not refuted, a conflict, we compute a reason for why  $s$  is a dead end, and use this to refine the unsolvability detector so that it recognizes  $s$  afterwards. This refinement has the potential to generalize to unseen states, namely all dead ends for which the same reason applies. To implement this principle, we developed methods for identifying conflicts during search based on the information that search is unveiling as part of its exploration. For dead-end refutation and refinement, we considered three families of unsolvability detectors: critical-path heuristics, state-equation and potential heuristics, as well as dead-end traps. We showed that they can converge to the perfect unsolvability detector, recognizing all dead ends, in the limit, and designed suitable conflict-based refinement algorithms. Wrapped in a depth-first search, the resulting technique reaches the elegance of conflict-driven learning in constraint satisfaction, including the ability to immediately backjump to the shallowest non-refuted state after a conflict was found.

For probabilistic planning, we focused on goal-probability maximization in MDPs, where dead ends take an especially important role. State-space search methods for goal-probability planning were however severely underexplored. In preparation of applying conflict-driven learning in this context, Part III filled that gap by (i) designing and exploring a large space of MDP state-space search methods, systematizing known algorithms and contributing several new algorithm variants, (ii) introducing simpler, but still practically relevant, special cases, and (iii) providing a comprehensive empirical analysis, which among other things demonstrated significant benefits of our new algorithm variants.

In Part IV, we concluded our journey by lifting the concepts introduced in Part II to the MDP search methods from Part III. Thanks to the fact that action-outcome uncertainty is irrelevant with respect to qualitative reachability questions, for dead-end refutation and refinement, the techniques from before apply unchanged. We showed the missing component, conflict identification, to be possible via only small modifications to our previous methods, accounting for the different structure of MDP versus graph search.

Our experimental evaluation in Parts II and IV confirmed that conflict-driven learning can significantly reduce search effort for (a) finding plans in classical planning in the presence of dead ends, (b) proving

classical planning tasks unsolvable, and (c) for goal-probability maximization in MDP planning. But, this requires certain properties to be present, with the result that exact impact varies substantially depending on the domain. Viewed overall, our techniques do not, as it stands, deliver an empirical breakthrough. Nevertheless, we expect our work to be the beginning of the story, not its end, and lots remains to be explored in future work.

Regarding learning, there are open questions pertaining to different refinement algorithms, to the integration of other dead-end detection methods, as well as to combined techniques that utilize the complementary strengths of different unsolvability detectors. There are open questions concerning other search techniques, such as symmetry or partial-order reduction. Besides investigating how to synergistically combine them with learning in search, these could be leveraged within the refinements for stronger generalization, e.g., by learning to recognize also dead ends where symmetric reasons apply.

A particular promising direction is the extension of the approach to broader forms of conflicts. In classical planning, one could learn to identify states that may reach the goal, but only if going back to some ancestor state. Such states are equally wasteful to explore than dead ends, yet presumably appear more frequently in search, and exist even in domains without dead ends. Challenges lie in the development of according unsolvability detector equivalents, and suitable refinement methods. In probabilistic planning, exciting questions pertain to taking the step from dead-end learning to learning goal-probability estimates.

Last but not least, there remains the exploration of other sorts of state-space search applications, where dead-end reasoning is equally or even more important than in the planning variants considered here. This includes, in particular, FOND planning, game playing, and qualitative and quantitative model checking.

## **Appendices**



# A. Supplemental Results

## A.1. Offline $\mathcal{U}$ -Trap Coverage

		Baseline	Offline $\mathcal{U}$ -trap													
		$\mathcal{U}^0$	$\mathcal{U}^0$		$\mathcal{U}^1$		$\mathcal{U}^2$		MSa		PDB		SEQ		PoT	
Domain	#		1	2	1	2	1	2	1	2	1	2	1	2	1	2
Unsolvability IPC (UIPC) 2016 Benchmarks																
BagBarman	20	<b>12</b>	<b>12</b>	<b>12</b>	<b>12</b>	<b>12</b>	4	0	0	0	<b>12</b>	<b>12</b>	<b>12</b>	<b>12</b>	<b>12</b>	<b>12</b>
BagGripper	25	<b>6</b>	5	1	3	0	0	0	3	2	3	1	3	0	0	0
BagTransport	29	<b>7</b>	<b>7</b>	<b>7</b>	<b>7</b>	<b>7</b>	<b>7</b>	6	0	0	<b>7</b>	<b>7</b>	<b>7</b>	<b>7</b>	<b>7</b>	<b>7</b>
Bottleneck	25	10	10	<b>15</b>	10	<b>15</b>	12	12	3	3	10	<b>15</b>	10	<b>15</b>	10	<b>15</b>
CaveDiving	25	7	7	7	7	7	7	7	2	2	7	7	7	<b>8</b>	7	7
ChessBoard	23	5	5	5	5	5	5	5	1	1	5	5	5	5	5	<b>13</b>
Diagnosis	11	4	4	<b>5</b>	4	<b>5</b>	4	3	2	1	4	<b>5</b>	4	<b>5</b>	4	<b>5</b>
DocTransfer	20	5	6	11	<b>12</b>	9	7	2	5	5	6	11	6	7	6	5
NoMystery	20	2	2	2	2	2	2	2	<b>8</b>	7	2	2	2	2	2	2
PegSol	24	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	0	0	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>
PegSolRow5	15	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	3	3	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>
Rovers	20	7	7	7	7	7	7	7	<b>9</b>	<b>9</b>	7	7	7	7	7	7
SlidingTiles	20	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Tetris	20	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	5	0	5	5	<b>10</b>	<b>10</b>	<b>10</b>	<b>10</b>	5	5
TPP	30	17	17	17	17	17	17	16	<b>21</b>	<b>21</b>	17	18	17	17	17	16
$\Sigma$ UIPC	327	131	131	138	135	135	116	99	72	69	129	<b>139</b>	129	134	121	133
Unsolvable Resource-Constrained Planning (RCP) Benchmarks																
NoMystery	150	26	26	26	26	42	26	44	<b>128</b>	<b>128</b>	26	28	26	26	26	33
Rovers	150	3	3	3	3	4	3	4	<b>96</b>	92	3	3	3	3	3	3
TPP	25	5	5	8	5	8	5	0	<b>9</b>	8	5	8	5	7	5	0
$\Sigma$ RCP	325	34	34	37	34	54	34	48	<b>233</b>	228	34	39	34	36	34	36
$\Sigma$ Total	652	165	165	175	169	189	150	147	<b>305</b>	297	163	178	163	170	155	169

Table A.1.: Coverage results for search with dead-end detection by offline  $\mathcal{U}$ -traps *only*, for conjunction candidates of size  $k \in \{1, 2\}$ , and different  $\mathcal{U}$ . Best results in **bold**.

		$\mathcal{U}^1$			$\mathcal{U}^2$			MSa			PDB			SEQ			Pot		
Domain	#	–	$\Gamma_1$	$\Gamma_2$	–	$\Gamma_1$	$\Gamma_2$	–	$\Gamma_1$	$\Gamma_2$	–	$\Gamma_1$	$\Gamma_2$	–	$\Gamma_1$	$\Gamma_2$	–	$\Gamma_1$	$\Gamma_2$
Unsolvability IPC (UIPC) 2016 Benchmarks																			
BagBarman	20	8	8	8	0	0	0	4	0	0	<b>12</b>	<b>12</b>	<b>12</b>	4	4	4	4	4	4
BagGripper	25	3	1	0	0	0	0	3	3	2	3	3	1	<b>23</b>	14	0	3	0	0
BagTransport	29	6	6	6	16	16	7	6	0	0	7	7	7	<b>29</b>	<b>29</b>	19	24	24	19
Bottleneck	25	20	20	21	21	21	12	10	3	3	19	19	19	<b>25</b>	<b>25</b>	<b>25</b>	<b>25</b>	<b>25</b>	22
CaveDiving	25	7	7	7	6	6	6	7	2	2	7	7	7	8	8	<b>9</b>	8	8	7
ChessBoard	23	5	5	5	4	4	4	5	1	1	5	5	5	<b>23</b>	<b>23</b>	<b>23</b>	<b>23</b>	<b>23</b>	<b>23</b>
Diagnosis	11	<b>6</b>	<b>6</b>	5	5	5	4	4	2	1	5	5	5	4	4	4	4	4	4
DocTransfer	20	7	<b>12</b>	9	7	7	2	10	5	5	<b>12</b>	<b>12</b>	11	6	6	7	7	7	5
NoMystery	20	2	2	2	2	2	2	8	8	7	<b>11</b>	<b>11</b>	<b>11</b>	2	2	2	5	5	2
PegSol	24	<b>24</b>	<b>24</b>	<b>24</b>	21	22	22	<b>24</b>	0	0	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	22	22	22
PegSolRow5	15	5	5	5	4	4	4	5	3	3	5	5	5	<b>15</b>	<b>15</b>	<b>15</b>	<b>15</b>	<b>15</b>	<b>15</b>
Rovers	20	7	7	7	7	7	7	9	9	9	<b>12</b>	<b>12</b>	<b>12</b>	6	6	6	6	6	6
SlidingTiles	20	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Tetris	20	5	5	5	5	5	0	5	5	5	10	10	10	<b>20</b>	<b>20</b>	15	<b>20</b>	15	5
TPP	30	17	17	17	15	15	15	<b>24</b>	21	21	<b>24</b>	<b>24</b>	<b>24</b>	14	14	14	19	19	17
$\Sigma$ UIPC	327	132	135	131	123	124	95	134	72	69	166	166	163	<b>213</b>	204	177	195	187	161
Unsolvable Resource-Constrained Planning (RCP) Benchmarks																			
NoMystery	150	52	52	52	83	84	63	130	128	128	<b>149</b>	<b>149</b>	<b>149</b>	16	16	16	68	68	54
Rovers	150	7	7	8	67	67	67	<b>111</b>	96	92	93	93	93	1	1	3	1	1	3
TPP	25	7	7	9	8	8	0	17	9	8	<b>20</b>	<b>20</b>	<b>20</b>	1	1	2	11	11	0
$\Sigma$ RCP	325	66	66	69	158	159	130	258	233	228	<b>262</b>	<b>262</b>	<b>262</b>	18	18	21	80	80	57
$\Sigma$ Total	652	198	201	200	281	283	225	392	305	297	<b>428</b>	<b>428</b>	425	231	222	198	275	267	218

Table A.2.: Coverage results for search using  $\mathcal{U}$  (“–”) versus  $\mathcal{U} + \mathcal{U}^\Gamma$  for dead-end detection, for offline constructed  $\mathcal{U}$ -traps  $\Gamma$  with  $k \in \{1, 2\}$  (“ $\Gamma_k$ ”). Best results in **bold**.

## A.2. MaxProb Coverage for DFHS Variants

		DFHS  <sub>U</sub> w/ PDB pruning													
		VI								LABEL					
		NoUp		BW		FW				BW		FW			
Domain	#	NONE	TIP	NONE	TIP	NONE	TIP	INC	TIP∪INC	NONE	TIP	NONE	TIP	INC	TIP∪INC
IPPC Benchmarks															
TriTire	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
IPPC Benchmarks with Budget Limit															
Blocksw-b	180	100	99	99	100	99	100	92	98	109	112	109	<b>113</b>	102	102
Boxw-b	18	<b>3</b>	<b>3</b>	<b>3</b>	0	<b>3</b>	<b>3</b>	0	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>
Drive-b	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90
Elevators-b	90	85	83	84	81	87	84	81	82	<b>90</b>	86	<b>90</b>	87	83	82
ExpBloc-b	150	98	97	99	95	99	98	97	98	100	<b>101</b>	99	<b>101</b>	96	100
Random-b	72	53	53	53	53	53	53	53	53	53	53	53	53	53	53
RecTire-b	36	36	36	36	36	36	36	36	36	36	36	36	36	36	36
Schedule-b	138	60	60	60	60	60	60	60	60	<b>62</b>	<b>62</b>	<b>62</b>	<b>62</b>	<b>62</b>	<b>62</b>
SeaResc-b	90	83	83	83	83	83	83	83	83	<b>84</b>	<b>84</b>	<b>84</b>	<b>84</b>	<b>84</b>	<b>84</b>
Tirew-b	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90
TriTire-b	60	57	54	57	56	57	56	56	55	57	<b>59</b>	57	<b>59</b>	<b>59</b>	<b>59</b>
Zenotra-b	78	<b>42</b>	40	<b>42</b>	<b>42</b>	<b>42</b>	<b>42</b>	39	39	<b>42</b>	<b>42</b>	<b>42</b>	<b>42</b>	<b>42</b>	<b>42</b>
Σ IPPC-b	1092	797	788	796	786	799	795	777	787	816	818	815	<b>820</b>	800	803
Canadian RCP Benchmarks with Budget Limit															
NoMyst-b	60	51	49	52	49	52	49	52	49	<b>57</b>	<b>57</b>	57	57	57	57
Rovers-b	60	58	58	58	57	58	58	58	58	59	<b>60</b>	<b>60</b>	<b>60</b>	59	<b>60</b>
TPP-b	60	48	47	50	46	54	48	51	47	<b>55</b>	54	<b>55</b>	54	<b>55</b>	54
Σ RCP-b	180	157	154	160	152	164	155	161	154	171	171	<b>172</b>	171	171	171
Pentesting Benchmarks															
Pentest-b	90	68	68	68	68	68	68	63	62	<b>70</b>	69	<b>70</b>	69	68	67
Pentest	15	<b>9</b>	<b>9</b>	<b>9</b>	<b>9</b>	<b>9</b>	<b>9</b>	8	8	<b>9</b>	<b>9</b>	<b>9</b>	<b>9</b>	8	8
Σ Pentest	105	77	77	77	77	77	77	71	70	<b>79</b>	78	<b>79</b>	78	76	75

Table A.3.: Acyclic planning. MaxProb coverage for different DFHS configurations. Best results in **bold**. DFHS parameters are abbreviated as “VI”: termination is checked via VI; “LABEL”: labeling is enabled; “NoUp” no value updates during the policy-graph exploration; “BW”: value updates only on the way back up of the exploration; “FW”: additionally doing value updates on the way down of exploration; “NONE” no cutoffs, exploration is terminated only at terminal and goal states; “TIP”: cutting off exploration at tip states (CUTOFFTIP), i.e., states that have not been expanded yet; “INC” cutting off exploration at inconsistent states (CUTOFFINCONSISTENT); and “TIP ∪ INC” using both CUTOFFTIP and CUTOFFINCONSISTENT. Recall that ILAO\* corresponds to entry VI/BW/TIP; HDP corresponds to LABEL/FW/INC. All configurations use PDB pruning. Default tie-breaking strategy.

Domain	#	FRET- $\pi$ DFHS <sub>U</sub> w/ $h^{\text{FF}}$ pruning													
		VI								LABEL					
		NoUp		BW		FW				BW		FW			
		NONE	TIP	NONE	TIP	NONE	TIP	INC	TIP $\cup$ INC	NONE	TIP	NONE	TIP	INC	TIP $\cup$ INC
IPPC Benchmarks															
Blocksw	30	<b>22</b>	<b>22</b>	<b>22</b>	16	<b>22</b>	16	<b>22</b>	16	<b>22</b>	16	<b>22</b>	16	16	<b>22</b>
Boxw	15	<b>7</b>	<b>7</b>	<b>7</b>	5	<b>7</b>	5	<b>7</b>	5	<b>7</b>	5	<b>7</b>	5	5	<b>7</b>
Drive	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
Elevators	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
ExpBloc	30	24	24	25	26	25	26	26	26	25	27	25	27	27	<b>28</b>
Random	15	<b>14</b>	<b>14</b>	<b>14</b>	13	<b>14</b>	13	<b>14</b>	13	<b>14</b>	<b>14</b>	<b>14</b>	<b>14</b>	<b>14</b>	<b>14</b>
RecTire	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14
Schedule	30	10	10	10	10	10	10	10	10	10	10	10	10	10	10
SeaResc	15	5	5	5	5	5	5	5	5	5	5	5	5	5	<b>6</b>
Tirew	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
Zenotra	30	<b>10</b>	<b>10</b>	<b>10</b>	8	<b>10</b>	8	<b>10</b>	8	<b>10</b>	8	<b>10</b>	8	8	<b>10</b>
$\Sigma$ IPPC	224	151	151	152	142	152	142	153	142	152	144	152	144	144	<b>156</b>
Canadian RCP Benchmarks															
NoMyst	10	<b>5</b>	<b>5</b>	<b>5</b>	4	<b>5</b>	4	<b>5</b>	4	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>
Rovers	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
TPP	10	8	8	8	8	8	8	8	8	8	8	8	8	8	<b>9</b>
$\Sigma$ RCP	30	23	23	23	22	23	22	23	22	23	23	23	23	23	<b>24</b>

Table A.4.: Cyclic planning. MaxProb coverage for different DFHS configurations. Results for FRET- $\pi$  only. Best results in **bold**. DFHS parameters are abbreviated as “VI”: termination is checked via VI; “LABEL”: labeling is enabled; “NoUp” no value updates during the policy-graph exploration; “BW”: value updates only on the way back up of the exploration; “FW”: additionally doing value updates on the way down of exploration; “NONE” no cutoffs, exploration is terminated only at terminal and goal states; “TIP”: cutting off exploration at tip states (CUTOFFTIP), i.e., states that have not been expanded yet; “INC” cutting off exploration at inconsistent states (CUTOFFINCONSISTENT); and “TIP $\cup$ INC” using both CUTOFFTIP and CUTOFFINCONSISTENT. Recall that ILAO\* corresponds to entry VI/BW/TIP; HDP corresponds to LABEL/FW/INC. All configurations use  $h^{\text{FF}}$  pruning. Default tie-breaking strategy.

## B. Proofs of Part II

### B.1. Conflict Identification in Forward State-Space Search

#### B.1.1. Correctness of the Known-Dead-End Labeling Procedure (Theorem 5.1)

**Theorem 5.1.** *At the start of the **while** loop in Algorithm 5.1, the labeled states are exactly the known dead ends.*

*Proof.* Soundness, i.e.,  $t$  labeled  $\Rightarrow t$  is a known dead end, follows immediately from construction because at the moment a state  $t$  is labeled we have  $\mathcal{R}[\hat{\Theta}](t) \subseteq \text{closed}$ , and once that condition is true obviously it remains true for the remainder of the search.

Completeness, i.e.,  $t$  is a known dead end  $\Rightarrow t$  labeled, can be proved by induction on the number of expansions. Assume that the claim holds before a state  $s$  is expanded; we need to show that, for any states  $t$  that were not known dead ends before but are known dead ends now,  $t$  will be labeled. Call such  $t$  *new-label states*. Clearly, any new-label state must be an ancestor of  $s$ . Therefore, a new-label state can exist only if, after the expansion,  $\mathcal{R}[\hat{\Theta}](s) \subseteq \text{closed}$ : else, an open state is reachable from  $s$ , and transitively is reachable from all ancestors of  $s$ . In the case where  $\mathcal{R}[\hat{\Theta}](s) \subseteq \text{closed}$ ,  $s$  is labeled and so is every new-label state parent  $t$  of  $s$ , due to the recursive invocation on  $t$ . It remains to show that each new-label state  $t$  will indeed be reached, and thus labeled, during the reverse traversal of the search space induced by the recursive invocations of **CheckAndLearn**. This is a direct consequence of the following two observations: (1) each ancestor  $t$  of  $s$  has not been labeled dead end so far, and (2) for each new-label state  $t$ , the search space contains a path of new-label states  $t = t_0, t_1, \dots, t_n = s$ . The first observation follows immediately from the algorithm: since  $t$  is an ancestor of  $s$ ,  $t$  must have been expanded at some point (which means that  $t$  could not have been labeled dead end before its expansion), and  $t$  could not have been labeled dead end during any previous call to **CheckAndLearn** because at least one open state was reachable from  $t$  during any such call. The second observation follows immediately from  $\mathcal{R}[\hat{\Theta}](t) \subseteq \text{closed}$ : (as above)  $t$  is an ancestor of  $s$ , i.e., the search space contains a path  $t = t_0, t_1, \dots, t_n = s$ , and due to the transitivity of reachability, for every  $1 \leq i < n$ ,  $\mathcal{R}[\hat{\Theta}](t_i) \subseteq \text{closed}$ . Obviously, for every  $1 \leq i \leq n$ ,  $s$  is also reachable from  $t_i$ , meaning that  $t_i$  must be a new-label state, too. Since **CheckAndLearn** will traverse at least one such path from  $t$  to  $s$  in reverse order,  $t$  will indeed be labeled eventually.

□

### B.2. Critical-Path Heuristics: Conflict Refinement & NoGood Learning

#### B.2.1. Correctness of Path-Cut Refinement Algorithm (Theorem 6.1)

**Theorem 6.1.** *Suppose  $c(a) = 1$  for all  $a \in \mathcal{A}$ . Let  $C$  be any set of atomic conjunctions. Let  $s$  be a state with  $h^C(s) < h^*(s)$ . Then:*

- (i) The execution of  $\text{PathCutRefine}(\mathcal{G}, h^C(s))$  terminates eventually, and is well defined, i.e., (a) in any call  $\text{PathCutRefine}(P, N)$  there exists  $c \in C$  so that  $c \subseteq P$  and  $h^C(s, c) \geq N$ ; and (b) if  $N = 0$ , then  $P \not\subseteq s$ .
- (ii) If  $X$  is the set of conjunctions resulting from  $\text{PathCutRefine}(\mathcal{G}, h^C(s))$ , then  $h^{C \cup X}(s) > h^C(s)$ .

*Proof.*

- (i) (a) follows directly from Equation (6.1). Initially, there must be some  $c \in C$  so that  $c \subseteq \mathcal{G}$  and  $h^C(s, c) = h^C(s, \mathcal{G})$  (last case of Equation (6.1)). Consider a recursive call  $\text{PathCutRefine}(P, n)$ . Let  $P', n'$  be the arguments of the call to  $\text{PathCutRefine}$  that caused the recursion, let  $a$  be the corresponding action, and let  $c' \subseteq P'$  be the conjunction selected in  $\text{PathCutRefine}(P', n')$ . Due to the selection of  $c'$ , we have  $h^C(s, c') = h^C(s, P') = n' = n + 1$ ; and because  $c' \subseteq P'$ , we also have  $\text{regress}(c', a) \subseteq \text{regress}(P', a) = P$ . Hence, by definition of  $h^C$ ,  $h^C(s, P) \geq n$ , i.e., there is a conjunction  $c \in C$ ,  $c \subseteq P$  so that  $h^C(s, c) \geq n$ .

For (b), assume for contradiction that there is a call  $\text{PathCutRefine}(P, 0)$  where  $P \subseteq s$ . Let  $a_n, \dots, a_1$  be the actions that label the recursion path down to the call  $\text{PathCutRefine}(P, 0)$ . It is easy to show by induction that  $\langle a_1, \dots, a_n \rangle$  is actually a plan for  $s$ . However,  $n$  is exactly  $h^C(s)$ , which means that  $h^C(s) = h^*(s)$ , a contradiction to the assumption.

- (ii) We show for every call  $\text{PathCutRefine}(P, n)$  and for the constructed conflict  $x$  that  $h^{C \cup X}(s, x) > n$  when  $\text{PathCutRefine}(P, n)$  terminates. In other words, when  $\text{PathCutRefine}(\mathcal{G}, h^C(s))$  terminates, then we have  $h^{C \cup X}(s) > h^C(s)$ , as desired. The proof is by induction on  $n$ . For  $n = 0$ , the conflict  $x \subseteq P$  is chosen such that  $x \not\subseteq s$ . Hence,  $h^{C \cup X}(s, x) > 0 = n$  due to Equation (6.1). For the induction step,  $n > 0$ , let  $x$  be the conflict that is constructed in the call  $\text{PathCutRefine}(P, n)$ . Since  $n > 0$ , there must be an atomic conjunction  $c \in C$  that is part of  $x$ ,  $c \subseteq x$ , and so that  $h^C(s, c) \geq n$ . If  $h^C(s, c) > n$ , then clearly  $h^C(s, x) > n$  and the claim follows. So, assume that  $h^C(s, c) = n$ , and let  $a \in \mathcal{A}[x]$  be an arbitrary achiever of  $x$  (i.e.,  $\text{regress}(x, a) \neq \perp$ ). In case  $\mathcal{A}[x]$  is empty, it directly follows that  $h^{C \cup X}(s, x) = \infty > n$  (Equation (6.1)). Otherwise, distinguish between the cases  $a \in \mathcal{A}[c]$  and  $a \notin \mathcal{A}[c]$ . If  $a \notin \mathcal{A}[c]$ , then, since  $c \subseteq x$  and  $x \cap \text{del}(a) = \emptyset$ , i.e.  $c \cap \text{del}(a) = \emptyset$ , we have that  $\text{add}(a) \cap c = \emptyset$ . Therefore,  $c \subseteq \text{regress}(x, a)$  and thus  $h^{C \cup X}(s, \text{regress}(x, a)) \geq n$ . On the other hand, if  $a \in \mathcal{A}[c]$ , then we must have recursed on  $P' = \text{regress}(P, a)$  and  $n' = n - 1$ . If  $x'$  is the conflict constructed in this call, then we know by induction hypothesis that  $h^{C \cup X}(s, x') > n'$ . Because of the selection of  $x$ , we ensured that  $x' \subseteq \text{regress}(x, a)$ , and as a consequence  $h^{C \cup X}(s, \text{regress}(x, a)) > n' = n - 1$ . Since  $a$  was chosen arbitrarily, this shows that  $h^{C \cup X}(s, x) > n$ .

□

### B.2.2. Proof that Size-Minimal $\mathcal{U}^C$ Neighbor-Conflict Extraction is NP-HARD

Consider the  $\text{Extract}(P)$  procedure from Algorithm 6.2. Let  $x \subseteq P$  be given such that the properties (b)  $x$  is unreachable from the dead-end states  $S$ ; and (c) for all neighbors  $t \in T$ ,  $\mathcal{U}^C(t, x) = \infty$ , are satisfied. We show that deciding whether  $|x|$  is minimal among all conjunctions  $x' \subseteq P$  satisfying those properties is NP-HARD in general.

We do so via a reduction from the minimal vertex cover problem. That problem asks for given graph  $G = (V, E)$  with vertices  $V$  and edges  $E$  for a size-minimal subset  $C \subseteq V$  such that for all edges  $(u, v) \in E$ ,

either  $u \in C$  or  $v \in C$ .

We construct a planning task  $\Pi^G$  as follows. Let  $\mathcal{F} = V \cup \{p\}$ , where  $g$  does not appear in  $V$ . Let  $\mathcal{G} = \mathcal{F}$ , and  $\mathcal{A} = \{a^{(u,v)} \mid (u,v) \in E\}$ , with  $\text{pre}(a^{(u,v)}) = \emptyset$ ,  $\text{add}(a^{(u,v)}) = \{g\}$ , and  $\text{del}(a^{(u,v)}) = \{u, v\}$ . The construction of  $\Pi^G$  is obviously polynomial in the size of  $G$ . Consider  $s = V$ . Note that  $S = \{s\}$  satisfies the  $\mathcal{U}^1$  recognized-neighbors property, because every action removes some “vertex facts”. Further, note that for every successor  $s \llbracket a^{(u,v)} \rrbracket = \mathcal{F} \setminus \{u, v\}$ , i.e., the facts not reachable by  $s \llbracket a^{(u,v)} \rrbracket$  under  $\mathcal{U}^1$  are exactly  $u$  and  $v$ . Let  $x \subseteq \mathcal{G}$  be a size-minimal conjunction that satisfies (b) and (c). From (c) and the previous observation, it follows  $C = x \setminus \{g\}$  is a minimal vertex cover. This shows the claim.  $\square$

### B.2.3. Worst-Case $\mathcal{U}^C$ Refinement Example (Proposition 6.2)

**Proposition 6.2.** *There are planning tasks  $\Pi$ , and dead-end states  $s \in \mathcal{S}^\Pi$  such that  $\mathcal{U}^C(s) = \infty$  entails that  $C$  contains exponentially many conjunctions in  $|\Pi|$ , even if restricting the states  $s$  to ones that satisfy the  $\mathcal{U}^{C'}$  recognized-neighbor property for some polynomially bounded  $C'$ .*

*Proof.* Consider the family of STRIPS planning tasks  $\Pi_n = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  with

- Facts  $\mathcal{F} = \{p_i \mid 1 \leq i \leq n\} \cup \{q_i \mid 1 \leq i \leq n\} \cup \{r\}$ .
- Actions  $\mathcal{A} = \{a_i \mid 1 \leq i \leq n\} \cup \{b_i \mid 1 \leq i \leq n\}$ , where
 
$$\begin{array}{lll} \text{pre}(a_i) = \{p_i\} & \text{add}(a_i) = \{r\} & \text{del}(a_i) = \{p_i\} \\ \text{pre}(b_i) = \{p_i, r\} & \text{add}(b_i) = \{q_i\} & \text{del}(b_i) = \{p_i\} \end{array}$$
- Initial state  $\mathcal{I} = \{p_i \mid 1 \leq i \leq n\}$ .
- Goal  $\mathcal{G} = \{q_i \mid 1 \leq i \leq n\}$ .

Each  $\Pi_n$ ,  $n \geq 1$ , is obviously unsolvable. Note that the initial state  $\mathcal{I}$  satisfies the recognized-neighbors property for  $\mathcal{U}^1$ : the actions applicable in  $\mathcal{I}$  are  $a_1, \dots, a_n$ ; and applying any  $a_i$  in  $\mathcal{I}$  removes  $p_i$ , which cannot be added, and hence  $\mathcal{U}^1(\mathcal{I} \llbracket a_i \rrbracket, \{q_i\}) = \mathcal{U}^1(\mathcal{I} \llbracket a_i \rrbracket, \text{pre}(b_i)) = \infty$ , i.e.,  $\mathcal{U}^1(\mathcal{I} \llbracket a_i \rrbracket, \mathcal{G}) = \infty$ .

However, in order to obtain  $\mathcal{U}^C(\mathcal{I}) = \mathcal{U}^C(\mathcal{I}, \mathcal{G}) = \infty$ ,  $C$  has to contain exponentially many conjunctions. Observe that  $\mathcal{U}^C(\mathcal{I}, \mathcal{G}) = \infty$  entails that  $\mathcal{U}^C(\mathcal{I}, P) = \infty$  for every  $P = \{r\} \cup \{X_i \mid 1 \leq i \leq n\}$ , where  $X_i \in \{p_i, q_i\}$ , and  $p_k \in P$  for at least one  $1 \leq k \leq n$ . Namely, suppose there was such  $P$  with  $\mathcal{U}^C(\mathcal{I}, P) < \infty$ . As the exact selection  $X_i$  is not important, assume w.l.o.g. that  $P = \{r, q_1, \dots, q_k, p_{k+1}, \dots, p_n\}$  for  $k < n$ . Observe that

$$\begin{aligned} \mathcal{U}^C(\mathcal{I}, \mathcal{G}) &\leq \mathcal{U}^C(\mathcal{I}, \text{regress}(\mathcal{G}, b_n)) \\ &\leq \mathcal{U}^C(\mathcal{I}, \text{regress}(\text{regress}(\mathcal{G}, b_n), b_{n-1})) \\ &\dots \\ &\leq \mathcal{U}^C(\mathcal{I}, \text{regress}(\text{regress}(\dots, b_{k+2}), b_{k+1})) \\ &= \mathcal{U}^C(\mathcal{I}, P) \\ &< \infty \end{aligned}$$

which contradicts the assumption  $\mathcal{U}^C(\mathcal{I}, \mathcal{G}) = \infty$ .

Finally, note that  $h^*(I, P') < \infty$  for every  $P' \subset P$  such that  $X_i \notin P'$  from some  $1 \leq i \leq n$ . Therefore,  $u^C(I, P) = \infty$  can only be true if  $C$  contains a conjunction  $c_P$  such that  $\{X_1, \dots, X_n\} \subseteq c_P$ . The number of such conjunctions is exponential in  $n$ , concluding the proof.  $\square$

### B.3. LP Heuristics Over Conjunctions: Compilation, Convergence & Conflict Refinement

#### B.3.1. $\Pi^C$ Dominates Partial Variable Merges (Theorem 7.1)

**Theorem 7.1.** *For every  $\Pi$  in TNF, every set of conjunctions  $C$ , and every state  $s$ , it holds that*

$$h^{\text{seq}}[\Pi^C](s) \geq h^{C\text{seq}}(s)$$

*Proof.* Let  $\text{SEQ}[\Pi^C]$  be the LP underlying  $h^{\text{seq}}[\Pi^C](s)$ , and  $C\text{SEQ}$  that underlying  $h^{C\text{seq}}(s)$ . If there is no feasible solution to  $\text{SEQ}[\Pi^C]$ , then  $h^{\text{seq}}[\Pi^C](s) = \infty$ , and  $h^{\text{seq}}[\Pi^C](s) \geq h^{C\text{seq}}(s)$  follows trivially.

Suppose that  $X$  is a feasible solution to  $\text{SEQ}[\Pi^C]$ , where  $X_{a^C}$  denotes the count-value for the action occurrence  $a^C \in \mathcal{A}^C$ . We show how to construct from  $X$  a feasible solution  $Y$  for  $C\text{SEQ}$  with equal objective value. We choose the values of  $Y_a$  and  $Y_a^{x \rightarrow x'}$  in the following way. For every action  $a \in \mathcal{A}$ , we set

$$Y_a := \sum_{C \subseteq C: a^C \in \mathcal{A}^C} X_{a^C} \quad (\text{T7.1.1})$$

(the sum over all occurrences  $a^C$  of  $a$ ). Let  $V$  be any non-unit set of variables glanced by the conjunctions  $C$ , and consider the corresponding partial-variable-merge state space  $\Theta_V^C$ . The transition variables are defined as follows:

- For every transition  $c \xrightarrow{a} c'$ :

$$Y_a^{c \rightarrow c'} := \sum_{\substack{C \subseteq C: a^C \in \mathcal{A}^C, \\ \pi_c \mapsto \perp \in \text{eff}(a^C), \\ \pi_{c'} \mapsto \top \in \text{eff}(a^C)}} X_{a^C} \quad (\text{T7.1.2})$$

- For every transition  $s_V \xrightarrow{a} c$ :

$$Y_a^{s_V \rightarrow c} := \sum_{\substack{C \subseteq C: a^C \in \mathcal{A}^C, \\ \pi_c \mapsto \top \in \text{eff}(a^C)}} X_{a^C} \quad (\text{T7.1.3})$$

- For every transition  $c \xrightarrow{a} s_V$ :

$$Y_a^{c \rightarrow s_V} := \sum_{\substack{C \subseteq C: a^C \in \mathcal{A}^C, \\ \pi_c \mapsto \top \in \text{pre}(a^C), \\ \pi_{c'} \mapsto \perp \in \text{eff}(a^C)}} X_{a^C} \quad (\text{T7.1.4})$$

Notice that every action occurrence  $a^C \in \mathcal{A}^C$  counts towards at most one transition in every partial variable merge. Assume the contrary, and let  $a^C \in \mathcal{A}^C$  be an action occurrence, which is counted in two distinct, state-changing transitions  $x_1 \xrightarrow{a} x_2$  and  $x_3 \xrightarrow{a} x_4$ . Note that  $x_1 \neq s_V$  or  $x_2 \neq s_V$ , and  $x_3 \neq s_V$  or  $x_4 \neq s_V$ . We distinguish between the following cases

- $x_2 = c_1$  and  $x_4 = c_2$  for  $c_1, c_2 \in C$  with  $\text{vars}(c_1) = \text{vars}(c_2) = V$  and  $c_1 \neq c_2$ . From (T7.1.2) and (T7.1.3) and the definition of  $a^C$ , it follows that  $c_1, c_2 \in C$  what contradicts the compatibility requirement of  $C$ .
- $x_1 = c_1$  and  $x_3 = c_2$  for  $c_1, c_2 \in C$  with  $\text{vars}(c_1) = \text{vars}(c_2) = V$  and  $c_1 \neq c_2$ . We show that  $c_1 \subseteq \text{regress}(C, a)$  and  $c_2 \subseteq \text{regress}(C, a)$ , which contradicts the compatibility requirement of  $C$ . As per the definition (T7.1.4),  $\pi_{c_i} \mapsto \top \in \text{pre}(a^C)$  if  $x_{i+1} = s_V$ , for both  $i \in \{1, 3\}$ , which by the definition of  $a^C$  implies  $c_i \subseteq \text{regress}(C, a)$ , as desired. Consider (T7.1.2), and suppose  $x_{i+1} = c'$ . Since  $\Pi$  is in TNF and  $c' \in C$ , as per  $\pi_{c'} \mapsto \top \in \text{eff}(a^C)$ , it follows that  $V \subseteq \text{vars}(\text{regress}(C, a))$ . By the definition of  $a^C$ ,  $\pi_{c_i} \mapsto \perp \in \text{eff}(a^C)$  implies that  $c_i \parallel \text{regress}(C, a)$ . With  $\text{vars}(c_i) = V \subseteq \text{vars}(\text{regress}(C, a))$ , hence  $c_i \subseteq \text{regress}(C, a)$ .
- $x_2 = c_1$  and  $x_3 = c_2$  for  $c_1, c_2 \in C$  with  $\text{vars}(c_1) = \text{vars}(c_2) = V$  where either  $x_1 \neq c_2$  or  $x_4 \neq c_1$ . From (T7.1.2), (T7.1.3), and (T7.1.4), it follows that  $\pi_{c_2} \mapsto \perp \in \text{eff}(a^C)$  and  $\pi_{c_1} \mapsto \top \in \text{eff}(a^C)$ . By definition of  $a^C$ ,  $\pi_{c_2} \mapsto \perp \in \text{eff}(a^C)$  implies that  $c_2 \parallel \text{regress}(C, a)$ , and hence  $c_2 \parallel \text{regress}(c_1, a)$ . Therefore, the partial variable merge must contain the transition  $c_2 \xrightarrow{a} c_1$ . Since  $\Pi$  is in TNF,  $a$  can only label a single transition going into  $c_1$ , respectively a single transition going out of  $c_2$ . But this means that  $x_1 = c_2$  and  $x_4 = c_1$ , a contradiction to the assumption.

Since  $x_1 \xrightarrow{a} x_2$  and  $x_3 \xrightarrow{a} x_4$  are two different transitions, one of the three cases must apply. But all of them lead to a contradiction to one of the assumptions. This hence shows that every  $a^C$  is associated with at most one transition of every partial variable merge.

It remains to show that the constructed  $Y$  satisfies the fact state-equation constraints (7.1); the conjunction state-equation constraints, simplified for TNF tasks, Equation (7.3); and the link constraints (7.4).

We start with showing Equation (7.4). Again, suppose that  $V$  is any non-singleton variable set glanced by the conjunctions  $C$ . Let  $a$  be any action. If Equation (7.4) enforces “ $\leq$ ”, the claim follows immediately as per the above arguments (every  $a^C$  is counted by  $Y_a^{x \rightarrow x'}$  for  $\Theta_V^C$  at most once). Suppose Equation (7.4) enforces “ $=$ ”, i.e.,  $\Theta_V^C$  contains exactly one transition  $x \xrightarrow{a} x'$ . Note that for this to be the case, it must hold in particular that  $V \subseteq \text{vars}(\text{pre}(a))$ . Since  $C\text{SEQ}$  ignores self-looping transitions, it must further be  $x \neq x'$ . Distinguish the following three cases:

- $x = c$  and  $x = c'$ , for  $c, c' \in C$ ,  $\text{vars}(c) = \text{vars}(c')$ . It follows that  $c \subseteq \text{pre}(a)$ , i.e.,  $c \subseteq \text{regress}(C, a)$ , for any  $C$ ;  $c \not\parallel \text{eff}(a)$ , i.e.,  $\pi_c \mapsto \perp \in \text{eff}(a^C)$  for all  $a^C$ ; and  $\text{regress}(c', a) = \text{pre}(a)$ , so via the definition of  $a^C$ ,  $c' \in C$ , and thus  $\pi_{c'} \mapsto \top \in \text{eff}(a^C)$ , for all  $a^C$ . In conclusion, via (T7.1.2) and (T7.1.1),  $Y_a^{c \rightarrow c'} = \sum_{C \subseteq C, a^C \in \mathcal{A}^C} X_{a^C} = Y_a$ .
- $x = s_V$  and  $x = c$ ,  $c \in C$ ,  $\text{vars}(c) = V$ . Via  $\text{vars}(c) = V \subseteq \text{vars}(\text{pre}(a))$ , it follows that  $\text{regress}(c, a) = \text{pre}(a)$ . Therefore, by the definition of  $a^C$ ,  $c \in C$  for every action occurrence of  $a$ , i.e.,  $\pi_c \mapsto \top \in \text{eff}(a^C)$ , for every  $a^C$ . In other words,  $Y_a^{s_V \rightarrow c} = Y_a$  via (T7.1.3) and (T7.1.1).
- $x = c$  and  $x = s_V$  for  $c \in C$ ,  $\text{vars}(c) = V$ . With  $V \subseteq \text{vars}(\text{pre}(a))$ , it follows that  $c \subseteq \text{regress}(C, a)$  for all  $C$ . Since  $\Theta_V^C$  contains the transition  $c \xrightarrow{a} s_V$ ,  $a$  must change some variable  $v \in V$ , i.e.,  $\text{eff}(a) \not\parallel c$ . Thus, for all action occurrence  $a^C \in \mathcal{A}^C$  of  $a$ ,  $\pi_c \mapsto \top \in \text{pre}(a^C)$  via the first observation, and  $\pi_c \mapsto \perp \in \text{eff}(a^C)$  as per the second observation. In conclusion,  $Y_a^{c \rightarrow s_V} = Y_a$ , as per (T7.1.4) and (T7.1.1).

Next, consider the state-equation constraint Equation (7.1) for the fact  $p = v \mapsto d$ . Denote by  $\text{PROD}[\Pi^C]$

and  $\text{CONS}[\Pi^C]$  the producers and consumers defined in  $\Pi^C$ . It is easy to see that

$$\text{PROD}[\Pi^C](p) = \{a^C \mid a \in \text{PROD}(p), C \subseteq \mathcal{C} \text{ s.t. } a^C \in \mathcal{A}^C\}$$

and that

$$\text{CONS}[\Pi^C](p) = \{a^C \mid a \in \text{CONS}(p), C \subseteq \mathcal{C} \text{ s.t. } a^C \in \mathcal{A}^C\}$$

As per (T7.1.1), it holds that

$$\sum_{a \in \text{PROD}(p)} Y_a = \sum_{a \in \text{PROD}(p)} \sum_{C \subseteq \mathcal{C}, a^C \in \mathcal{A}^C} X_{a^C} = \sum_{a^C \in \text{PROD}[\Pi^C](p)} X_{a^C}$$

and similarly for the consumers. Since  $X$  satisfies the constraint for  $p$  in  $\text{SEQ}[\Pi^C]$  by assumption,  $Y$  hence satisfies  $p$ 's constraint in  $\text{CSEQ}$ .

It is left to show that  $Y$  satisfies Equation (7.3) for all non-singleton conjunctions  $c \in \mathcal{C}$ . We show that (a) the consumption part for  $c$  in Equation (7.3) is at most as large as the consumption part for  $\pi_c \mapsto \top$  in Equation (7.1), and vice versa that (b) the production part for  $c$  is at least as large as the production part of  $\pi_c \mapsto \top$ . Let  $c \in \mathcal{C}$  be arbitrary, and let  $V = \text{vars}(c)$ .

- (a) The consumption part of  $c$  in Equation (7.3) is  $\sum_{\langle c, a, x \rangle \in \mathcal{T}_V^C, x \neq c} Y_a^{c \rightarrow x}$ . Let  $a$  be any action occurring in this sum, and let  $c \rightarrow x$  be the corresponding transition. This transition must be unique as per the TNF assumptions. Suppose  $x = c' \in \mathcal{C}$ . Due the TNF assumption, it holds for all  $a^C \in \mathcal{A}^C$  with  $\pi_{c'} \mapsto \top \in \text{eff}(a^C)$  that  $\pi_c \mapsto \top \in \text{pre}(a^C)$ . Hence, as per (T7.1.2),  $Y_a^{c \rightarrow c'} = \sum_{C \text{ s.t. } a^C \in \mathcal{A}^C, \pi_c \mapsto \perp \in \text{eff}(a^C), \pi_{c'} \mapsto \top \in \text{eff}(a^C)} X_{a^C} \leq \sum_{C \text{ s.t. } a^C \in \text{CONS}[\Pi^C](\pi_c \mapsto \top)} X_{a^C}$ . Suppose  $x = s_V$ . The construction (T7.1.4) directly yields that " $\leq$ " relation. Since all  $X$  variables are non-negative, the  $\leq$  relation is preserved by the summing over all the actions  $a$ , yielding the desired result.
- (b) For the production part of  $c$  in Equation 7.3, notice that every action occurrence  $a^C$  with  $\pi_c \mapsto \top \in \text{eff}(a^C)$  is counted in at least one transition  $x \xrightarrow{a} c$ . Let  $a^C$  be any such action occurrence. By definition,  $\text{regress}(c, a) \neq \perp$ , and hence  $a$  must induce at least one transition going into  $c$  in the respective variable merge. If  $s_V \xrightarrow{a} c$ , then  $Y_a^{s_V \rightarrow c}$  counts  $X_{a^C}$  by construction (T7.1.3). If  $c' \xrightarrow{a} c$ ,  $c' \neq c$ , then as per the TNF assumption, it must be  $c' \subseteq \text{regress}(c, a)$  and  $c' \not\vdash \text{eff}(a)$ . Hence,  $\pi_{c'} \mapsto \perp \in \text{eff}(a^C)$ . As per (T7.1.2),  $X_{a^C}$  is counted in  $Y_a^{c' \rightarrow c}$ .

Since  $\Delta_c = \Delta_{\pi_c \mapsto \top}$ , and  $X$  satisfies the state-equation constraint for  $\pi_c \mapsto \top$ , we conclude that  $Y$  also satisfies Equation (7.3) for conjunctions  $c \in \mathcal{C}$ .

Finally, note that the construction as per (T7.1.1) guarantees matching objective values. This completes the proof.  $\square$

### B.3.2. Partial Variable Merges May Need Exponentially More Conjunctions Than $\Pi^C$ (Theorem 7.2)

**Theorem 7.2.** *There exists families of  $\Pi$  and  $C$  s.t., to obtain  $h^{C' \text{ seq}}(s) \geq h^{\text{seq}}[\Pi^C](s)$  for all states  $s$ ,  $C'$  must be exponentially larger than  $C$ .*

*Proof.* Consider the following transportation example. The map consists of two locations  $A$  and  $B$ , and is fully connected. There is a single truck  $t$  with load capacity  $l$ , which must bring  $n$  packages  $p_1, \dots, p_n$  to their destinations. To do so, there are three types of actions: to *move* the truck between  $A$  and  $B$ ; to *load* package  $p_i$  into truck at  $B$ , requiring that enough load capacity is available; and to *unload* the package

$p_i$  at location  $A$ . All actions have cost 1. In the initial state,  $t$  is at  $A$ ,  $l$  is 1, and all packages are at  $B$ . The goal is to have all variables  $t, p_1, \dots, p_n$  at  $A$ , and  $l = 1$ . Every optimal plan for this task needs to do one *load*, one *unload*, and two *move* actions for every package, summing up to a total of  $h^*(I) = 4n$ .

In  $h^{\text{seq}}[\Pi^C]$ , considering all conjunctions  $c$  of size  $|c| \leq 3$  makes visible that no two packages can be in the truck at the same time, yielding  $h^{\text{seq}}[\Pi^C](I) = h^*(I)$ . Similar to Example 7.2, every solution to  $\text{SEQ}[\Pi^C]$  must *load* and *unload* every package once. To see that  $h^{\text{seq}}[\Pi^C](I)$  must also account for two *move* actions for each package, consider the action  $a_0 = \text{unload}(p_i, A)$  for any package  $p_i$ . Every action occurrence  $a_0^C$  of  $a_0$  consumes the conjunction  $c_i = \{t \mapsto A, p_i \mapsto T\}$ . The only possibility to produce  $c_i$  is via an action occurrence of the *move* action, moving the truck to  $A$ , and assuming  $p_i \mapsto T$  in its context. Observe that the same *move* action occurrence cannot be used to achieve  $c_i$  and  $c_j$  for two different packages  $i \neq j$ . This is true because every action occurrence  $a_1^C$  of  $a_1 = \text{move}(B, A)$  with  $\{p_i \mapsto T, p_j \mapsto T\} \subseteq C$  consumes the conjunction  $c' = \{t \mapsto B, p_i \mapsto T, p_j \mapsto T\}$ . However,  $c'$  cannot be produced without violating some constraint. There are two possibilities: (1) via an action occurrence of *move*( $A, B$ ), including  $\{p_i \mapsto T, p_j \mapsto T\}$  in the context; and (2) loading one of the packages, e.g., *load*( $p_i, B$ ) including  $\{p_j \mapsto T\}$  in the context. Option (1) cannot be used, as this would basically lead to a cyclic dependency between the respective *move*( $A, B$ ) and *move*( $B, A$ ) action occurrences. Option (2) leads to the consumption of the conjunction  $\{p_j \mapsto T, l \mapsto 1\}$ , which obviously cannot be produced without violating the state equation constraints. Hence, for every package  $p_i$ ,  $c_i$  must be produced through a separate *move*( $B, A$ ) action occurrence. Since every such occurrence consumes  $t \mapsto B$ , its state equation constraint forces to count for one *move*( $A, B$ ) action application per package. This shows that  $h^{\text{seq}}[\Pi^C](I) = h^*(I)$ .

In contrast, in order to obtain  $h^{C\text{seq}}(I) = h^*(I)$ ,  $C$  needs to contain exponentially many conjunctions. Let  $C$  be any set of conjunctions. Let  $\text{COUNT}$  denote any solution to  $C\text{SEQ}$  with minimal objective value. Consider first the state equation constraints in Equation (7.1) over facts  $p = v \mapsto d$ . For  $v = t$ , the state equation constraints are satisfied if the number of *move*( $B, A$ ) action counts matches the number of *move*( $A, B$ ) action counts. For  $v \neq t$ , the *move* count variables do not appear in any constraint. Next, consider any partial variable merge over the variable set  $V$ , and let  $m$  denote the number of packages considered in  $V$ . For  $V = \{v\}$ , the satisfaction of the corresponding constraints in Equation (7.3) and Equation (7.4) are implied by the satisfaction of the state equation constraints, Equation (7.1), for  $v$ . Assume that  $|V| > 1$ . We distinguish between the following cases: For  $m = 0$ , i.e.,  $V = \{t, l\}$ , the abstract initial state in the corresponding partial variable merge is identical to the abstract goal state. No *move* action transitions are required to satisfy the constraints corresponding to  $V$ . For  $m > 0$  but  $t \notin V$ , the corresponding partial variable merge does not contain any *move* transition. For  $m > 0$ ,  $t \in V$ , but  $l \notin V$ , to reach the abstract goal state from the abstract initial state, at most one *move*( $B, A$ ) transition and at most one *move*( $A, B$ ) transition is required to satisfy the constraints. If  $m > 0$  and  $t, l \in V$ , reaching the abstract goal state requires at most  $m$  *move*( $B, A$ ) transitions, and at most  $2m$  *move* transitions in total. Choosing  $\text{COUNT}_{\text{move}(A, B)}$  or  $\text{COUNT}_{\text{move}(B, A)}$  to a value larger than the maximal number of *move*( $A, B$ ) and *move*( $B, A$ ) transition counts over all considered partial variable merges, leads to a contradiction to the minimality of  $\text{COUNT}$ . Hence, the combination of all the above cases shows that  $\text{COUNT}_{\text{move}(A, B)} + \text{COUNT}_{\text{move}(B, A)} \leq \max\{2, 2\hat{m}\}$ , for the variable set  $\hat{V}$  with  $t, l \in \hat{V}$  and number of packages  $\hat{m}$  maximal among all such variable sets. As an immediate consequence, if it holds that  $h^{C\text{seq}}(I) = h^*(I)$ , then it must also hold that  $\hat{m} = n$ , i.e.,  $\hat{V} = \mathcal{V}$ .

We finally show that if  $C$  contains from any optimal plan less than  $4n - 3$  states, then the partial variable merge corresponding to  $\mathcal{V}$  requires less than  $n$  *move*( $B, A$ ) transitions to reach the abstract goal, and hence  $h^{C\text{seq}}(I)$  cannot encode  $h^*(I)$ . Since there are exponentially many optimal plans, one for each permutation of  $p_1, \dots, p_n$ , and they all commonly visit exactly two states (the initial state and the goal

state), this hence shows that  $C$  must contain exponentially many conjunctions. We show the claim by contraposition. Assume there is an optimal plan  $\pi = \langle a_1, \dots, a_{4n} \rangle$  visiting states  $\mathcal{I} = s_0, s_1, \dots, s_{4n}$  with indices  $0 \leq i < j \leq 4n$  such that  $i+3 \leq j$  and  $s_i, s_j \notin C$ . Since  $\pi$  is optimal, one of the actions  $a_i, \dots, a_{i+3}$  must be  $move(B, A)$ . However, since  $s_i, s_j \notin C$ , both states are represented in the partial variable merge by the same abstract state, introducing a shortcut, and avoiding at least one  $move(B, A)$  transition. Hence, the minimal number of  $move(B, A)$  transitions required to reach the abstract goal must be smaller than  $n$ .

In conclusion,  $C'$  must contain exponentially many conjunctions (in  $n$ ) in order that

$$h^{C' \text{SEQ}}(\mathcal{I}) \geq h^{\text{seq}}[\Pi^C](\mathcal{I}) = h^*(\mathcal{I})$$

while  $|C|$  is polynomially bounded in  $n$ .

□

### B.3.3. Potential Heuristic Convergence (Theorem 7.5)

**Theorem 7.5.** *Let  $\Pi$  be any task in TNF, and  $U \in \mathbb{R}_0^+$ . Then there exists a set  $C$  of conjunctions s.t., with  $w$  obtained from any solution to  $\text{Pot}[\Pi_{\text{TNF}}^C, U]$  optimal for (O2),  $h_{C,w}^{\text{pot}}(s) = h^*(s)$  for all states  $s$  with  $h^*(s) \leq U$ .*

*Proof.* Let  $C = \mathcal{S}$ . Let  $M = |\mathcal{V}|$  be the number of variables, and  $N = |\mathcal{S}|$  be the number of states, and let  $\alpha = M + N$ . We first construct a feasible solution  $\hat{w}$  for  $\text{Pot}[\Pi_{\text{TNF}}^C, U]$  such that, for the corresponding conjunction weights  $w: C \mapsto \mathbb{R}$ ,  $h_{C,w}^{\text{pot}}(s) = \min\{h^*(s), \alpha U\}$ . Hence, in particular  $h_{C,w}^{\text{pot}}(s) = h^*(s)$  for all states with  $h^*(s) < U$ . We conclude the proof by showing that every feasible solution to  $\text{Pot}[\Pi_{\text{TNF}}^C, U]$ , optimal under objective (O2), must achieve the same heuristic values.

The desired  $\hat{w}$  can be constructed as follows:

$$\begin{aligned} \hat{w}(\pi_s \mapsto \perp) &= U & s \in \mathcal{S} \\ \hat{w}(\pi_s \mapsto *) &= U & s \in \mathcal{S} \\ \hat{w}(\pi_s \mapsto \top) &= \min\{U, h^*(s) - (\alpha - 1)U\} & s \in \mathcal{S} \\ \hat{w}(v \mapsto d) &= U & \text{remaining facts } v \mapsto d \end{aligned}$$

First, note that  $\hat{w}$  indeed achieves the desired heuristic values, i.e., it holds for all states  $s \in \mathcal{S}$ , where  $h^*(s) \leq \alpha U$ , that

$$\begin{aligned} h_{C,\hat{w}}^{\text{pot}}[\Pi^C](s^C) &= M \cdot U + \sum_{s' \neq s} \hat{w}(\pi_{s'} \mapsto \perp) + \hat{w}(\pi_s \mapsto \top) \\ &= M \cdot U + \sum_{s' \neq s} U + h^*(s) - (\alpha - 1)U \\ &= (\alpha - 1)U + h^*(s) - (\alpha - 1)U \\ &= h^*(s) \end{aligned}$$

For the states  $s \in \mathcal{S}$ ,  $h^*(s) > \alpha U$ , we have

$$\begin{aligned} h_{C,\hat{w}}^{\text{pot}}[\Pi^C](s^C) &= M \cdot U + \sum_{s' \neq s} \hat{w}(\pi_{s'} \mapsto \perp) + \hat{w}(\pi_s \mapsto \top) \\ &= M \cdot U + \sum_{s' \neq s} U + U \\ &= \alpha \cdot U \end{aligned}$$

Further note that the weights defined by  $\hat{w}$  are all bounded from above by  $U$  by construction.  $\hat{w}$  guarantees goal-awareness, i.e., satisfies Equation (7.7), as

$$\begin{aligned} \sum_{p \in \mathcal{G}^C} \hat{w}(p) &= M \cdot U + \sum_{s \neq s_*} \hat{w}(\pi_s \mapsto \perp) + \hat{w}(\pi_{s_*} \mapsto \top) \\ &= (\alpha - 1)U + h^*(s_*) - (\alpha - 1)U \\ &= 0 \end{aligned}$$

where  $s_*$  is the unique (as per the TNF assumption) goal state.

We finally need to show that  $\hat{w}$  satisfies the consistency constraints of all actions of  $\Pi_{\text{TNF}}^C$ . To simply presentation, we abuse notion and overload  $\text{PROD}$  and  $\text{CONS}$ , denoting  $\text{PROD}(a^C) = \{p \in \mathcal{F}^{\Pi^C} \mid a^C \in \text{PROD}(p)\}$ , and  $\text{CONS}(a^C)$  likewise. We distinguish between the following four cases:

- For the auxiliary  $\text{unset}_{\pi_s \mapsto \perp}$  action, we have

$$\hat{w}(\pi_s \mapsto \perp) - \hat{w}(\pi_s \mapsto *) \leq 0$$

which is trivially satisfied.

- For the auxiliary  $\text{unset}_{\pi_s \mapsto \top}$  action, we have

$$\hat{w}(\pi_s \mapsto \top) - \hat{w}(\pi_s \mapsto *) \leq U - \hat{w}(\pi_s \mapsto *) = U - U = 0$$

- For the action occurrences  $a^C$  with  $\text{vars}(\text{regress}(C, a)) \subset \mathcal{V}$ , we have

$$\begin{aligned} &\sum_{p \in \text{CONS}(a^C)} \hat{w}(p) - \sum_{p \in \text{PROD}(a^C)} \hat{w}(p) \\ &= \sum_{p \in \text{CONS}(a)} \hat{w}(p) + \sum_{s \in \mathcal{S}, \text{pre}(a) \subseteq s} \hat{w}(\pi_s \mapsto *) - \sum_{p \in \text{PROD}(a)} \hat{w}(p) - \sum_{s \in \mathcal{S}, \text{pre}(a) \subseteq s} \hat{w}(\pi_s \mapsto \perp) \\ &= \sum_{s \in \mathcal{S}, \text{pre}(a) \subseteq s} \hat{w}(\pi_s \mapsto *) - \sum_{s \in \mathcal{S}, \text{pre}(a) \subseteq s} \hat{w}(\pi_s \mapsto \perp) \\ &= \sum_{s \in \mathcal{S}, \text{pre}(a) \subseteq s} (\hat{w}(\pi_s \mapsto *) - \hat{w}(\pi_s \mapsto \perp)) \\ &= 0 \leq \mathfrak{c}(a) \end{aligned}$$

where the production and consumption of the original facts from  $\Pi$  cancel out as per the TNF assumption; and as for the chosen conjunctions  $s = c \in C$ ,  $c \parallel \text{pre}(a)$  iff  $\text{pre}(a) \subseteq c$ ; and for every such  $c$ , it holds  $c \not\parallel \text{eff}(a)$ , i.e.,  $\pi_c \mapsto \perp \in \text{eff}(a^C)$ . However,  $c \not\subseteq \text{regress}(C, a)$ , so it must be  $\pi_c \mapsto * \in \text{pre}(a^C)$ .  $a^C$  cannot produce any  $\pi_s \mapsto \top$  fact, as  $\pi_c \mapsto \top \in \text{eff}(a^C)$  implies  $\mathcal{V} = \text{vars}(s) \subseteq \text{regress}(C, a)$  as per the TNF assumption.

- Finally, consider the action occurrences  $a^C$  where  $\text{vars}(\text{regress}(C, a)) = \mathcal{V}$ . Let  $s = \text{regress}(C, a)$  be the corresponding source state, and let  $t = s[a]$  be the associated target state. By the definition of  $a^C$ , and as per the chosen conjunctions  $C$ , we have  $\text{pre}(a^C) = \text{pre}(a) \cup \{\pi_s \mapsto \top\} \cup \{\pi_{s'} \mapsto \perp \mid s' \neq s\}$  and  $\text{eff}(a^C) = \text{eff}(a) \cup \{\pi_s \mapsto \perp, \pi_t \mapsto \top\}$ . This yields:

$$\begin{aligned} &\sum_{p \in \text{CONS}(a^C)} \hat{w}(p) - \sum_{p \in \text{PROD}(a^C)} \hat{w}(p) \\ &= \hat{w}(\pi_s \mapsto \top) + \hat{w}(\pi_t \mapsto \perp) - \hat{w}(\pi_s \mapsto \perp) - \hat{w}(\pi_t \mapsto \top) \\ &= \hat{w}(\pi_s \mapsto \top) - \hat{w}(\pi_t \mapsto \top) = \dots \text{ (to be continued next)} \end{aligned}$$

where, again, the original facts cancel out due to the TNF assumption. We distinguish between the four cases, according to the  $\hat{w}(\pi_s \mapsto \top)$  and  $\hat{w}(\pi_t \mapsto \top)$  definitions:

- If  $h^*(s) > \alpha \cdot U$ , and  $h^*(t) > \alpha \cdot U$ , then

$$\dots = U - U = 0 \leq \mathfrak{c}(a)$$

- If  $h^*(s) \leq \alpha \cdot U$ , and  $h^*(t) > \alpha \cdot U$ , then

$$\dots = h^*(s) - (\alpha - 1)U - U = h^*(s) - \alpha \cdot U \leq 0 \leq \mathfrak{c}(a)$$

- Assume  $h^*(s) > \alpha \cdot U$ , and  $h^*(t) \leq \alpha \cdot U$ . Note that the former implies that  $h^*(t) + \mathfrak{c}(a) > \alpha \cdot U$ . So,

$$\dots = U - h^*(t) + (\alpha - 1)U = \alpha \cdot U - h^*(t) \leq \mathfrak{c}(a)$$

- If  $h^*(s) \leq \alpha \cdot U$ , and  $h^*(t) \leq \alpha \cdot U$ , then

$$\begin{aligned} \dots &= h^*(s) - (\alpha - 1)U - h^*(t) + (\alpha - 1)U \\ &= h^*(s) - h^*(t) \leq \mathfrak{c}(a) \end{aligned}$$

which is satisfied as  $h^*$  is consistent.

In all cases, Equation (7.6) is satisfied.

We conclude that  $\hat{w}$  satisfies the consistency constraints, Equation (7.6), for all action occurrences.

Finally, consider any solution  $\hat{w}'$  to  $\text{Pot}[\Pi_{\text{TNF}}^C, U]$  that is optimal under objective (O2). As per the definition of (O2):

$$\sum_{s \in \mathcal{S}} \frac{1}{|\mathcal{S}|} h_{C, \hat{w}'}^{\text{pot}}[\Pi^C](s^C) \geq \sum_{s \in \mathcal{S}} \frac{1}{|\mathcal{S}|} h_{C, \hat{w}}^{\text{pot}}[\Pi^C](s^C)$$

Suppose for contradiction, there were some states  $s$  such that  $h_{C, \hat{w}'}^{\text{pot}}[\Pi^C](s^C) \neq h_{C, \hat{w}}^{\text{pot}}[\Pi^C](s^C)$ . In particular, as per the above relation, there must then be some state  $s$  for which  $h_{C, \hat{w}'}^{\text{pot}}[\Pi^C](s^C) > h_{C, \hat{w}}^{\text{pot}}[\Pi^C](s^C)$  (otherwise the sum couldn't be larger or equal). For all states  $t$  with  $h^*(t) > \alpha \cdot U$ , it holds that

$$\begin{aligned} h_{C, \hat{w}}^{\text{pot}}[\Pi^C](t^C) &= \alpha U \\ &= M \cdot U + (N - 1) \cdot U + U \\ &\geq \sum_{v \in \mathcal{V}} \hat{w}'(v \mapsto t^C[v]) + \sum_{s' \neq t} \hat{w}'(\pi_{s'} \mapsto \perp) + \hat{w}'(\pi_t \mapsto \top) \\ &= h_{C, \hat{w}'}^{\text{pot}}[\Pi^C](t^C) \end{aligned}$$

Hence, for the selected state  $s$ ,  $h^*(s) \leq \alpha \cdot U$ . But then,  $h_{C, \hat{w}}^{\text{pot}}[\Pi^C](s^C) = h^*(s)$ , as per our construction, which means that  $h_{C, \hat{w}'}^{\text{pot}}[\Pi^C](s^C) > h^*(s)$ . This is a contradiction to the admissibility of  $h_{C, \hat{w}'}^{\text{pot}}[\Pi^C]$ . In conclusion, such a state  $s$  cannot exist. This completes the proof.

□

**Learning Effectiveness: State Equation vs. Critical-Path Heuristics (Propositions 7.1 and 7.2)**

**Proposition 7.1.** *There exist planning tasks  $\Pi$ , dead-end states  $s$ , and sets of conjunctions  $C$  such that  $\mathcal{U}^{\text{seq}}[\Pi^C](s) = \infty$ , but in order that  $\mathcal{U}^{C'}(s) = \infty$ ,  $|C'|$  must be exponential in  $|\Pi^C|$ .*

*Proof.* Consider the planning tasks  $\Pi_n = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  with components

- $\mathcal{V} = \{v_i \mid 1 \leq i \leq n\} \cup \{u\}$  with domains  $\mathcal{D}_{v_i} = \{0, 1\}$  and  $\mathcal{D}_u = \{0, 1, \dots, n-1\}$ ,
- $\mathcal{A} = \{a_{i,k} \mid 1 \leq i \leq n, 0 < k < n\}$ , where

$$\text{pre}(a_{i,k}) = \{v_i \mapsto 0, u \mapsto k\} \quad \text{eff}(a_{i,k}) = \{v_i \mapsto 1, u \mapsto k-1\}$$

- $\mathcal{I} = \{v_i \mapsto 0 \mid 1 \leq i \leq n\} \cup \{u \mapsto n-1\}$
- $\mathcal{G} = \{v_i \mapsto 1 \mid 1 \leq i \leq n\}$

The tasks are clearly unsolvable. Moreover,  $\mathcal{U}^{\text{seq}}(\mathcal{I}) = \infty$  because, intuitively, producing all  $v_i \mapsto 1$  is not possible, while consuming no  $u \mapsto k$  fact more than once. More precisely, consider the SEQ constraint for the  $u$  facts:

$$\begin{aligned} u \mapsto n-1: & \quad \sum_{i=1}^n \text{COUNT}_{a_{i,n-1}} \leq 1 \\ u \mapsto k: & \quad \sum_{i=1}^n \text{COUNT}_{a_{i,k}} - \sum_{i=1}^n \text{COUNT}_{a_{i,k+1}} \leq 0 \end{aligned}$$

where  $0 < k < n-1$ . It follows via induction on  $k$  that

$$\sum_{i=1}^n \text{COUNT}_{a_{i,k}} \leq 1$$

for all  $0 < k < n$ . However, to satisfy all goal-fact constraints, it must be

$$\sum_{i=1}^n \sum_{k=1}^{n-1} \text{COUNT}_{a_{i,k}} \geq n$$

This is in contradiction to the inequality above.

Finally, we show that  $\mathcal{U}^C(\mathcal{I}) = \infty$  requires  $C$  to contain (at least) one conjunction for every subset of  $\mathcal{G}$ . More specifically, we show that for each non-empty set  $\emptyset \subset x \subset \{v_i \mapsto 1 \mid 1 \leq i \leq n\}$ , there must exist  $c \in C$  such that  $c[v_i] = 1$  iff  $x[v_i] = 1$ , and  $c[u] = n - |x|$ . Suppose for contradiction that  $C$  does not contain such a conjunction for some  $x$ . Without loss of generality, assume  $x = \{v_1 \mapsto 1, \dots, v_j \mapsto 1\}$ , for some  $1 \leq j < n$  (the variable indices are interchangeable). The following inequalities are satisfied:

$$\begin{aligned} \mathcal{U}^C(\mathcal{I}) &= \mathcal{U}^C(\mathcal{I}, \mathcal{G}) \\ &\leq \mathcal{U}^C(\mathcal{I}, \text{regress}(\mathcal{G}, a_{n,n-1})) \\ &\leq \mathcal{U}^C(\mathcal{I}, \text{regress}(\text{regress}(\mathcal{G}, a_{n,n-1}), a_{n-1,n-2})) \\ &\leq \mathcal{U}^C(\mathcal{I}, \text{regress}(\text{regress}(\text{regress}(\mathcal{G}, a_{n,n-1}), a_{n-1,n-2}), a_{n-2,n-3})) \\ &\vdots \\ &\leq \mathcal{U}^C(\mathcal{I}, \text{regress}(\dots, a_{j+1,n-j})) \\ &= \mathcal{U}^C(\mathcal{I}, \underbrace{x \cup \{v_i \mapsto 0 \mid j+1 \leq i \leq n\} \cup \{u \mapsto n-j\}}_{=:P}) \end{aligned}$$

By assumption, for every  $c \in C$ ,  $c \subseteq P$ ,  $c$  contains strictly less than  $j$  facts  $v_i \mapsto 1$ . However, every such conjunction  $c$  is reachable, i.e.,  $h^*(s, c) < \infty$ , and thus  $u^C(s, c) < \infty$ . Hence,  $u^C(I, P) < \infty$ , and consequently  $u^C(I) < \infty$ . The claim follows, because there are exponentially many  $x$ , and each  $x$  defines at least one unique conjunction  $c_x \in C$ .  $\square$

**Proposition 7.2.** *There exist planning tasks  $\Pi$ , sets of conjunctions  $C$ , and dead-end states  $s$ , where  $u^C(s) = \infty$ , but in order that  $u^{\text{seq}}[\Pi^C](s) = \infty$ ,  $|\Pi^C|$  must be exponential in  $|C|$ .*

*Proof.* A family of planning tasks satisfying the claim is given by  $\Pi_n = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  with components

- $\mathcal{V} = \{v_i \mid 1 \leq i \leq n\}$  with domains  $\mathcal{D}_{v_i} = \{0, 1, 2\}$
- $\mathcal{A} = \{ \text{set0}(i) \mid 1 \leq i \leq n \}$   
 $\cup \{ \text{set1}(i, j) \mid 1 \leq i, j \leq n, i \neq j \}$   
 $\cup \{ \text{set2}(i) \mid 1 \leq i \leq n \}$

where

Action $a$	pre( $a$ )	eff( $a$ )
$\text{set0}(i)$	$\{v_i \mapsto 2\}$	$\{v_i \mapsto 0\}$
$\text{set1}(i, j)$	$\{v_i \mapsto 1, v_j \mapsto 0\}$	$\{v_j \mapsto 1\}$
$\text{set2}(i)$	$\{v_i \mapsto 0\}$	$\{v_i \mapsto 2\}$

- $\mathcal{I} = \{v_i \mapsto 0 \mid 1 \leq i \leq n\}$ ,
- $\mathcal{G} = \{v_i \mapsto 1 \mid 1 \leq i \leq n\}$ .

For the critical-path heuristic, the singleton conjunction already suffice to recognize  $\mathcal{I}$ , i.e.,  $u^1(\mathcal{I}) = \infty$ , because of the mutual dependency between  $v_i \mapsto 1$  and  $v_j \mapsto 1$ .

However, in order that  $u^{\text{seq}}[\Pi^C](\mathcal{I}) = \infty$ ,  $|\Pi^C|$  must be exponential in  $n$ . Namely, consider the set of dead-end states

$$S = \{s \in \mathcal{S}^{\Pi_n} \mid \forall i: s[v_i] \in \{0, 2\}\}$$

Note that all  $s \in S$  are reachable from  $\mathcal{I}$ . Hence, due to transitivity property, if  $u^{\text{seq}}[\Pi^C](\mathcal{I}) = \infty$ , then  $u^{\text{seq}}[\Pi^C](s) = \infty$  for all  $s \in S$ . This entails that, for each  $1 \leq j \leq n$ , every  $s \in S$  with  $s[v_j] = 0$ , and each  $1 \leq i \leq n$ ,  $i \neq j$ ,  $C$  contains a conjunction  $c$  such that  $v_i, v_j \in \text{vars}(c)$ , and either

- (i)  $c[v_i] = 1$  and  $(c \setminus \{v_i \mapsto 1\}) \subseteq s$ , or
- (ii)  $c \subseteq s$ .

If for some  $i$  and  $j$ , such a conjunction did not exist, one can construct a solution to the  $\text{SEQ}[\Pi^C](s)$  LP as follows. Let  $t$  be identical to  $s$ , but  $t[v_i] = 1$  and  $t[v_j] = 1$ . Note that  $t$  is solvable. To produce all conjunctions  $c \subseteq t$ , where  $c \not\subseteq s$ , while consuming all conjunctions  $c \subseteq s$ , where  $c \not\subseteq t$ , it suffices to set  $\text{COUNT}_{\text{set1}(i,j)^C} = 1$  with

$$C = \{c \in C \mid c \subseteq t, c[v_j] = 1\}$$

With the absence of conjunctions (i) and (ii), all  $\Pi^C$  facts consumed by  $\text{set1}(i, j)^C$  are provided by  $s^C$ , and there exists no  $\Pi^C$  fact from  $s^C$  not in  $t^C$  that is not also consumed by  $\text{set1}(i, j)^C$ . Moreover, by construction of  $C$ ,  $\text{set1}(i)^C$  produces all  $\Pi^C$  facts contained in  $t^C \setminus s^C$ . To obtain a feasible solution, it then only remains to fill the COUNT values according to a plan for the state  $t$ .

If  $C$  contained exponentially many conjunctions, the claim would follow immediately. If  $|C|$  is not exponential in  $n$ , there must be an upper limit on the size of the *context*, enumerated via the conjunctions in  $C$ , i.e., some  $k$  such that for all  $x \subseteq \{v_i \mapsto d \mid 1 \leq i \leq n, d \in \{0, 2\}\}$ ,  $|x| \leq k$ , and all  $v_i, v_j \notin \text{vars}(x)$ : either (i)  $x \cup \{v_i \mapsto 1, v_j \mapsto 0\} \in C$  or (ii)  $x \cup \{v_i \mapsto 0, v_j \mapsto 0\} \in C$  and  $x \cup \{v_i \mapsto 2, v_j \mapsto 0\} \in C$ . The action occurrences of  $\text{set0}(j)$  enumerate all subsets of the conjunctions. The number of variables not enumerated by  $x$ , i.e.,  $n - k$ , must still be polynomial in  $n$ . Hence, the number of corresponding conjunctions must be polynomially related to  $n$ , and therewith the number of subsets (hence action occurrences) scales exponentially in  $n$ . This concludes the proof.  $\square$

## B.4. On Unsolvability Detectors, the Traps They Set, and Trap Learning

### B.4.1. Learning Effectiveness: Comparison to the State-Equation and Critical-Path Heuristics (Propositions 8.1 and 8.2 and 8.3)

**Proposition 8.1.** *There exist planning tasks  $\Pi$ , dead-end states  $s$ , and sets of conjunctions  $\Gamma$  satisfying (i) and (ii) of Lemma 8.1, where  $s \in T^\Gamma$  but in order that  $\mathcal{U}^C(s) = \infty$ ,  $|C|$  must be exponential in  $|\Gamma|$ .*

*Proof.* A family of tasks  $\Pi_n = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  suiting these requirements is given by an FDR variant of the task from of Proposition 6.2.

- $\mathcal{V} = \{v_i \mid 1 \leq i \leq n\} \cup \{u\}$  with domains  $\mathcal{D}_{v_i} = \{0, 1, 2\}$  and  $\mathcal{D}_u = \{0, 1\}$
- $\mathcal{A} = \{a_i \mid 1 \leq i \leq n\} \cup \{b_i \mid 1 \leq i \leq n\}$  with
 
$$\begin{aligned} \text{pre}(a_i) &= \{v_i \mapsto 0, u \mapsto 0\} & \text{eff}(a_i) &= \{v_i \mapsto 2, u \mapsto 1\} \\ \text{pre}(b_i) &= \{v_i \mapsto 0, u \mapsto 1\} & \text{eff}(b_i) &= \{v_i \mapsto 1\} \end{aligned}$$
- $\mathcal{I} = \{v_i \mapsto 0 \mid 1 \leq i \leq n\} \cup \{u \mapsto 0\}$
- $\mathcal{G} = \{v_i \mapsto 1 \mid 1 \leq i \leq n\} \cup \{u \mapsto 1\}$

As shown in the proof of Proposition 6.2,  $\mathcal{U}^C(\mathcal{I}) = \infty$  entails that  $C$  contains an exponential number of conjunctions in  $n$ . However, consider  $\Gamma = \{\{v_i \mapsto 2\} \mid 1 \leq i \leq n\} \cup \{u \mapsto 0\}$ .  $\Gamma$  obviously satisfies trap condition (i), given that every such fact disagrees with the goal. Trap condition (ii) is satisfied because  $\{v_i \mapsto 2\}$  is invariant under all transitions;  $\{u \mapsto 0\} \not\models \text{pre}(b_i)$ ; and the progression of  $\{u \mapsto 0\}$  via any  $a_i$  makes true the respective  $\{v_i \mapsto 2\}$ . Therefore,  $T^\Gamma$  is a trap, and it holds that  $\mathcal{I} \in T^\Gamma$ .  $\square$

**Proposition 8.2.** *There exist planning tasks  $\Pi$ , dead-end states  $s$ , and sets of conjunctions  $C$ , where  $\mathcal{U}^C(s) = \infty$ , but for every  $\Gamma$  satisfying (i) and (ii) of Lemma 8.1 with  $s \in T^\Gamma$ ,  $|\Gamma|$  is exponential in  $|C|$ .*

*Proof.* Such a task is given in Proposition 7.2. Let  $\Gamma$  be any set of conjunctions satisfying (i) and (ii) of Lemma 8.1 such that  $\mathcal{I} \in T^\Gamma$ . Reconsider the set of dead ends

$$S = \{s \in \mathcal{S}^{\Pi_n} \mid \forall i: s[v_i] \in \{0, 2\}\}$$

Observe that  $S \subseteq \Gamma$ . Suppose not. Consider any  $s \in S \setminus \Gamma$ . Since all states from  $S$  are reachable from  $\mathcal{I}$ ,  $\mathcal{I} \in T^\Gamma$  implies that  $s \in T^\Gamma$ . Hence, there exists some  $c \in \Gamma$  such that  $c \subseteq s$ , and it must be  $c \subset s$  by assumption. Let  $v_i \notin \text{vars}(c)$ . We can construct a solvable state  $t$  where  $c \subseteq t$  by defining  $t$  to be identical

to  $s$  but  $t[v_i] = 1$ . However, with  $t \in T^\Gamma$ , we get a contradiction to the assumption that  $T^\Gamma$  is a trap. Hence,  $S \subseteq \Gamma$ . In contrast, as shown in Proposition 7.2, for  $\mathcal{U}^C(\mathcal{I}) = \infty$ , it suffices that  $C$  contains just the singleton conjunctions. This concludes the proof.  $\square$

**Proposition 8.3.** *There exist planning tasks  $\Pi$ , dead-end states  $s$ , and sets of conjunctions  $C$  such that  $\mathcal{U}^{\text{seq}}[\Pi^C](s) = \infty$ , but for every  $\Gamma$  satisfying (i) and (ii) of Lemma 8.1 with  $s \in T^\Gamma$ ,  $|\Gamma|$  is exponential in  $|\Pi^C|$ .*

*Proof.* Such a task is given in Proposition 7.1. Let  $\Gamma$  be any set of conjunctions satisfying (i) and (ii) of Lemma 8.1 such that  $\mathcal{I} \in T^\Gamma$ . Consider the dead-end states

$$S = \{ s \in \mathcal{S}^{\Pi_n} \mid |V_0(s)| \geq 1, s[u] = |V_0(s)| - 1 \}$$

where  $V_0(s) = \{ v_i \mid 1 \leq i \leq n, s[v_i] = 0 \}$ . Note that all states from  $S$  are reachable from  $\mathcal{I}$ . Given that  $T^\Gamma$  is a dead-end trap and  $\mathcal{I} \in T^\Gamma$ , it follows that  $S \subseteq T^\Gamma$ . Hence, for every  $s \in S$ , there must exist a conjunction  $c_s \in \Gamma$  such that  $c_s \subseteq s$ . Observe that  $V_0(s) \cup \{ u \} \subseteq \text{vars}(c_s)$ . Suppose not. Then, no matter whether  $v_i \notin \text{vars}(c_s)$  or  $u \notin \text{vars}(c_s)$ , we can construct a solvable state  $s'$  such that  $c_s \subseteq s'$ . This contradicts the assumption that  $T^\Gamma$  is a trap. But, with  $V_0(s) \cup \{ u \} \subseteq \text{vars}(c_s)$ ,  $\Gamma$  must contain a distinct conjunction for each subset of  $\{ v_i \mapsto 0 \mid 1 \leq i \leq n \}$ , and hence  $|\Gamma|$  is exponential in  $n$ . In contrast, as shown in Proposition 7.1,  $\mathcal{U}^{\text{seq}}(\mathcal{I}) = \infty$ , without the need of any additional conjunction.  $\square$

# C. Proofs of Part III

## C.1. Fundamental Algorithms

### C.1.1. Proof that Bellman Residuals are Monotonically Decreasing

**Theorem C.1.** Let  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, s_I, \mathcal{S}_* \rangle$  be an MDP. Let  $V^{(0)}$  be a monotone (lower or upper) bound for  $\mathcal{M}$ . Let  $V^{(i)} = BV^{(i-1)}$  for all  $i \geq 1$ . Then  $\Delta V^{(i)} \geq \Delta V^{(i+1)}$  for all  $i \geq 0$ .

*Proof.* Suppose  $V^{(0)}$  is a monotone lower bound. The case of upper bounds can be shown analogously. Let  $s \in \mathcal{S}$  be arbitrary. We show that  $(\Delta V^{(i+1)})(s) \leq \Delta V^{(i)}$ . If  $s$  is a terminal or goal state, then  $V^{(i+1)}(s) = V^*(s)$ , and therefore  $(BV^{(i+1)})(s) - V^{(i+1)}(s) = V^*(s) - V^*(s) = 0$ , i.e.,  $(\Delta V^{(i+1)})(s) = 0$  and the claim is trivially satisfied. Suppose that  $s$  is neither terminal nor a goal state. Let  $a \in \mathcal{A}(s)$  be the action with maximal  $(QV^{(i+1)})(s, a)$  value, i.e.,  $(BV^{(i+1)})(s) = (QV^{(i+1)})(s, a)$ . Then

$$\begin{aligned}
 (\Delta V^{(i+1)})(s) &= (QV^{(i+1)})(s, a) - V^{(i+1)}(s) \\
 &= \sum_{s'} \mathcal{P}(s, a, s') V^{(i+1)}(s') - V^{(i+1)}(s) \\
 &= \sum_{s'} \mathcal{P}(s, a, s') (V^{(i)}(s') + (\Delta V^{(i)})(s')) - V^{(i+1)}(s) \\
 &\leq \sum_{s'} \mathcal{P}(s, a, s') (V^{(i)}(s') + \Delta V^{(i)}) - V^{(i+1)}(s) \\
 &= \sum_{s'} \mathcal{P}(s, a, s') V^{(i)}(s') - V^{(i+1)}(s) + \Delta V^{(i)} \\
 &= (QV^{(i)})(s, a) - V^{(i+1)}(s) + \Delta V^{(i)} \\
 &\leq \Delta V^{(i)}
 \end{aligned}$$

□

### C.1.2. Correctness Proof of VI Policy Extraction (Theorem 11.6 and Lemma 11.1)

**Theorem 11.6.** Let  $V$  be a monotone value function. Algorithm 11.2 on  $V$  terminates. Suppose  $\pi_V$  is the resulting policy. If there exists a policy  $\pi$  with  $V^\pi \geq V$  that is greedy on  $V$ , then  $V^{\pi_V} \geq V$ .

*Proof.* Algorithm 11.2 terminates eventually since each state can be processed at most once, and the number of states is finite. Let  $\pi_V$  be the resulting policy.

By assumption, there exists a greedy policy  $\pi$  of  $V$  with  $V^\pi \geq V$ . Due to the exhaustive backpropagation done by Algorithm 11.2,  $\pi_V(s)$  must be defined for all non-goal states with  $V(s) > 0$ . Namely, for every such state, it holds  $V^\pi(s) > 0$  by definition. Therefore, there must be a path from  $s$  to some goal state with action labels according to the selections by  $\pi$ . Given that Algorithm 11.2 considers all greedy actions, this path will eventually be considered, leading to an assignment of  $\pi_V(s)$ .

To show that  $V^{\pi_V}(s) \geq V(s)$  for all states  $s \in \mathcal{S}$ , consider the policy graph  $\mathcal{M}^{\pi_V}$ . Let  $V^*[\mathcal{M}^{\pi_V}]$  be the optimal goal probability value function in  $\mathcal{M}^{\pi_V}$ . Observe that  $V^*[\mathcal{M}^{\pi_V}] = V^{\pi_V}$ . By construction, executing  $\pi_V$  from any state  $s$  with  $\pi_V(s) \neq \perp$  will eventually end in either a goal state, or a state where  $\pi_V$  is not defined. The same must necessarily also hold for all other policies in  $\mathcal{M}^{\pi_V}$ , since the only choice left is whether to apply the action already chosen by  $\pi_V$ , or not to apply any action. Therefore,  $\mathcal{M}^{\pi_V}$  is an SSP. So, the Bellman equations for  $\mathcal{M}^{\pi_V}$  have a unique solution, and this solution is  $V^*[\mathcal{M}^{\pi_V}]$ . This also means that  $V^*[\mathcal{M}^{\pi_V}]$  is the fixed point to the sequence of value functions  $V^{(0)}, V^{(1)}, \dots$  where  $V^{(0)} = V$  and  $V^{(i)} = \mathbf{B}^{\pi_V} V^{(i-1)}$ . By  $V$ 's monotonicity assumption, it holds that  $V \leq \mathbf{B}V$ . By definition of  $\pi_V$ , it holds for all states  $s$  with  $\pi_V(s) \neq \perp$  that  $(\mathbf{Q}V)(s, \pi_V(s)) = \mathbf{B}V$ . Thus,  $(\mathbf{B}^{\pi_V} V)(s) = (\mathbf{B}V)(s) \geq V(s)$ . For all states  $s$  with  $\pi_V(s) = \perp$ , it either holds that  $s$  is a goal state, or  $V(s) = 0$ . In either case,  $(\mathbf{B}^{\pi_V} V)(s) \geq V(s)$ . In combination,  $V^{(0)} \leq V^{(1)}$ . As per Theorem 11.2,  $\mathbf{B}^{\pi_V}$  preserves monotonicity, i.e., it holds that  $V^{(0)} \leq V^{(\infty)}$ . Since  $V^{(0)} = V$  as per definition, and  $V^{(\infty)} = V^*[\mathcal{M}^{\pi_V}] = V^{\pi_V}$  given that  $\mathcal{M}^{\pi_V}$  is an SSP, the claim  $V \leq V^{\pi_V}$  follows. □

**Lemma 11.1.** Suppose  $V^{(0)}(s) = 0$  for all states  $s \in \mathcal{S}$ , and, for  $i > 0$ ,  $V^{(i)} = \mathbf{B}V^{(i-1)}$ . At any point  $i \geq 0$ , and for every non-goal state  $s_0 \in \mathcal{S} \setminus \mathcal{S}_*$  with  $V^{(i)}(s_0) > 0$ , there exists a path  $s_0, a_0, s_1, a_1, \dots, s_n$  such that

- (i)  $s_n \in \mathcal{S}_*$ , and
- (ii)  $\forall 0 \leq j < n$ ,  $a_j$  is greedy on  $V^{(i)}$  for  $s_j$ , and
- (iii)  $\forall 0 \leq j < n$ ,  $V^{(i)}(s_j) \leq V^{(i)}(s_{j+1})$ .

*Proof.* The proof is via a nested induction. The outer induction is on  $i$ . For the induction beginning, note that the claim is trivially satisfied for  $i = 0$  and  $i = 1$ . For  $i = 0$ ,  $V^{(0)}(s) = 0$  holds for all states. For  $i = 1$ ,  $V^{(1)}(s) = 1$  iff  $s$  is a goal-state, and  $V^{(1)}(s) = 0$  holds for all other states. Suppose as the induction hypothesis (IH1) that the claim is satisfied for all value functions up to (and including)  $V^{(i)}$ . We show the induction step  $i + 1$  via a second induction, now on the  $V^{(i+1)}$  values (more specifically, we do an induction over a sequence of states, in which states are ordered from large to small  $V^{(i+1)}(s)$  values).

For the induction beginning, let  $s \in \mathcal{S}$  be any non-goal state with  $V^{(i+1)}(s) > 0$  and whose  $V^{(i+1)}(s)$  value is maximal among all non-goal states. Let  $a \in \mathcal{A}(s)$  be any action that is greedy on  $V^{(i)}$  for  $s$ , i.e.,  $V^{(i+1)}(s) = (\mathbf{Q}V^{(i)})(s, a)$ . Let  $s_1 \in \text{Succ}(s, a)$  be any successor with  $V^{(i)}(s_1) \geq (\mathbf{Q}V^{(i)})(s, a) > 0$ . If  $s_1$  is a goal state, we are done immediately. Otherwise, we use (IH1) to obtain a goal-path  $s_1, a_1, s_2, \dots, s_n$  that satisfies (ii) and (iii) w.r.t.  $V^{(i)}$ . Let  $1 \leq j < n$  be the first position along this path such that either (ii) or (iii) w.r.t.  $V^{(i+1)}$  is violated. If such a  $j$  does not exist, then  $s, a, s_1, a_1, \dots, s_n$  would already constitute the requested path for  $s$  and  $V^{(i+1)}$ .

- If (ii) is violated, then there must be another action  $a'_j \in \mathcal{A}(s_j)$  that achieves a higher  $\mathbf{Q}$ -value:  $(\mathbf{Q}V^{(i+1)})(s_j, a'_j) > (\mathbf{Q}V^{(i+1)})(s_j, a_j)$ . Choose again some successor  $s'_{j+1} \in \text{Succ}(s_j, a'_j)$  that supports that  $\mathbf{Q}$ -value with a sufficient goal-probability value:  $V^{(i+1)}(s'_{j+1}) \geq (\mathbf{Q}V^{(i+1)})(s_j, a'_j)$ . Ob-

serve that

$$\begin{aligned}
V^{(i+1)}(s) &\leq V^{(i)}(s_1) \\
&\stackrel{(1)}{\leq} V^{(i+1)}(s_1) \\
&\stackrel{(2)}{\leq} V^{(i+1)}(s_j) \\
&\stackrel{(3)}{=} (QV^{(i)})(s_j, a_j) \\
&\stackrel{(4)}{\leq} (QV^{(i+1)})(s_j, a_j) \\
&< (QV^{(i+1)})(s_j, a'_j) \\
&\leq V^{(i+1)}(s'_{j+1})
\end{aligned}$$

where (1) and (4) hold due to monotonicity; (2) holds because (iii) is satisfied up to point  $j$  by assumption; and (3) since  $a_j$  is greedy on  $V^{(i)}$  for  $s_j$ . Since  $s$  has maximal  $V^{(i+1)}$ -value among all non-goal states by assumption, we conclude that  $s'_{j+1}$  is a goal state. The path  $s, a, s_1, a_1, \dots, s_j, a'_j, s'_{j+1}$  shows the claim.

- If (iii) is violated, then  $V^{(i+1)}(s_j) > V^{(i+1)}(s_{j+1})$ . Observe that

$$\begin{aligned}
V^{(i+1)}(s) &\leq V^{(i)}(s_1) \\
&\stackrel{(1)}{\leq} V^{(i)}(s_{j+1}) \\
&\stackrel{(2)}{\leq} V^{(i+1)}(s_{j+1}) \\
&< V^{(i+1)}(s_j)
\end{aligned}$$

where (1) holds since the path satisfies (iii) on  $V^{(i)}$  by its construction; (2) holds due to monotonicity. Again, by assumption,  $s_j$  must be a goal state, and the path  $s, a, s_1, a_1, \dots, s_{j-1}, a_{j-1}, s_j$  concludes the claim.

For the induction step, consider any non-goal state  $s$  with  $V^{(i+1)}(s) > 0$ . Suppose as the second induction hypothesis (IH2) that, for all non-goal states  $s'$  with  $V^{(i+1)}(s') > V^{(i+1)}(s)$ , there exists a path as in the claim. Consider an action  $a$ , successor state  $s_1$ , and path  $\sigma = s_1, a_1, s_2, \dots, s_n$  just as in the proof of the induction beginning. Via the exact same arguments, this path violates (ii) or (iii) for  $V^{(i+1)}$  only if one reaches some state  $s'$  with  $V^{(i+1)}(s') > V^{(i+1)}(s)$ . If  $s'$  is a goal state, then we can construct the requested path for  $s$  in the exact same manner as in the induction beginning. If  $s'$  is not a goal state, then as per (IH2), there exists a goal-path  $\sigma'$  starting from  $s'$  that satisfies (ii) and (iii) w.r.t.  $V^{(i+1)}$ . We obtain the requested path for  $s$  by concatenating  $s, a$ ; the part of  $\sigma$  up to the failure point; and  $\sigma'$ .

□

## C.2. MDP Heuristic Search

### C.2.1. Relation Between SSPs and the Existence of Traps (Theorem 12.6)

**Theorem 12.6.**  $\mathcal{M}$  is an SSP if and only if  $\mathcal{M}$  contains no reachable trap.

*Proof.* Suppose  $\mathcal{M}$  is not an SSP. There must exist a policy  $\pi$  and a reachable state  $s$  s.t.  $\mathcal{R}^\pi(s) \cap \mathcal{S}_\perp^\pi = \emptyset$  and  $\mathcal{R}^\pi(s) \cap \mathcal{S}_* = \emptyset$ . Let  $S_1, \dots, S_n$  be the maximal SCCs in the policy graph subgraph  $\mathcal{M}^\pi|_{\mathcal{R}^\pi(s)}$ , i.e., all size-maximal subsets  $S_i \subseteq \mathcal{R}^\pi(s)$  such that  $\mathcal{M}^\pi|_{\mathcal{R}^\pi(s)}$  contains a path between every distinct pair of states in  $S_i$ . There must exist at least one SCC since  $s \in \mathcal{R}^\pi(s)$ , i.e.,  $\mathcal{R}^\pi(s) \neq \emptyset$ . Given that by assumption  $\mathcal{M}^\pi|_{\mathcal{R}^\pi(s)}$  contains no terminal state, it holds for all  $t \in \mathcal{R}^\pi(s)$  that  $\pi(t) \neq \perp$ . Since the graph over the maximal SCCs is acyclic, there must be some leaf, i.e.,  $S_i$  where for each  $t \in S_i$  and  $t' \notin S_i$ ,  $\mathcal{P}(t, \pi(t), t') = 0$ . Consider the subgraph  $\mathcal{M}'$  of  $\mathcal{M}$  with states  $\mathcal{S}' = S_i$  and transition probabilities  $\mathcal{P}'$ , where for each  $t, t' \in S_i$  and  $a \in \mathcal{A}$ :

$$\mathcal{P}'(t, a, t') = \begin{cases} \mathcal{P}(t, a, t') & \text{if } a = \pi(t) \\ 0 & \text{otherwise} \end{cases}$$

This is a valid transition probability function as per the selection of  $S_i$ . Since  $S_i$  is strongly connected, if  $S_i$  contains more than one state, then all pairs of (not necessarily distinct) states from  $S_i$  are reachable from each other. If  $S_i = \{t\}$ , then  $\mathcal{P}'(t, \pi(t), t) = 1$ , i.e.,  $t$  has a self-loop. In both cases, it follows that  $\mathcal{M}'$  is an EC. Finally, by assumption,  $S_i$  is reachable and does not contain a goal state. This concludes one half of the claim.

Now, suppose that  $\mathcal{M}$  contains a reachable, goal-free EC  $\mathcal{M}'$ . Let  $\mathcal{S}'$  be its states, and  $\mathcal{P}'$  be its transition function. We construct a policy  $\pi$  that violates the SSP requirements. For each  $t \in \mathcal{S}'$ , let  $\pi(t) = a$  for some  $a \in \mathcal{A}[\mathcal{M}'](t)$ . Note that  $\mathcal{A}[\mathcal{M}'](t) \neq \emptyset$  for each  $t \in \mathcal{S}'$  since there is path between every pair of states from  $\mathcal{S}'$  in  $\mathcal{M}'$ , so every state must have at least one outgoing transition. Let  $s \in \mathcal{S}'$  be arbitrary. By construction,  $\mathcal{R}^\pi(s) \subseteq \mathcal{S}'$ . By assumption,  $\mathcal{S}'$  does not contain a goal state, i.e.,  $\mathcal{R}^\pi(s) \cap \mathcal{S}_* = \emptyset$ . Since  $\pi$  is defined on all states in  $\mathcal{S}'$ ,  $\mathcal{R}^\pi(s) \cap \mathcal{S}_\perp^\pi = \emptyset$ . By assumption,  $s$  is reachable from  $s_I$ . In conclusion, we have found a state  $s$  and policy  $\pi$  that do not comply with the SSP conditions.

□

### C.2.2. Collapsing Traps Preserves Goal Probabilities (Theorem 12.7)

**Lemma C.1.** *Let  $V$  be a fixed point of  $B$  in  $\mathcal{M}$ . Let  $T$  be a trap in  $\mathcal{M}$ . It holds for all  $s, s' \in T$  that  $V(s) = V(s')$ .*

*Proof.* Let  $T_{min} \subseteq T$  be the subset of states with minimal  $V$ -value among all states of  $T$ . Suppose for contradiction that  $T_{min} \neq T$ . As per the definition of end components, there must exist some  $s \in T_{min}$  and action  $a \in \mathcal{A}(s)$  such that  $\text{Succ}(s, a) \subseteq T$  and  $\text{Succ}(s, a) \cap (T \setminus T_{min}) \neq \emptyset$ . Since  $T$  does not contain any goal state, we obtain

$$\begin{aligned} (BV)(s) &\geq (QV)(s, a) \\ &= \sum_{t \in T_{min}} \mathcal{P}(s, a, t)V(s) + \sum_{t \in (T \setminus T_{min})} \mathcal{P}(s, a, t)V(t) \\ &> \sum_{t \in T_{min}} \mathcal{P}(s, a, t)V(s) + \sum_{t \in (T \setminus T_{min})} \mathcal{P}(s, a, t)V(s) \\ &= V(s) \end{aligned}$$

which is a contradiction to the fixed point assumption.

□

**Theorem 12.7.** *Suppose  $T$  is a trap in  $\mathcal{M}$ . Denote by  $V^*[\mathfrak{Q}^T]$  the optimal goal-probability function of the quotient  $\mathfrak{Q}^T$  of  $\mathcal{M}$  and  $T$ . It holds for all states  $s \in \mathcal{S}$  that  $V^*(s) = V^*[\mathfrak{Q}^T](\mathfrak{s}^T(s))$ .*

*Proof.* We first show  $V^*(s) \leq V[\mathfrak{Q}^T]^*(s^T(s))$ , then  $V^*(s) \geq V[\mathfrak{Q}^T]^*(s^T(s))$ . Note that in either case, we only need to argue about the values of the states of  $T$ , since  $\mathfrak{Q}^T$  changes the transition function only by rewiring transition from and into  $\mathfrak{t}$ , i.e., if they all agree on their value, then necessarily every other state must do so as well.

Regarding “ $\leq$ ”, if it holds  $V^*(t) = 0$ , for any  $t \in T$ , then  $V^*(t) = 0$  for all  $t \in T$  as per Lemma C.1. Hence, the  $\leq$  case is trivially satisfied. Otherwise, let  $\pi^*$  be a policy in  $\mathcal{M}$  that is optimal in all states.  $\pi^*(t)$  must be defined for all  $t \in T$ , since  $V^*(t) > 0$  by assumption, and  $T$  does not contain any goal state. Moreover, due to the same reasons,  $\text{Succ}[\mathcal{M}](t, \pi^*(t)) \not\subseteq T$  must be true for at least one state  $t \in T$ . Consider the policy

$$\pi^T(\mathfrak{s}) := \begin{cases} \pi^*(s) & \text{if } \mathfrak{s} = s \in \mathcal{S} \\ \langle t, \pi^*(t) \rangle & \text{otherwise} \end{cases}$$

As per Lemma C.1, all states of  $T$  have the same goal probability. Since the transition in  $\mathfrak{Q}^T$  for  $\langle t, \pi^*(t) \rangle$  is exactly the same as for  $t$  and  $\pi^*(t)$  in  $\mathcal{M}$ , modulo self-loops, it follows that  $V^{\pi^T}(\mathfrak{t}) = V^{\pi^*}(t)$ . In conclusion, it holds for all  $s \in \mathcal{S}$  that  $V^*(s) = V^{\pi^*}(s) = V^{\pi^T}(s^T(s)) \leq V^*[\mathfrak{Q}^T](s^T(s))$ .

The opposite direction, “ $\geq$ ”, is again trivially satisfied if  $V^*[\mathfrak{Q}^T](s^T) = 0$ . For the remaining case, let  $\pi^T$  be a policy for  $\mathfrak{Q}^T$  that is optimal in all states  $\mathfrak{s} \in \mathcal{S}^T$ . We construct a policy  $\pi$  for  $\mathcal{M}$  such that  $V^\pi(s) = V^{\pi^T}(s^T(s))$  holds for all states  $s \in \mathcal{S}$ . For all  $s \notin T$ , let  $\pi(s) := \pi^T(s)$ . Since  $V^{\pi^T}(\mathfrak{t}) > 0$ , and  $\mathfrak{t}$  cannot be a goal state in the quotient system, there must be some  $t \in T$  and  $a \in \mathcal{A}(t)$  such that  $\pi^T(\mathfrak{t}) = \langle t, a \rangle$ . Define  $\pi(t) := a$ . To define the policy on the remaining states in  $T$ , we follow a procedure similar to Algorithm 11.2. We start with  $\text{processed} = \{t\}$ . If  $\text{processed} \neq T$ , then by the definition of end components, there must exist some state  $t' \in T \setminus \text{processed}$ , and some action  $a' \in \mathcal{A}(t')$  such that  $\text{Succ}[\mathcal{M}](t', a') \subseteq T$  and  $\text{Succ}[\mathcal{M}](t', a') \cap \text{processed} \neq \emptyset$ . Set  $\pi(t') := a'$ ,  $\text{processed} := \text{processed} \cup \{t'\}$ , and repeat. Eventually, all states in  $T$  must have been processed. Then, by construction,  $\pi$  achieves the same goal probability for all states in  $T$ , i.e., for all  $t' \in T$ :  $V^\pi(t') = V^{\pi^T}(t') = V^{\pi^T}(\mathfrak{t})$ . In conclusion, it holds for all states  $s \in \mathcal{S}$  that  $V^*(s) \geq V^\pi(s) = V^{\pi^T}(s^T(s)) = V^*[\mathfrak{Q}^T](s^T(s))$ . □

### C.2.3. Correctness of FRET-V (Theorem 12.8)

**Theorem 12.8.** Suppose that  $V(s) = (BV)(s)$  for all states  $s \in \mathcal{R}^V(s_I)$ . If  $\mathcal{M}^V$  contains no reachable permanent trap, then  $V(s) = V^*(s)$  holds for states  $s \in \mathcal{R}^V(s_I)$ .

*Proof.* We show that for every state  $s \in \mathcal{R}^V(s_I)$  with  $V(s) > 0$ , there exists a path in  $\mathcal{M}^V$  from  $s$  to some goal state. Then, since all paths in  $\mathcal{M}^V$  are over actions greedy on  $V$ , applying Algorithm 11.2 on  $V$  yields a policy  $\pi_V$  greedy on  $V$  such that  $V^{\pi_V}(s) \geq V(s)$  is satisfied for all  $s \in \mathcal{R}^V(s_I)$ , as per the proof arguments of Theorem 11.6: In summary,  $\pi_V$  induces an SSP  $\mathcal{M}^{\pi_V}$  since, by construction, all policy execution runs from any state  $s$  where  $\pi_V(s) \neq \perp$  reach an absorbing state eventually; hence  $V^{\pi_V}$  is the unique fixed point of  $B$  in  $\mathcal{M}^{\pi_V}$ ; and it holds that  $V(s) = (BV)(s) = (B^{\pi_V}V)(s)$  since  $\pi_V$  is greedy on  $V$  for all states; concluding that  $V^{\pi_V} = V$ , as desired. The claim follows with  $V^*(s) \geq V^{\pi_V}(s)$ , and  $V(s) \geq V^*(s)$ , where the latter is true because  $V^*$  is the piecewise smallest fixed point.

To show that such paths indeed exist, let  $S_1, \dots, S_n$  be the maximal SCCs of  $\mathcal{M}^V$  reachable from  $s_I$ , in a topological order, i.e., such that for all  $1 \leq i < j \leq n$ , there is no transition from any state in  $S_j$  to any state in  $S_i$ . Suppose for contradiction that there is a state  $s \in \mathcal{R}^V(s_I)$  with  $V(s) > 0$ , yet  $s$  does not

start any goal-path in  $\mathcal{M}^V$ . Let  $i$  be the index of the SCC where  $s \in S_i$ . Without loss of generality, assume that for all  $i < j \leq n$ ,  $S_j$  does not contain such a state. Suppose all states in  $S_i$  have the same value. By assumption,  $S_i$  cannot contain a goal state. Since the states in  $S_i$  have non-zero  $V$ -value, and  $\mathcal{M}^V$  contains no reachable permanent traps, there must hence exist a state  $s' \in S_i$ , and an action  $a$  that is greedy on  $V$  for  $s'$  such that  $\text{Succ}[\mathcal{M}](s', a) \not\subseteq S_i$ . Let  $t \in \text{Succ}[\mathcal{M}](s', a) \setminus S_i$  be arbitrary. Note that  $t$  must appear in some SCC  $S_j$  with  $j > i$ . Since  $s$  cannot reach a goal state in  $\mathcal{M}^V$ ,  $s'$  cannot either. So,  $t$  cannot reach a goal state, and with  $j > i$ , it follows that  $V(t) = 0$ . This leads to a contradiction to fixed point assumption:

$$\begin{aligned}
 V(s') &= (BV)(s') = (QV)(s', a) \\
 &= \sum_{t \in S_i} \mathcal{P}(s', a, t) V(s') + \sum_{t \notin S_i} \mathcal{P}(s', a, t) V(t) \\
 &< V(s') + \sum_{t \notin S_i} \mathcal{P}(s', a, t) V(t) \\
 &= V(s')
 \end{aligned}$$

Now, suppose that not all states have the same value. Let  $S_{\max} \subset S_i$  be the subset of states with maximal  $V$ -value among the states of  $S_i$ . Let  $s \in S_{\max}$  be any state, which has an action  $a \in \mathcal{A}(s)$  that is greedy on  $V$  such that  $\text{Succ}[\mathcal{M}](s, a) \not\subseteq S_{\max}$ . Given that  $S_i$  is an SCC, such a state and an action must exist. Regardless of whether or not  $\text{Succ}[\mathcal{M}](s, a) \subseteq S_i$  is satisfied, with the same reasons as above,  $s$  transitions via a greedy action into a state with strictly smaller  $V$ -value, while the values of all other successors are no larger than  $V(s)$ . This contradicts once again the fixed point assumption. □

#### C.2.4. Comparison Between FRET- $V$ and FRET- $\pi$ (Theorem 12.12)

**Theorem 12.12.** *There exist parameterized families of MDPs  $\mathcal{M}_n$ , and monotone upper bounds  $H_n$  where*

- I) *FRET- $\pi$  is guaranteed to find an optimal policy while considering only states polynomial in  $n$ , whereas FRET- $V$  must consider exponentially many states in  $n$ .*
- II) *FRET- $V$  terminates after a polynomial number of iterations in  $n$  (after eliminating a polynomial number of traps in  $n$ ), whereas FRET- $\pi$  needs an exponential number of iterations in  $n$  (needs to eliminate exponentially many traps in  $n$ ).*
- III) *FRET- $V$  requires an exponential number of iterations in  $n$  (needs to eliminate an exponential number of traps in  $n$ ), whereas FRET- $\pi$  has the potential to terminate after just a polynomial number of iterations in  $n$  (after eliminating a polynomial number of traps in  $n$ ).*

*Proof.*

- I) Consider the following family of probabilistic FDR tasks:  $\Pi_n = \langle \mathcal{V}_n, \mathcal{A}_n, s_{I_n}, \mathcal{G}_n \rangle$  with

- Binary variables  $\mathcal{V}_n = \{B_1, B_2, \dots, B_n\}$ , with domains  $\mathcal{D}_{B_i} = \{0, 1\}$  for all  $1 \leq i \leq n$ .
- Actions  $\mathcal{A}_n = \{set_1, set_2, \dots, set_n\}$  with precondition  $\text{pre}(set_i) = \{B_i \mapsto 0\}$  and just a single outcome  $o_i$  with  $\text{eff}(o_i) = \{B_i \mapsto 1\}$ , for each  $i$ .
- Initial state  $s_{I_n} = \{B_1 \mapsto 0, \dots, B_n \mapsto 0\}$ .
- Goal  $\mathcal{G}_n = \{B_1 \mapsto 1, \dots, B_n \mapsto 1\}$ .

Note that  $V^*(s) = 1$  for all states, and that every closed policy is optimal. Starting from  $V^U = 1 = V^*$ , FIND-AND-REVISE will expand just a single closed policy greedy on  $V^*$  until termination, which is optimal as per the previous observation. Hence,  $n^2$  is an upper bound on the states touched by FIND-AND-REVISE (each policy needs  $n$  steps to reach the goal state, and each visited state along this path has no more than  $n$  successors). Since the MDP is acyclic, there are no traps, and FRET- $\pi$  immediately terminates. In contrast, FRET- $V$  builds the  $V^*$ -greedy graph to check termination, which encompasses all  $2^n$  states.

II) Consider the family of probabilistic FDR tasks  $\Pi_n = \langle \mathcal{V}_n, \mathcal{A}_n, s_{I_n}, \mathcal{G}_n \rangle$ , modeling a binary counter:

- Variables  $\mathcal{V}_n = \{X, B_1, B_2, \dots, B_n\}$ , with domains  $\mathcal{D}_{B_i} = \{0, 1\}$  for all  $1 \leq i \leq n$ , and  $\mathcal{D}_X = \{0, 1, 2\}$ .
- Actions  $\mathcal{A}_n = \{set_1, set_2, \dots, set_n\} \cup \{unset_1, unset_2, \dots, unset_n\} \cup \{end\}$ , where
  - $set_i$  has precondition  $\text{pre}(set_i) = \{X \mapsto 0, B_1 \mapsto 1, \dots, B_{i-1} \mapsto 1, B_i \mapsto 0\}$ , and just a single outcome  $o$  with  $\text{eff}(o) = \{B_1 \mapsto 0, \dots, B_{i-1} \mapsto 0, B_i \mapsto 1\}$ .
  - Vice versa,  $unset_i$  has precondition  $\text{pre}(unset_i) = \{X \mapsto 0, B_1 \mapsto 0, \dots, B_{i-1} \mapsto 0, B_i \mapsto 1\}$ , and just a single outcome  $o$  with  $\text{eff}(o) = \{B_1 \mapsto 1, \dots, B_{i-1} \mapsto 1, B_i \mapsto 0\}$ .
  - $end$  has precondition  $\text{pre}(end) = \{B_1 \mapsto 1, \dots, B_n \mapsto 1, X \mapsto 0\}$ , and two outcomes  $\text{eff}(o_1) = \{X \mapsto 1\}$  and  $\text{prob}(o_1) = \frac{1}{2}$ ; and  $\text{eff}(o_2) = \{X \mapsto 2\}$  and  $\text{prob}(o_2) = \frac{1}{2}$ .
- Initial state  $s_{I_n} = \{X \mapsto 0, B_1 \mapsto 0, \dots, B_n \mapsto 0\}$ .
- Goal  $\mathcal{G}_n = \{X \mapsto 1\}$ .

Note that the state space induced by  $\Pi_n$  is simply a chain of  $2^n$  states corresponding to the numbers  $0 - 2^{n-1}$  (according to the binary representation as per the  $B_i$  bits), where each pair of states  $k$  and  $k + 1$  are bi-connected via corresponding  $set$  and  $unset$  actions. The chain is followed by a probabilistic transition via  $end$  into a goal state  $s_*$  and a dead end  $s_\perp$ . The optimal goal probabilities are  $V^*(s) = \frac{1}{2}$  for all the “number states”,  $V^*(s_*) = 1$ , and  $V^*(s_\perp) = 0$ . Starting from the upper bound  $V^U(s) = 0$  if  $s = s_\perp$  and  $V^U(s) = 1$  otherwise, FIND-AND-REVISE terminates immediately.  $V^U$  satisfies the Bellman equations for all number states.  $end$  is not greedy on  $V^U$ , i.e., the  $V^U$ -greedy graph contains only the number states. Since the  $V^U$ -greedy graph contains both  $set$  and  $unset$  transitions for each of those states, the entire graph is one big end component. FRET- $V$  hence collapses all  $2^n$  states into a single quotient state, after which FIND-AND-REVISE will terminate with the optimal solution. In contrast, FRET- $\pi$  requires  $2^{n-1}$  iterations, since every possible greedy policy can only use  $unset$  in only a single state that is reached by following the policy. It must include an  $unset$  action, because  $end$  becomes greedy on  $V^U$  only after having removed all cycles. No matter on which state  $unset$  is applied, the policy graph contains only a single trap, and this trap consists of exactly 2 states. Hence, FRET- $\pi$  requires  $2^{n-1}$  iterations, eliminating the same number of traps, until the entire chain has been collapsed into a single quotient state.

III) Consider a modified version of the binary counter example from above, where everything stays the same but

- $\mathcal{A}_n = \{set_1, set_2, \dots, set_n\} \cup \{unset_1, unset_3, \dots, unset_n\} \cup \{start, end\}$ , i.e.,  $unset$  is no longer possible for the second bit.
  - $set_i$  and  $unset_i$  are defined as above.

- *start* has precondition  $\text{pre}(\text{start}) = \{X \mapsto 2\}$ , and just a single outcome  $\text{eff}(o) = \{X \mapsto 0\}$ .
- *end* has precondition  $\text{pre}(\text{end}) = \{X \mapsto 2\}$ , and just a single outcome  $\text{eff}(o) = \{X \mapsto 1\}$ .
- Initial state  $s_{In} = \{X \mapsto 2, B_1 \mapsto 0, \dots, B_n \mapsto 0\}$ , i.e.,  $X$  is initially 2.

As opposed to the previous binary counter example, the state space now starts with a choice at the initial state whether to directly transition into a goal state via *end*, or to start counting via *start*. Similarly as before, the states reached via *start* form a chain of numbers from 0 to  $2^{n-1}$ . Successors in this chain are however no longer completely bi-connected, due to the missing *unset<sub>2</sub>* action. The chain is no longer one big trap. Instead it is now grouped into connected components of exactly 4 states, with the exception of the first two, and the last two states. This makes  $2^{n-2} + 1$  traps in total. The optimal goal probability is  $V^*(s) = 1$  for all states where  $X$  is not 0, and  $V^*(s) = 0$  if  $X$  is 0 (the counter states). Starting from the upper bound  $V^U = 1$ , FIND-AND-REVISE again terminates immediately.  $V^U$  satisfies the Bellman equations for all reachable states, since every counter state is still part of at least one *unset* cycle. After the first call to FIND-AND-REVISE, the  $V^U$ -greedy graph thus includes the entire (reachable) state space. Moreover, *start* remains a greedy action of  $V^U$  thereafter, as long as not every trap has been eliminated. Note that at any point in time, there is only a single trap in the counter chain that is permanent. In other words, FRET- $V$  can eliminate only a single trap per iteration. Hence, FRET- $V$  requires  $2^{n-2} + 1$  iteration until all traps have been collapsed. On the other hand, if we are lucky, and the initial FIND-AND-REVISE call returned the policy  $\pi^U(s_{In}) = \text{end}$ , then FRET- $\pi$  terminates immediately, without eliminating any trap. (Note however that this depends on tie-breaking. In particular, in the worst case, FRET- $\pi$  may also need to eliminate all the  $2^{n-2} + 1$  traps.)

□

### C.3. Goal-Probability Occupation-Measure Heuristics

#### C.3.1. Equations for the $H^{\text{gpm}}$ Monotonicity Proof (Theorem 14.1)

For the sake of simplicity, in the following, we use the projection actions  $a|_X$  and the original actions  $a$  interchangeably. We first rewrite  $\text{out}_X$  and  $\text{in}_X$  individually:

$$\begin{aligned}
 \text{out}_X(s) &= \sum_{a \in \mathcal{A}^X(s)} \text{om}_{X,s,a} \\
 &= \sum_{a \in \mathcal{A}^X(s)} \left( \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) \text{om}_{X,s,a}^t + [\mathbf{s} \subseteq \hat{s} \wedge a = \hat{a}] \right) \quad (\text{def. of } y) \\
 &= \left( \sum_{a \in \mathcal{A}^X(s)} \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) \text{om}_{X,s,a}^t \right) + [\mathbf{s} \subseteq \hat{s}] \quad (a = \hat{a} \text{ is satisfied once}) \\
 &= \left( \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) \sum_{a \in \mathcal{A}^X(s)} \text{om}_{X,s,a}^t \right) + [\mathbf{s} \subseteq \hat{s}] \quad (\text{commutat. and distributivity}) \\
 &= \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) \text{out}_X^t(s) + [\mathbf{s} \subseteq \hat{s}] \quad (\text{def. of } \text{out}_X \text{ for } t)
 \end{aligned} \tag{C.1}$$

$$in_X(s) = \sum_{t \in \mathcal{S}^X} \sum_{a \in \mathcal{A}^X} \mathcal{P}^X(t, a|_X, s) om_{X,t,a} \\ = \sum_{t \in \mathcal{S}^X} \sum_{a \in \mathcal{A}^X} \mathcal{P}^X(t, a|_X, s) \left( \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) om_{X,t,a}^t + [\mathbf{t} \subseteq \hat{s} \wedge a = \hat{a}] \right) \quad (\text{C.2a})$$

$$= \left( \sum_{t \in \mathcal{S}^X} \sum_{a \in \mathcal{A}^X} \mathcal{P}^X(t, a|_X, s) \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) om_{X,t,a}^t \right) + \mathcal{P}^X(\hat{s}|_X, \hat{a}|_X, s) \quad (\text{C.2b})$$

$$= \left( \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) \sum_{t \in \mathcal{S}^X} \sum_{a \in \mathcal{A}^X} \mathcal{P}^X(t, a|_X, s) om_{X,t,a}^t \right) + \mathcal{P}^X(\hat{s}|_X, \hat{a}|_X, s) \quad (\text{C.2c})$$

$$= \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) in_X^t(s) + \mathcal{P}^X(\hat{s}|_X, \hat{a}|_X, s) \quad (\text{C.2d})$$

where (C.2a) uses the definition of  $om$ ; (C.2b) uses the fact that  $[\mathbf{t} \subseteq \hat{s} \wedge a = \hat{a}]$  is satisfied once (when  $\mathbf{t} = \hat{s}|_X$ ); (C.2c) apply the commutativity and distributivity laws; and (C.2d) uses the definition of  $in_X$  for  $t$ .

Putting (C.1) and (C.2) together yields:

$$out_X(s) - in_X(s) \\ = \left( \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) out_X^t(s) + [\mathbf{s} \subseteq \hat{s}] \right) - \left( \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) in_X^t(s) + \mathcal{P}^X(\hat{s}|_X, \hat{a}|_X, s) \right) \quad (\text{C.3}) \\ = \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) (out_X^t(s) - in_X^t(s)) + [\mathbf{s} \subseteq \hat{s}] - \mathcal{P}^X(\hat{s}|_X, \hat{a}|_X, s)$$

Plugging (C.3) into the goal constraint (14.1d) yields:

$$\sum_{s \in \mathcal{S}_*^X} (out_X(s) - in_X(s)) + v_{\mathcal{G}} \\ = \sum_{s \in \mathcal{S}_*^X} \left( \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) (out_X^t(s) - in_X^t(s)) + [\mathbf{s} \subseteq \hat{s}] - \mathcal{P}^X(\hat{s}|_X, \hat{a}|_X, s) \right) + v_{\mathcal{G}} \\ = \sum_{s \in \mathcal{S}_*^X} \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) (out_X^t(s) - in_X^t(s)) + [\mathcal{G}|_X \subseteq \hat{s}] - \sum_{s \in \mathcal{S}_*^X} \mathcal{P}^X(\hat{s}|_X, \hat{a}|_X, s) + v_{\mathcal{G}} \quad (\text{C.4a})$$

$$= \left( \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) \left[ \sum_{s \in \mathcal{S}_*^X} (out_X^t(s) - in_X^t(s)) + v_{\mathcal{G}}^t \right] \right) + [\mathcal{G}|_X \subseteq \hat{s}] - \sum_{s \in \mathcal{S}_*^X} \mathcal{P}^X(\hat{s}|_X, \hat{a}|_X, s) \quad (\text{C.4b})$$

$$\leq \sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) [\mathcal{G}|_X \subseteq t] + [\mathcal{G}|_X \subseteq \hat{s}] - \sum_{s \in \mathcal{S}_*^X} \mathcal{P}^X(\hat{s}|_X, \hat{a}|_X, s) \quad (\text{C.4c})$$

where (C.4a) holds because  $\hat{s}|_X$  is the only state in  $\mathcal{S}^X$ , which satisfies  $\hat{s}|_X \subseteq \hat{s}$ , and  $\hat{s}|_X \in \mathcal{S}_*^X$  iff  $\mathcal{G}|_X \subseteq \hat{s}$  by the definition of the syntactic projection; (C.4b) plugs in the definition of  $v_{\mathcal{G}}$ , and applies the commutativity and distributivity laws; (C.4c) uses the fact that each  $t$  satisfies the (14.1d) constraint.

Similar to flow constraint, the  $[\mathcal{G}|_X \subseteq \hat{s}]$  term cancels out with the right-hand side of the goal constraint

(14.1d). The goal constraint is satisfied, because the following equations hold true:

$$\begin{aligned}
\sum_{t \in \text{Succ}(\hat{s}, \hat{a})} \mathcal{P}(\hat{s}, \hat{a}, t) [\mathcal{G}|_X \subseteq t] &= \sum_{o \in \text{out}(\hat{a})} \text{prob}(o) [\mathcal{G}|_X \subseteq \hat{s}[\![o]\!]] \\
&= \sum_{o \in \text{out}(\hat{a})} \text{prob}(o) [\mathcal{G}|_X \subseteq \hat{s}[\![o]\!]|_X] \\
&= \sum_{o \in \text{out}(\hat{a})} \text{prob}(o) [\mathcal{G}|_X \subseteq \hat{s}[\![o|_X]\!]|_X] \\
&= \sum_{o \in \text{out}(\hat{a})} \text{prob}(o) [\mathcal{G}|_X \subseteq \hat{s}|_X[\![o|_X]\!]] \\
&= \sum_{o \in \text{out}(\hat{a})} \text{prob}(o) [\hat{s}|_X[\![o|_X]\!] \in \mathcal{S}_*^X] \\
&= \sum_{s \in \mathcal{S}_*^X} \mathcal{P}^X(\hat{s}|_X, \hat{a}|_X, s)
\end{aligned}$$

### C.3.2. Dominance Relation Between $H^{\text{gpom}}$ and $H_{\text{seq}}^{\text{gpo}}$ (Theorem 14.4)

**Theorem 14.4.** Let  $\mathcal{X} = \{ \{v\} \mid v \in \mathcal{V} \}$  be the set of singleton variable sets. For every state  $s$ , there is an optimal solution  $om^*, v_{\mathcal{G}}^*$  to the  $H_{\mathcal{X}}^{\text{gpom}}(s)$  LP such that, for an arbitrary  $v_0 \in \mathcal{V}$ ,

$$\begin{aligned}
v_{\mathcal{G}} &= v_{\mathcal{G}}^* \\
y_a &= \sum_{d \in \mathcal{D}_{v_0}} om_{v_0, d, a}^* \quad \text{for all } a \in \mathcal{A}
\end{aligned}$$

satisfies  $\Gamma^{\text{seq}}(s)$ .

*Proof.* Let  $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$  be a probabilistic FDR task, and  $s$  be a state of  $\Pi$ . Suppose  $H_{\mathcal{X}}^{\text{gpom}}$  is the goal-probability occupation-measure heuristic over the atomic projections. Let  $om^*, v_{\mathcal{G}}^*$  be an optimal solution to the  $H_{\mathcal{X}}^{\text{gpom}}(s)$  LP, and let  $y, v_{\mathcal{G}}$  be defined as in the claim. Let  $v \in \mathcal{V}$ , and  $d \in \mathcal{D}_v$  be arbitrary. Consider the following action partitioning:

$$\begin{aligned}
\mathcal{A}_d &= \{ a \in \mathcal{A} \mid \text{pre}(a)[v] = d \} \\
\mathcal{A}_{\neg d} &= \{ a \in \mathcal{A} \mid \text{pre}(a)[v] = d' \neq d \} \\
\mathcal{A}_{\perp} &= \{ a \in \mathcal{A} \mid \text{pre}(a)[v] = \perp \}
\end{aligned}$$

Similarly, we partition the outcomes of each action  $a \in \mathcal{A}$ :

$$\begin{aligned}
\text{out}_d(a) &= \{ o \in \text{out}(a) \mid \text{eff}(o)[v] = d \} \\
\text{out}_{\neg d}(a) &= \{ o \in \text{out}(a) \mid \text{eff}(o)[v] = d' \neq d \} \\
\text{out}_{\perp}(a) &= \{ o \in \text{out}(a) \mid \text{eff}(o)[v] = \perp \}
\end{aligned}$$

We denote the respective summed up outcome probabilities by  $\rho_d^a = \sum_{o \in \text{out}_d(a)} \text{prob}(o)$ , and analogously,  $\rho_{\neg d}^a$  and  $\rho_{\perp}^a$ . Note that, as per the definition of the syntactic projection, it holds for all  $d' \in \mathcal{D}_v$  and  $a \in \mathcal{A}$  that

$$\mathcal{P}^v(d', a|_X, d) = \begin{cases} 0 & \text{if } \text{pre}(a)[v] = d'' \neq d' \\ \rho_d^a + \rho_{\perp}^a & \text{if } d' = d \\ \rho_d^a & \text{otherwise} \end{cases}$$

Further notice that, for all actions  $a \in \mathcal{A}$ , where  $\text{pre}(a)[v]$  is defined, there is just a single occupation-measure variable for  $v$  and  $a$ , which is  $y_a = \text{om}_{v, \text{pre}(a)[v], a}$ . Plugging these observation into the definition of  $\text{in}_v(d)$  gives:

$$\begin{aligned}
 \text{in}_v(d) &= \sum_{a \in \mathcal{A}} \sum_{d' \in \mathcal{D}_v} \mathcal{P}^v(d', a|_X, d) \text{om}_{v, d', a} \\
 &= \sum_{a \in \mathcal{A}_d} (\rho_d^a + \rho_\perp^a) \text{om}_{v, d, a} + \sum_{a \in \mathcal{A}_{-d}} \rho_d^a \text{om}_{v, \text{pre}(a)[v], a} + \sum_{a \in \mathcal{A}_\perp} \sum_{d' \in \mathcal{D}_v} \mathcal{P}^v(d', a|_X, d) \text{om}_{v, d', a} \\
 &= \sum_{a \in \mathcal{A}_d} (\rho_d^a + \rho_\perp^a) y_a + \sum_{a \in \mathcal{A}_{-d}} \rho_d^a y_a + \sum_{a \in \mathcal{A}_\perp} \left( \sum_{d' \in \mathcal{D}_v} \rho_d^a \text{om}_{v, d', a} + \rho_\perp^a \text{om}_{v, d, a} \right) \\
 &= \sum_{a \in \mathcal{A}_d} (\rho_d^a + \rho_\perp^a) y_a + \sum_{a \in \mathcal{A}_{-d}} \rho_d^a y_a + \sum_{a \in \mathcal{A}_\perp} \rho_d^a \sum_{d' \in \mathcal{D}_v} \text{om}_{v, d', a} + \sum_{a \in \mathcal{A}_\perp} \rho_\perp^a \text{om}_{v, d, a} \\
 &= \sum_{a \in \mathcal{A}_d} (\rho_d^a + \rho_\perp^a) y_a + \sum_{a \in \mathcal{A}_{-d}} \rho_d^a y_a + \sum_{a \in \mathcal{A}_\perp} \rho_d^a y_a + \sum_{a \in \mathcal{A}_\perp} \rho_\perp^a \text{om}_{v, d, a}
 \end{aligned}$$

For the actions  $a \in \mathcal{A}_{-d}$ ,  $a|_v$  is by definition not applicable in  $v \mapsto d$ . Moreover, the three  $\rho$  probability weights cover all outcomes, i.e.,  $\rho_d^a + \rho_{-d}^a + \rho_\perp^a = 1$ , for all  $a \in \mathcal{A}$ . Therefore,

$$\begin{aligned}
 \text{out}_v(d) &= \sum_{a|_X \in \mathcal{A}^v(d)} \text{om}_{v, d, a} = \sum_{a \in \mathcal{A}_d} \text{om}_{v, d, a} + \sum_{a \in \mathcal{A}_\perp} \text{om}_{v, d, a} \\
 &= \sum_{a \in \mathcal{A}_d} y_a + \sum_{a \in \mathcal{A}_\perp} \text{om}_{v, d, a} \\
 &= \sum_{a \in \mathcal{A}_d} (\rho_d^a + \rho_{-d}^a + \rho_\perp^a) y_a + \sum_{a \in \mathcal{A}_\perp} (\rho_d^a + \rho_{-d}^a + \rho_\perp^a) \text{om}_{v, d, a}
 \end{aligned}$$

For all  $a \in \mathcal{A}_d$ , it holds by definition that  $\text{prod}_{v \mapsto d}^a = 0$ . Likewise, for all  $a \notin \mathcal{A}_d$ , it holds that  $\text{cons}_{v \mapsto d}^a = 0$ . In combination with the above equations, this results in

$$\begin{aligned}
 \text{out}_v(d) - \text{in}_v(d) &= \sum_{a \in \mathcal{A}_d} \rho_{-d}^a y_a - \sum_{a \in \mathcal{A}_{-d}} \rho_d^a y_a - \sum_{a \in \mathcal{A}_\perp} \rho_d^a y_a + \sum_{a \in \mathcal{A}_\perp} (\rho_d^a + \rho_{-d}^a) \text{om}_{v, d, a} \\
 &= \sum_{a \in \mathcal{A}} \text{cons}_{v \mapsto d}^a y_a - \sum_{a \in \mathcal{A}} \text{prod}_{v \mapsto d}^a y_a + \sum_{a \in \mathcal{A}_\perp} (\rho_d^a + \rho_{-d}^a) \text{om}_{v, d, a} \\
 &\geq \sum_{a \in \mathcal{A}} (\text{cons}_{v \mapsto d}^a - \text{prod}_{v \mapsto d}^a) y_a
 \end{aligned}$$

If  $\mathcal{G}[v] = d$ , then via (14.1d):

$$\begin{aligned}
 [\mathcal{G}|_v \subseteq s] &= [s[v] = d] \\
 &\geq \text{out}_v(d) - \text{in}_v(d) + v_{\mathcal{G}} \\
 &\geq \sum_{a \in \mathcal{A}} (\text{cons}_{v \mapsto d}^a - \text{prod}_{v \mapsto d}^a) y_a + v_{\mathcal{G}}
 \end{aligned}$$

i.e., (14.3) is satisfied. Similarly, if  $\mathcal{G}[v] \neq d$ , then via (14.1c)

$$[s[v] = d] \geq \text{out}_v(d) - \text{in}_v(d) \geq \sum_{a \in \mathcal{A}} (\text{cons}_{v \mapsto d}^a - \text{prod}_{v \mapsto d}^a) y_a$$

We conclude that  $y, v_{\mathcal{G}}$  satisfy (14.3) for all  $v$  and  $d$ .  $\square$

### C.4. Goal-Probability Fact Potential Heuristics

Let  $w: \mathcal{F}^\Pi \rightarrow \mathbb{R}$  be a function assigning real-valued weights to facts. The fact potential heuristic induced by  $w$  is given by  $h_{\mathcal{F}^\Pi, w}^{\text{pot}}(s) = \sum_{v \mapsto d \in s} w_{v \mapsto d}$  (cf. Definition 7.5). Now, consider the dual of the goal-probability state-equation LP (cf. Definition 14.5):

$$\underset{w}{\text{minimize}} \quad \sum_{v \in \mathcal{V}} w_{v \mapsto \perp} [v] \quad (\text{C.5a})$$

$$\text{subject to} \quad w_{v \mapsto d} \geq 0 \quad v \in \mathcal{V}, d \in \mathcal{D}_v, \quad (\text{C.5b})$$

$$\sum_{v \in \text{vars}(\mathcal{G})} w_{v \mapsto \mathcal{G}[v]} \geq 1, \quad (\text{C.5c})$$

$$\sum_{v, d} (\text{cons}_{v \mapsto d}^a - \text{prod}_{v \mapsto d}^a) w_{v \mapsto d} \geq 0 \quad a \in \mathcal{A} \quad (\text{C.5d})$$

Suppose  $w$  is a feasible solution of this LP. Obviously, the constraints (C.5b) and (C.5c) ensure that  $h_{\mathcal{F}^\Pi, w}^{\text{pot}}(s_*) \geq 1$ , for all goal states. Furthermore, by plugging in the definition of the transition-probability function, it is straightforward to show that the following relation is satisfied (cf. below):

$$h_{\mathcal{F}^\Pi, w}^{\text{pot}}(s) - \sum_t \mathcal{P}^\Pi(s, a, t) h_{\mathcal{F}^\Pi, w}^{\text{pot}}(t) \geq \sum_{v, d} (\text{cons}_{v \mapsto d}^a - \text{prod}_{v \mapsto d}^a) w_{v \mapsto d} \quad (\text{C.6})$$

From (C.5d) it then follows that  $h_{\mathcal{F}^\Pi, w}^{\text{pot}} \geq B h_{\mathcal{F}^\Pi, w}^{\text{pot}}$ , which, as per the argument provided in the proof of Theorem 14.1, is sufficient to conclude that  $h_{\mathcal{F}^\Pi, w}^{\text{pot}}$  indeed constitutes a monotone goal-probability upper bound. With additional constraints similar to those provided by Pommerening et al. (2015) for non-TNF tasks, it is furthermore possible to drop the domain restriction (C.5b), allowing weights to be negative, and by that establishing the strict equivalence between feasible LP solutions and monotone upper-bounding fact potential heuristics.

It is left to show that (C.6) is satisfied. To do so, we use the following shorthand:

$$\text{sometimesCons}_{v \mapsto d}^a = \begin{cases} \sum_{\substack{o \in \text{out}(a): \\ \text{eff}(o)[v] = d' \neq d}} \text{prob}(o), & \text{if } \text{pre}(a)[v] = \perp \\ 0, & \text{otherwise} \end{cases}$$

Let  $s$  be an arbitrary state, and  $a \in \mathcal{A}(s)$  be any applicable action. We omit the  $\mathcal{F}^\Pi, w$  subscript in the

following.

$$\begin{aligned} h^{\text{pot}}(s) &= \sum_t \mathcal{P}^\Pi(s, a, t) h^{\text{pot}}(t) \\ &= h^{\text{pot}}(s) - \sum_{o \in \text{out}(a)} \text{prob}(o) h^{\text{pot}}(s \llbracket o \rrbracket) \end{aligned} \quad (\text{C.7a})$$

$$= \sum_{v \in \mathcal{V}} w_{v \mapsto s[v]} - \sum_{o \in \text{out}(a)} \text{prob}(o) \left( \sum_{v \in \mathcal{V}} w_{v \mapsto s[v]} - \sum_{v \in \text{vars}(\text{eff}(o))} w_{v \mapsto s[v]} + \sum_{v \in \text{vars}(\text{eff}(o))} w_{v \mapsto \text{eff}(o)[v]} \right) \quad (\text{C.7b})$$

$$= \sum_{o \in \text{out}(a)} \text{prob}(o) \left( \sum_{v \in \text{vars}(\text{eff}(o))} w_{v \mapsto s[v]} - \sum_{v \in \text{vars}(\text{eff}(o))} w_{v \mapsto \text{eff}(o)[v]} \right) \quad (\text{C.7c})$$

$$= \sum_{o \in \text{out}(a)} \text{prob}(o) \left( \sum_{\substack{v \in \text{vars}(\text{eff}(o)) \\ \text{eff}(o)[v] \neq s[v]}} w_{v \mapsto s[v]} - \sum_{\substack{v \in \text{vars}(\text{eff}(o)) \\ \text{eff}(o)[v] \neq s[v]}} w_{v \mapsto \text{eff}(o)[v]} \right) \quad (\text{C.7d})$$

$$\begin{aligned} &= \sum_{o \in \text{out}(a)} \text{prob}(o) \sum_{\substack{v \in \text{vars}(\text{eff}(o)) \\ \text{eff}(o)[v] \neq s[v]}} w_{v \mapsto s[v]} - \sum_{o \in \text{out}(a)} \text{prob}(o) \sum_{\substack{v \in \text{vars}(\text{eff}(o)) \\ \text{eff}(o)[v] \neq s[v]}} w_{v \mapsto \text{eff}(o)[v]} \\ &= \sum_{v \in \mathcal{V}} \left( \text{cons}_{v \mapsto s[v]}^a + \text{sometimesCons}_{v \mapsto s[v]}^a \right) w_{v \mapsto s[v]} - \sum_{v \in \mathcal{V}} \sum_{\substack{d \in \mathcal{D}_v, \\ d \neq s[v]}} \text{prod}_{v \mapsto d}^a w_{v \mapsto d} \end{aligned} \quad (\text{C.7e})$$

$$= \sum_{v, d} \text{cons}_{v \mapsto d}^a w_{v \mapsto d} + \sum_{v \in \mathcal{V}} \text{sometimesCons}_{v \mapsto s[v]}^a w_{v \mapsto s[v]} - \sum_{v \in \mathcal{V}} \sum_{\substack{d \in \mathcal{D}_v, \\ d \neq s[v]}} \text{prod}_{v \mapsto d}^a w_{v \mapsto d} \quad (\text{C.7f})$$

$$\geq \sum_{v, d} \text{cons}_{v \mapsto d}^a w_{v \mapsto d} - \sum_{v \in \mathcal{V}} \sum_{\substack{d \in \mathcal{D}_v, \\ d \neq s[v]}} \text{prod}_{v \mapsto d}^a w_{v \mapsto d} \quad (\text{C.7g})$$

$$\geq \sum_{v, d} \text{cons}_{v \mapsto d}^a w_{v \mapsto d} - \sum_{v, d} \text{prod}_{v \mapsto d}^a w_{v \mapsto d} \quad (\text{C.7h})$$

where (C.7a) uses the state space definition; (C.7b) plugs in the definition of potential heuristics; (C.7c) holds because the outcome probabilities sum up to 1 by assumption; (C.7d) effects  $\text{eff}(o)[v] = s[v]$  cancel out; (C.7e) uses the consumption/production probability weight definitions; (C.7f) uses the fact that  $\text{cons}_{v \mapsto d}^a = 0$  if  $s[v] \neq d$ ; (C.7g) and (C.7h) hold because of the domain restriction  $w_{v \mapsto d} \geq 0$ .

From (C.5d), it then follows that  $h^{\text{pot}}(s) \geq \sum_t \mathcal{P}^\Pi(s, a, t) h^{\text{pot}}(t)$ . From (C.5b) and (C.5c), it follows that  $h^{\text{pot}}(s_*) \geq 1$ , for all goal states. This suffices as per the arguments of the proof of Theorem 14.1 to conclude that  $h^{\text{pot}}$  is a monotone goal-probability upper bound.

## C.5. Goal-Probability Occupation-Measure Heuristics vs. Multiplicative Goal-Probability PDBs

To see that goal-probability occupation-measure heuristics are generally incomparable to multiplicative goal-probability PDB heuristics, consider the following two simple probabilistic planning tasks. Both tasks share the same variables  $x$  and  $y$ , have initial state  $\mathcal{I} = \{x \mapsto 0, y \mapsto 0\}$ , and goal  $\mathcal{G} = \{x \mapsto 1, y \mapsto 1\}$ . They differ in their actions:

- $\mathcal{A}^1 = \{a_x^1, a_y^1\}$ . Action  $a_x^1$  has precondition  $\{x \mapsto 0\}$ , and two outcomes with effects  $\{x \mapsto 1\}$  and  $\{x \mapsto 2\}$  and a probability of  $\frac{1}{2}$  respectively.  $a_y^1$  is defined analogously.

- $\mathcal{A}^2 = \{a_x^2, a_y^2\}$ . Action  $a_x^2$  has precondition  $\{x \mapsto 0, y \mapsto 0\}$  and a single outcome with effect  $\{x \mapsto 1, y \mapsto 2\}$ .  $a_y^2$  is defined symmetrically.

Consider the atomic projections, i.e., the projections onto  $X_1 = \{x\}$  and  $X_2 = \{y\}$ . Without spelling out all details, in the first task, both projection heuristics  $H_{X_1}$  and  $H_{X_2}$  are *multiplicative*, simply because none of the actions affects both variables. They combine into the goal-probability bound  $H_{X_1}(s_I) \cdot H_{X_2}(s_I) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4} = V^*(s_I)$ . On the contrary, the goal-probability projection occupation-measure heuristics yields  $H_{\{X_1, X_2\}}^{\text{gpm}}(s_I) = \frac{1}{2}$ . In the second task,  $H_{X_1}$  and  $H_{X_2}$  are no longer multiplicative. They combine into  $\min\{H_{X_1}(s_I), H_{X_2}(s_I)\} = \min\{1, 1\} = 1$ , whereas  $H_{\{X_1, X_2\}}^{\text{gpm}}(s_I) = \frac{1}{2}$ .

## Bibliography

- Abbeel, P., D. Dolgov, A.Y. Ng, and S. Thrun (2008). “Apprenticeship learning for motion planning with application to parking lot navigation.” In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 1083–1090.
- Aberdeen, Douglas, Sylvie Thiébaux, and Lin Zhang (2004). “Decision-Theoretic Military Operations Planning.” In: *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling ICAPS 2004*. AAAI, pp. 402–412.
- Altman, Eitan (1996). “Constrained Markov decision processes with total cost criteria: Occupation measures and primal LP.” In: *Mathematical Methods of Operations Research* 43.1, pp. 45–72.
- Altman, Eitan (1999). *Constrained Markov Decision Processes*. CRC Press.
- Audemard, Gilles and Laurent Simon (2009). “Predicting Learnt Clauses Quality in Modern SAT Solvers.” In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, pp. 399–404.
- Backes, Paul G., Gregg Rabideau, Kam S. Tso, and Steve A. Chien (1999). “Automated Planning and Scheduling for Planetary Rover Distributed Operations.” In: *1999 IEEE International Conference on Robotics and Automation, ICRA 1999*. IEEE Robotics and Automation Society, pp. 984–991.
- Bäckström, Christer and Bernhard Nebel (1995). “Complexity Results for SAS+ Planning.” In: *Computational Intelligence* 11, pp. 625–656.
- Bäckström, Christer, Peter Jonsson, and Simon Ståhlberg (2013). “Fast Detection of Unsolvable Planning Instances Using Local Consistency.” In: *Proceedings of the Sixth Annual Symposium on Combinatorial Search, SOCS 2013*. AAAI Press.
- Baier, Christel and Marta Z. Kwiatkowska (1998). “Model Checking for a Probabilistic Branching Time Logic with Fairness.” In: *Distributed Computing* 11.3, pp. 125–155.
- Balyo, Tomás and Martin Suda (2016). “Reachlunch Entering The Unsolvability IPC 2016.” In: *UIPC 2016 planner abstracts*, pp. 3–5.
- Barto, Andrew G., Steven J. Bradtke, and Satinder P. Singh (1995). “Learning to Act Using Real-Time Dynamic Programming.” In: *Artificial Intelligence* 72.1-2, pp. 81–138.
- Bayardo, Roberto J. and Robert Schrag (1996). “Using CSP Look-Back Techniques to Solve Exceptionally Hard SAT Instances.” In: *Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming, CP 1996*. Springer-Verlag, pp. 46–60.
- Beame, Paul, Henry A. Kautz, and Ashish Sabharwal (2004). “Towards Understanding and Harnessing the Potential of Clause Learning.” In: *Journal of Artificial Intelligence Research* 22, pp. 319–351.
- Beetz, Michael (2002). *Plan-Based Control of Robotic Agents: Improving the Capabilities of Autonomous Robots*. Vol. 2554. Lecture Notes in Computer Science. Springer.
- Bellman, Richard (1957). *Dynamic Programming*. 1st ed. Princeton University Press.
- Bertsekas, Dimitri (1995). *Dynamic programming and optimal control*. Athena Scientific.
- Bertsekas, Dimitri P. and John N. Tsitsiklis (1996). *Neuro-dynamic programming*. Vol. 3. Optimization and neural computation series. Athena Scientific.
- Bhatnagar, Neeraj and Jack Mostow (1994). “On-Line Learning from Search Failures.” In: *Machine Learning* 15.1, pp. 69–117.

- Bianco, Andrea and Luca de Alfaro (1995). “Model Checking of Probabilistic and Nondeterministic Systems.” In: *Foundations of Software Technology and Theoretical Computer Science, 15th Conference, Proceedings*. Vol. 1026. Lecture Notes in Computer Science. Springer, pp. 499–513.
- Biere, Armin, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu (1999). “Symbolic Model Checking without BDDs.” In: *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, Proceedings, TACAS 1999*. Vol. 1579. Lecture Notes in Computer Science. Springer, pp. 193–207.
- Blum, Avrim and Merrick L. Furst (1997). “Fast Planning Through Planning Graph Analysis.” In: *Artificial Intelligence* 90.1-2, pp. 281–300.
- Bonet, Blai (2013). “An Admissible Heuristic for SAS+ Planning Obtained from the State Equation.” In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*. IJCAI/AAAI, pp. 2268–2274.
- Bonet, Blai and Hector Geffner (2001). “Planning as heuristic search.” In: *Artificial Intelligence* 129.1-2, pp. 5–33.
- Bonet, Blai and Hector Geffner (2003a). “Faster Heuristic Search Algorithms for Planning with Uncertainty and Full Feedback.” In: *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, pp. 1233–1238.
- Bonet, Blai and Hector Geffner (2003b). “Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming.” In: *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003)*. AAAI, pp. 12–21.
- Bonet, Blai and Hector Geffner (2005). “mGPT: A Probabilistic Planner Based on Heuristic Search.” In: *Journal of Artificial Intelligence Research* 24, pp. 933–944.
- Bonet, Blai and Hector Geffner (2006). “Learning Depth-First Search: A Unified Approach to Heuristic Search in Deterministic and Non-Deterministic Settings, and Its Application to MDPs.” In: *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006*. AAAI, pp. 142–151.
- Bonet, Blai and Menkes van den Briel (2014). “Flow-Based Heuristics for Optimal Planning: Landmarks and Merges.” In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014*. AAAI, pp. 47–55.
- Borgwardt, Stefan, Jörg Hoffmann, Alisa Kovtunova, and Marcel Steinmetz (2021). “Making DL-Lite Planning Practical.” In: *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021*, pp. 641–645.
- Borgwardt, Stefan, Jörg Hoffmann, Alisa Kovtunova, Markus Krötzsch, Bernhard Nebel, and Marcel Steinmetz (2022). “Expressivity of Planning with Horn Description Logic Ontologies.” In: *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022*. AAAI Press, pp. 5503–5511.
- Bradley, Aaron R. (2011). “SAT-Based Model Checking without Unrolling.” In: *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011*. Vol. 6538. Lecture Notes in Computer Science. Springer, pp. 70–87.
- Brázdil, Tomáš, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Kretínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma (2014). “Verification of Markov Decision Processes Using Learning Algorithms.” In: *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014*. Vol. 8837. Lecture Notes in Computer Science. Springer, pp. 98–114.
- Brechtel, Sebastian, Tobias Gindele, and Rüdiger Dillmann (2011). “Probabilistic MDP-behavior planning for cars.” In: *14th International IEEE Conference on Intelligent Transportation Systems, ITSC 2011*. IEEE, pp. 1537–1542.

- Bruynooghe, Maurice and Luís Moniz Pereira (1984). “Deduction Revision by Intelligent Backtracking.” In: *Implementations of Prolog*. Ellis Horwood/Halsted Press/Wiley, pp. 194–215.
- Bryant, Randal E. (1986). “Graph-Based Algorithms for Boolean Function Manipulation.” In: *IEEE Transactions on Computers* 35.8, pp. 677–691.
- Bryce, Daniel and Olivier Buffet (2008). “6th International Planning Competition: Uncertainty Part.” In: *Proceedings of the 6th International Planning Competition (IPC’08)*.
- Budde, Carlos E., Christian Dehnert, Ernst Moritz Hahn, Arnd Hartmanns, Sebastian Junges, and Andrea Turrini (2017). “JANI: Quantitative Model and Tool Interaction.” In: *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017*. Vol. 10206. Lecture Notes in Computer Science, pp. 151–168.
- Bylander, Tom (1994). “The Computational Complexity of Propositional STRIPS Planning.” In: *Artificial Intelligence* 69.1-2, pp. 165–204.
- Camacho, Alberto, Christian J. Muise, and Sheila A. McIlraith (2016). “From FOND to Robust Probabilistic Planning: Computing Compact Policies that Bypass Avoidable Deadends.” In: *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016*. AAAI Press, pp. 65–69.
- Celorrio, Sergio Jiménez, Tomás de la Rosa, Susana Fernández, Fernando Fernández, and Daniel Borrajo (2012). “A review of machine learning for automated planning.” In: *The Knowledge Engineering Review* 27.4, pp. 433–467.
- Chatterjee, Krishnendu, Martin Chmelik, Raghav Gupta, and Ayush Kanodia (2016). “Optimal cost almost-sure reachability in POMDPs.” In: *Artificial Intelligence* 234, pp. 26–48.
- Chvátal, Vašek (1983). *Linear programming*. W.H. Freeman.
- Ciesinski, Frank, Christel Baier, Marcus Größer, and Joachim Klein (2008). “Reduction Techniques for Model Checking Markov Decision Processes.” In: *Fifth International Conference on the Quantitative Evaluation of Systems (QEST 2008)*. IEEE Computer Society, pp. 45–54.
- Clarke, Edmund M., E. Allen Emerson, and A. Prasad Sistla (1986). “Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications.” In: *ACM Transactions on Programming Languages and Systems* 8.2, pp. 244–263.
- Coles, Amanda Jane (2012). “Opportunistic Branched Plans to Maximise Utility in the Presence of Resource Uncertainty.” In: *Proceedings of the 20th European Conference on Artificial Intelligence, ECAI 2012*. Vol. 242. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 252–257.
- Coles, Amanda Jane, Andrew Coles, Maria Fox, and Derek Long (2013). “A Hybrid LP-RPG Heuristic for Modelling Numeric Resource Flows in Planning.” In: *Journal of Artificial Intelligence Research* 46, pp. 343–412.
- Coles, Andrew and Amanda Smith (2007). “Marvin: A Heuristic Search Planner with Online Macro-Action Learning.” In: *Journal of Artificial Intelligence Research* 28, pp. 119–156.
- Courcoubetis, Costas and Mihalis Yannakakis (1995). “The Complexity of Probabilistic Verification.” In: *Journal of the ACM* 42.4, pp. 857–907.
- D’Argenio, Pedro R., Bertrand Jeannet, Henrik Ejersbo Jensen, and Kim Guldstrand Larsen (2001). “Reachability Analysis of Probabilistic Systems by Successive Refinements.” In: *Process Algebra and Probabilistic Methods, Performance Modeling and Verification: Joint International Workshop, PAPM-PROBMIV 2001*. Vol. 2165. Lecture Notes in Computer Science. Springer, pp. 39–56.
- Dai, Peng, Mausam, Daniel S. Weld, and Judy Goldsmith (2011). “Topological Value Iteration Algorithms.” In: *Journal of Artificial Intelligence Research* 42, pp. 181–209.
- Dantzig, George B. (1951). “Maximization of a linear function of variables subject to linear inequalities.” In: *Activity analysis of production and allocation* 13, pp. 359–373.
- Dantzig, George B. and Mukund N. Thapa (1997). *Linear Programming*. Springer-Verlag GmbH.

- Davis, Randall (1984). “Diagnostic Reasoning Based on Structure and Behavior.” In: *Artificial Intelligence* 24.1-3, pp. 347–410.
- De Alfaro, Luca (1997). “Formal Verification of Probabilistic Systems.” PhD thesis.
- Dean, Thomas L. and Robert Givan (1997). “Model Minimization in Markov Decision Processes.” In: *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97*. AAAI Press / The MIT Press, pp. 106–111.
- Dechter, Rina (1986). “Learning While Searching in Constraint-Satisfaction-Problems.” In: *Proceedings of the 5th National Conference on Artificial Intelligence*. Morgan Kaufmann, pp. 178–185.
- Dechter, Rina (1990). “Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition.” In: *Artificial Intelligence* 41.3, pp. 273–312.
- Dijkstra, Edsger W. (1959). “A note on two problems in connexion with graphs.” In: *Numerische Mathematik* 1, pp. 269–271.
- Domshlak, Carmel and Vitaly Mirkis (2015). “Deterministic Oversubscription Planning as Heuristic Search: Abstractions and Reformulations.” In: *Journal of Artificial Intelligence Research* 52, pp. 97–169.
- Domshlak, Carmel, Jörg Hoffmann, and Michael Katz (2015). “Red-black planning: A new systematic approach to partial delete relaxation.” In: *Artificial Intelligence* 221, pp. 73–114.
- Doran, James E. and Donald Michie (1966). “Experiments with the Graph Traverser program.” In: *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 294.1437, pp. 235–259.
- Dräger, Klaus, Bernd Finkbeiner, and Andreas Podelski (2009). “Directed model checking with distance-preserving abstractions.” In: *International Journal on Software Tools for Technology Transfer* 11.1, pp. 27–37.
- Edelkamp, Stefan (2002). “Symbolic Pattern Databases in Heuristic Search Planning.” In: *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, IJCAI 2002*. AAAI, pp. 274–283.
- Edelkamp, Stefan, Peter Kissmann, and Álvaro Torralba (2015). “BDDs Strike Back (in AI Planning).” In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI 2015*. AAAI Press, pp. 4320–4321.
- Eifler, Rebecca and Maximilian Fickert (2018). “Online Refinement of Cartesian Abstraction Heuristics.” In: *Proceedings of the Eleventh International Symposium on Combinatorial Search, SOCS 2018*. AAAI Press, pp. 46–54.
- Eifler, Rebecca, Michael Cashmore, Jörg Hoffmann, Daniele Magazzeni, and Marcel Steinmetz (2020a). “A New Approach to Plan-Space Explanation: Analyzing Plan-Property Dependencies in Oversubscription Planning.” In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020*. AAAI Press, pp. 9818–9826.
- Eifler, Rebecca, Marcel Steinmetz, Álvaro Torralba, and Jörg Hoffmann (2020b). “Plan-Space Explanation via Plan-Property Dependencies: Faster Algorithms & More Powerful Properties.” In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*. ijcai.org, pp. 4091–4097.
- Eriksson, Salomé, Gabriele Röger, and Malte Helmert (2017). “Unsolvability Certificates for Classical Planning.” In: *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017*. AAAI Press, pp. 88–97.
- Ernst, George W. and A. Newell (1971). “GPS: a case study in generality and problem solving.” In: *American Mathematical Monthly* 78, p. 923.
- Faqeh, Rasha, Christof Fetzer, Holger Hermanns, Jörg Hoffmann, Michaela Klauck, Maximilian A. Köhl, Marcel Steinmetz, and Christoph Weidenbach (2020). “Towards Dynamic Dependable Systems Through

- Evidence-Based Continuous Certification.” In: *Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISOFA 2020, Proceedings, Part II*, pp. 416–439.
- Feng, Lu, Clemens Wiltsche, Laura R. Humphrey, and Ufuk Topcu (2015). “Controller synthesis for autonomous systems interacting with human operators.” In: *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems, ICCPS 2015*. ACM, pp. 70–79.
- Fickert, Maximilian and Jörg Hoffmann (2017). “Complete Local Search: Boosting Hill-Climbing through Online Relaxation Refinement.” In: *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017*. AAAI Press, pp. 107–115.
- Fickert, Maximilian, Jörg Hoffmann, and Marcel Steinmetz (2016). “Combining the Delete Relaxation with Critical-Path Heuristics: A Direct Characterization.” In: *Journal of Artificial Intelligence Research* 56, pp. 269–327.
- Fikes, Richard and Nils J. Nilsson (1971). “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.” In: *Artificial Intelligence* 2.3/4, pp. 189–208.
- Fikes, Richard, Peter E. Hart, and Nils J. Nilsson (1972). “Learning and Executing Generalized Robot Plans.” In: *Artificial Intelligence* 3.1-3, pp. 251–288.
- Gale, David, Harold W Kuhn, and Albert W Tucker (1951). “Linear programming and the theory of games.” In: *Activity analysis of production and allocation* 13, pp. 317–335.
- Genesereth, Michael R. (1984). “The Use of Design Descriptions in Automated Diagnosis.” In: *Artificial Intelligence* 24.1-3, pp. 411–436.
- Ghallab, Malik, Dana S. Nau, and Paolo Traverso (2004). *Automated planning - theory and practice*. Elsevier.
- Givan, Robert, Sonia M. Leach, and Thomas L. Dean (2000). “Bounded-parameter Markov decision processes.” In: *Artificial Intelligence* 122.1-2, pp. 71–109.
- Givan, Robert, Thomas L. Dean, and Matthew Greig (2003). “Equivalence notions and model minimization in Markov decision processes.” In: *Artificial Intelligence* 147.1-2, pp. 163–223.
- Gnad, Daniel, Jörg Hoffmann, and Carmel Domshlak (2015). “From Fork Decoupling to Star-Topology Decoupling.” In: *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015*. AAAI Press, pp. 53–61.
- Gnad, Daniel, Álvaro Torralba, Jörg Hoffmann, and Martin Wehrle (2016a). “Decoupled Search for Proving Unsolvability.” In: *UIPC 2016 planner abstracts*, pp. 16–18.
- Gnad, Daniel, Marcel Steinmetz, and Jörg Hoffmann (2016b). “Django: Unchaining the Power of Red-Black Planning.” In: *UIPC 2016 planner abstracts*, pp. 19–23.
- Gnad, Daniel, Marcel Steinmetz, Mathäus Jany, Jörg Hoffmann, Ivan Serina, and Alfonso Gerevini (2016c). “Partial Delete Relaxation, Unchained: On Intractable Red-Black Planning and Its Applications.” In: *Proceedings of the Ninth Annual Symposium on Combinatorial Search, SOCS 2016*. AAAI Press, pp. 45–53.
- Gnad, Daniel, Álvaro Torralba, Martín Ariel Domínguez, Carlos Areces, and Facundo Bustos (2019). “Learning How to Ground a Plan - Partial Grounding in Classical Planning.” In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*. AAAI Press, pp. 7602–7609.
- Gros, Timo P., Holger Hermanns, Jörg Hoffmann, Michaela Klauck, and Marcel Steinmetz (2020a). “Deep Statistical Model Checking.” In: *Formal Techniques for Distributed Objects, Components, and Systems - 40th IFIP WG 6.1 International Conference, FORTE 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020*. Vol. 12136. Lecture Notes in Computer Science. Springer, pp. 96–114.

- Gros, Timo P., David Groß, Stefan Gumhold, Jörg Hoffmann, Michaela Klauck, and Marcel Steinmetz (2020b). “TraceVis: Towards Visualization for Deep Statistical Model Checking.” In: *Leveraging Applications of Formal Methods, Verification and Validation: Tools and Trends - 9th International Symposium on Leveraging Applications of Formal Methods, ISO LA 2020, Proceedings, Part IV*. Vol. 12479. Lecture Notes in Computer Science. Springer, pp. 27–46.
- Hahn, Ernst Moritz, Arnd Hartmanns, Christian Hensel, Michaela Klauck, Joachim Klein, Jan Kretínský, David Parker, Tim Quatmann, Enno Ruijters, and Marcel Steinmetz (2019). “The 2019 Comparison of Tools for the Analysis of Quantitative Formal Models - (QComp 2019 Competition Report).” In: *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Proceedings, Part III*. Vol. 11429. Lecture Notes in Computer Science. Springer, pp. 69–92.
- Hansen, Eric A. (2017). “Error bounds for stochastic shortest path problems.” In: *Mathematical Methods of Operations Research* 86.1, pp. 1–27.
- Hansen, Eric A. and Shlomo Zilberstein (2001). “LAO\*: A heuristic search algorithm that finds solutions with loops.” In: *Artificial Intelligence* 129.1-2, pp. 35–62.
- Hansson, Hans and Bengt Jonsson (1994). “A Logic for Reasoning about Time and Reliability.” In: *Formal Aspects of Computing* 6.5, pp. 512–535.
- Hart, Peter, Nils Nilsson, and Bertram Raphael (1968). “A Formal Basis for the Heuristic Determination of Minimum Cost Paths.” In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.
- Haslum, Patrik (2006). “Improving Heuristics Through Relaxed Search - An Analysis of TP4 and HSP\*a in the 2004 Planning Competition.” In: *Journal of Artificial Intelligence Research* 25, pp. 233–267.
- Haslum, Patrik (2012). “Incremental Lower Bounds for Additive Cost Planning Problems.” In: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012*. AAAI, pp. 74–82.
- Haslum, Patrik (2016). “Adapting  $h^{++}$  for Proving Plan Non-Existence.” In: *UIPC 2016 planner abstracts*, pp. 1–1.
- Haslum, Patrik and Hector Geffner (2000). “Admissible Heuristics for Optimal Planning.” In: *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, AIPS 2000*. AAAI, pp. 140–149.
- Haslum, Patrik and Hector Geffner (2001). “Heuristic Planning with Time and Resources.” In: *Proceedings of the 6th European Conference on Planning*, pp. 107–112.
- Haslum, Patrik, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig (2007). “Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning.” In: *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, AAAI 2007*. AAAI Press, pp. 1007–1012.
- Helmert, Malte (2006). “The Fast Downward Planning System.” In: *Journal of Artificial Intelligence Research* 26, pp. 191–246.
- Helmert, Malte (2009). “Concise finite-domain representations for PDDL planning tasks.” In: *Artificial Intelligence* 173.5-6, pp. 503–535.
- Helmert, Malte and Carmel Domshlak (2009). “Landmarks, Critical Paths and Abstractions: What’s the Difference Anyway?” In: *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009*. AAAI, pp. 162–169.
- Helmert, Malte and Hauke Lasinger (2010). “The Scanalyzer Domain: Greenhouse Logistics as a Planning Problem.” In: *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010*. AAAI, pp. 234–237.
- Helmert, Malte and Gabriele Röger (2008). “How Good is Almost Perfect?” In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*. AAAI Press, pp. 944–949.

- Helmert, Malte, Patrik Haslum, Jörg Hoffmann, and Raz Nissim (2014). “Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces.” In: *Journal of the ACM* 61.3, 16:1–16:63.
- Hermanns, Holger, Björn Wachter, and Lijun Zhang (2008). “Probabilistic CEGAR.” In: *Computer Aided Verification, 20th International Conference, CAV 2008*. Vol. 5123. Lecture Notes in Computer Science. Springer, pp. 162–175.
- Heusner, Manuel, Martin Wehrle, Florian Pommerening, and Malte Helmert (2014). “Under-Approximation Refinement for Classical Planning.” In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014*. AAAI.
- Hoffmann, Jörg (2005). “Where ‘Ignoring Delete Lists’ Works: Local Search Topology in Planning Benchmarks.” In: *Journal of Artificial Intelligence Research* 24, pp. 685–758.
- Hoffmann, Jörg (2011). “Analyzing Search Topology Without Running Any Search: On the Connection Between Causal Graphs and  $h^+$ .” In: *Journal of Artificial Intelligence Research* 41, pp. 155–229.
- Hoffmann, Jörg (2015). “Simulated Penetration Testing: From “Dijkstra” to “Turing Test++”.” In: *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015*. AAAI Press, pp. 364–372.
- Hoffmann, Jörg and Maximilian Fickert (2015). “Explicit Conjunctions without Compilation: Computing  $h^{FF}(Pi^C)$  in Polynomial Time.” In: *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015*. AAAI Press, pp. 115–119.
- Hoffmann, Jörg and Bernhard Nebel (2001). “The FF Planning System: Fast Plan Generation Through Heuristic Search.” In: *Journal of Artificial Intelligence Research* 14, pp. 253–302.
- Hoffmann, Jörg, Julie Porteous, and Laura Sebastia (2004). “Ordered Landmarks in Planning.” In: *Journal of Artificial Intelligence Research* 22, pp. 215–278.
- Hoffmann, Jörg, Peter Kissmann, and Álvaro Torralba (2014). ““Distance”? Who Cares? Tailoring Merge-and-Shrink Heuristics to Detect Unsolvability.” In: *ECAI 2014 - 21st European Conference on Artificial Intelligence*. Vol. 263. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 441–446.
- Hoffmann, Jörg, Holger Hermanns, Michaela Klauck, Marcel Steinmetz, Erez Karpas, and Daniele Magazzeni (2020). “Let’s Learn Their Language? A Case for Planning with Automata-Network Languages from Model Checking.” In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020*. AAAI Press, pp. 13569–13575.
- Hofmann, Till, Tim Niemueller, Jens Claßen, and Gerhard Lakemeyer (2016). “Continual Planning in Golog.” In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI 2016*. AAAI Press, pp. 3346–3353.
- Hou, Ping, William Yeoh, and Pradeep Varakantham (2014). “Revisiting Risk-Sensitive MDPs: New Algorithms and Results.” In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014*. AAAI, pp. 136–144.
- Howard, Ronald (1960). *Dynamic programming and Markov processes*. M.I.T. Press.
- Iba, Glenn A. (1989). “A Heuristic Approach to the Discovery of Macro-Operators.” In: *Machine Learning* 3, pp. 285–317.
- Iverson, Kenneth E. (1962). “A programming language.” In: *Proceedings of the 1962 spring joint computer conference, AFIPS 1962*. ACM, pp. 345–351.
- Jeong, Byeong-Min, Jung-Su Ha, and Han-Lim Choi (2014). “MDP-based mission planning for multi-UAV persistent surveillance.” In: *2014 14th International Conference on Control, Automation and Systems, ICCAS 2014*. IEEE, pp. 831–834.

- Jimenez, S., A.I. Coles, and A.J. Smith (2006). "Planning in probabilistic domains using a deterministic numeric planner." In: *25th Workshop of the UK Planning and Scheduling Special Interest Group*.
- Jiménez, Pablo and Carme Torras (2000). "An efficient algorithm for searching implicit AND/OR graphs with cycles." In: *Artificial Intelligence* 124.1, pp. 1–30.
- Kambhampati, Subbarao (1998). "On the Relations Between Intelligent Backtracking and Failure-Driven Explanation-Based Learning in Constraint Satisfaction and Planning." In: *Artificial Intelligence* 105.1-2, pp. 161–208.
- Kambhampati, Subbarao (2000). "Planning Graph as a (Dynamic) CSP: Exploiting EBL, DDB and other CSP Search Techniques in Graphplan." In: *Journal of Artificial Intelligence Research* 12, pp. 1–34.
- Kambhampati, Subbarao, Suresh Katukam, and Yong Qu (1996). "Failure Driven Dynamic Search Control for Partial Order Planners: An Explanation Based Approach." In: *Artificial Intelligence* 88.1-2, pp. 253–315.
- Karpas, Erez and Carmel Domshlak (2009). "Cost-Optimal Planning with Landmarks." In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, pp. 1728–1733.
- Kautz, Henry A. and Bart Selman (1999). "Unifying SAT-based and Graph-based Planning." In: *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 1999*. Morgan Kaufmann, pp. 318–325.
- Keyder, Emil Ragip, Jörg Hoffmann, and Patrik Haslum (2012). "Semi-Relaxed Plan Heuristics." In: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012*. AAAI.
- Keyder, Emil Ragip, Jörg Hoffmann, and Patrik Haslum (2014). "Improving Delete Relaxation Heuristics Through Explicitly Represented Conjunctions." In: *Journal of Artificial Intelligence Research* 50, pp. 487–533.
- Khachiyan, Leonid (1979). "A Polynomial Algorithm in Linear Programming." In: *Doklady Akademii Nauk SSSR* 224, pp. 1093–1096.
- Klauck, Michaela and Holger Hermanns (2021). "A Modest Approach to Dynamic Heuristic Search in Probabilistic Model Checking." In: *Quantitative Evaluation of Systems - 18th International Conference, QEST 2021*. Vol. 12846. Lecture Notes in Computer Science. Springer, pp. 15–38.
- Klauck, Michaela, Marcel Steinmetz, Jörg Hoffmann, and Holger Hermanns (2018). "Compiling Probabilistic Model Checking into Probabilistic Planning." In: *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018*. AAAI Press, pp. 150–154.
- Klauck, Michaela, Marcel Steinmetz, Jörg Hoffmann, and Holger Hermanns (2020). "Bridging the Gap Between Probabilistic Model Checking and Probabilistic Planning: Survey, Compilations, and Empirical Comparison." In: *Journal of Artificial Intelligence Research* 68, pp. 247–310.
- Klee, Victor and George J. Minty (1972). "How good is the simplex algorithm?" In: *Inequalities III*, pp. 159–175.
- Klößner, Thorsten, Jörg Hoffmann, Marcel Steinmetz, and Álvaro Torralba (2021). "Pattern Databases for Goal-Probability Maximization in Probabilistic Planning." In: *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021*. AAAI Press, pp. 201–209.
- Klößner, Thorsten, Florian Pommerening, Thomas Keller, and Gabriele Röger (2022a). "Cost Partitioning Heuristics for Stochastic Shortest Path Problems." In: *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022*. AAAI Press, pp. 193–202.
- Klößner, Thorsten, Marcel Steinmetz, Álvaro Torralba, and Jörg Hoffmann (2022b). "Pattern Selection Strategies for Pattern Databases in Probabilistic Planning." In: *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022*. AAAI Press, pp. 184–192.
- Kolobov, Andrey (2013). "Scalable methods and expressive models for planning under uncertainty." PhD thesis. University of Washington.

- Kolobov, Andrey, Mausam, and Daniel S. Weld (2010a). "Classical Planning in MDP Heuristics: with a Little Help from Generalization." In: *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010*. AAAI, pp. 97–104.
- Kolobov, Andrey, Mausam, and Daniel S. Weld (2010b). "SixthSense: Fast and Reliable Recognition of Dead Ends in MDPs." In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*. AAAI Press, pp. 1108–1114.
- Kolobov, Andrey, Mausam, Daniel S. Weld, and Hector Geffner (2011). "Heuristic Search for Generalized Stochastic Shortest Path MDPs." In: *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011*. AAAI, pp. 130–137.
- Kolobov, Andrey, Mausam, and Daniel S. Weld (2012a). "A Theory of Goal-Oriented MDPs with Dead Ends." In: *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, pp. 438–447.
- Kolobov, Andrey, Mausam, and Daniel S. Weld (2012b). "Discovering hidden structure in factored MDPs." In: *Artificial Intelligence* 189, pp. 19–47.
- Korf, Richard E. (1990). "Real-Time Heuristic Search." In: *Artificial Intelligence* 42.2-3, pp. 189–211.
- Korovin, Konstantin and Martin Suda (2016). "IProverPlan: a system description." In: *UIPC 2016 planner abstracts*, pp. 6–7.
- Korte, Bernhard and Jens Vygen (2000). *Combinatorial Optimization*. Springer Berlin Heidelberg.
- Kurniawati, Hanna, David Hsu, and Wee Sun Lee (2008). "SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces." In: *Robotics: Science and Systems IV*. The MIT Press.
- Kuter, Ugur and Jiaqiao Hu (2007). "Computing and Using Lower and Upper Bounds for Action Elimination in MDP Planning." In: *Abstraction, Reformulation, and Approximation, 7th International Symposium, Proceedings, SARA 2007*. Vol. 4612. Lecture Notes in Computer Science. Springer, pp. 243–257.
- Kwiatkowska, Marta Z., Gethin Norman, and David Parker (2002). "Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach." In: *Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002*. Vol. 2280. Lecture Notes in Computer Science. Springer, pp. 52–66.
- Kwiatkowska, Marta Z., Gethin Norman, and David Parker (2006). "Game-based Abstraction for Markov Decision Processes." In: *Third International Conference on the Quantitative Evaluation of Systems (QEST 2006)*. IEEE Computer Society, pp. 157–166.
- Larsen, Kim Guldstrand and Arne Skou (1991). "Bisimulation through Probabilistic Testing." In: *Information and Computation* 94.1, pp. 1–28.
- Lipovetzky, Nir and Hector Geffner (2012). "Width and Serialization of Classical Planning Problems." In: *20th European Conference on Artificial Intelligence, ECAI 2012*. Vol. 242. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 540–545.
- Lipovetzky, Nir, Christian J. Muise, and Hector Geffner (2016). "Traps, Invariants, and Dead-Ends." In: *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016*. AAAI Press, pp. 211–215.
- Little, Iain, Douglas Aberdeen, and Sylvie Thiébaux (2005). "Prottrle: A Probabilistic Temporal Planner." In: *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*. AAAI Press / The MIT Press, pp. 1181–1186.
- Littman, Michael L. (1997). "Probabilistic Propositional Planning: Representations and Complexity." In: *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97*. AAAI Press / The MIT Press, pp. 748–754.
- Löhr, Johannes (2014). "Planning in Hybrid Domains: Domain Predictive Control = Planen in Hybriden Domänen: Domänen-prädiktive Regelung." PhD thesis. University of Freiburg, Germany.

- Long, Derek and Maria Fox (1999). "Efficient Implementation of the Plan Graph in STAN." In: *Journal of Artificial Intelligence Research* 10, pp. 87–115.
- Marecki, Janusz and Milind Tambe (2008). "Towards Faster Planning with Continuous Resources in Stochastic Domains." In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*. AAAI Press, pp. 1049–1055.
- McMahan, H. Brendan, Maxim Likhachev, and Geoffrey J. Gordon (2005). "Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees." In: *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005)*. Vol. 119. ACM International Conference Proceeding Series. ACM, pp. 569–576.
- McMillan, Kenneth L. (1993). *Symbolic model checking*. Kluwer.
- Meuleau, Nicolas, Emmanuel Benazera, Ronen I. Brafman, Eric A. Hansen, and Mausam (2009). "A Heuristic Search Approach to Planning with Continuous Resources in Stochastic Domains." In: *Journal of Artificial Intelligence Research* 34, pp. 27–59.
- Milner, Robin (1990). "Operational and Algebraic Semantics of Concurrent Processes." In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Elsevier and MIT Press, pp. 1201–1242.
- Minton, Steven, Jaime G. Carbonell, Craig A. Knoblock, Daniel Kuokka, Oren Etzioni, and Yolanda Gil (1989). "Explanation-Based Learning: A Problem Solving Perspective." In: *Artificial Intelligence* 40.1-3, pp. 63–118.
- Moore, Edward F. (1959). "The shortest path through a maze." In: *Proceedings of the International Symposium on the Theory of Switching*. Harvard University Press, pp. 285–292.
- Moskewicz, Matthew W., Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik (2001). "Chaff: Engineering an Efficient SAT Solver." In: *Proceedings of the 38th Design Automation Conference, DAC 2001*. ACM, pp. 530–535.
- Mostow, Jack and Neeraj Bhatnagar (1987). "Failsafe - A Floor Planner that Uses EBG to Learn from Its Failures." In: *Proceedings of the 10th International Joint Conference on Artificial Intelligence, IJCAI 1987*. Morgan Kaufmann, pp. 249–255.
- Muise, Christian J., Sheila A. McIlraith, and J. Christopher Beck (2012a). "Improved Non-Deterministic Planning by Exploiting State Relevance." In: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012*. AAAI, pp. 172–180.
- Muise, Christian J., Sheila A. McIlraith, and J. Christopher Beck (2012b). "Improved Non-Deterministic Planning by Exploiting State Relevance." In: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012*. AAAI, pp. 172–180.
- Nakhost, Hootan, Jörg Hoffmann, and Martin Müller (2012). "Resource-Constrained Planning: A Monte Carlo Random Walk Approach." In: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012*. AAAI, pp. 181–189.
- Newell, Allen, J. C. Shaw, and Herbert A. Simon (1959). "Report on a general problem-solving program." In: *Information Processing, Proceedings of the 1st International Conference on Information Processing*. UNESCO (Paris), pp. 256–264.
- Nilsson, Nils J. (1971). *Problem-solving methods in artificial intelligence*. McGraw-Hill computer science series. McGraw-Hill.
- Nissim, Raz, Jörg Hoffmann, and Malte Helmert (2011). "Computing Perfect Heuristics in Polynomial Time: On Bisimulation and Merge-and-Shrink Abstraction in Optimal Planning." In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011*. IJCAI/AAAI, pp. 1983–1990.
- Papadimitriou, Christos H. and John N. Tsitsiklis (1987). "The Complexity of Markov Decision Processes." In: *Mathematical Methods of Operations Research* 12.3, pp. 441–450.

- Papadimitriou, Christos H. and Mihalis Yannakakis (1991). "Shortest Paths Without a Map." In: *Theoretical Computer Science* 84.1, pp. 127–150.
- Pearl, Judea (1984). *Heuristics - intelligent search strategies for computer problem solving*. Addison-Wesley series in artificial intelligence. Addison-Wesley.
- Pell, Barney, Douglas E. Bernard, Steve A. Chien, Erann Gat, Nicola Muscettola, P. Pandurang Nayak, Michael D. Wagner, and Brian C. Williams (1998). "An Autonomous Spacecraft Agent Prototype." In: *Autonomous Robots* 5.1, pp. 29–52.
- Pineda, Luis Enrique, Takeshi Takahashi, Hee-Tae Jung, Shlomo Zilberstein, and Roderic A. Grupen (2015). "Continual planning for search and rescue robots." In: *15th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2015, IEEE-RAS 2015*. IEEE, pp. 243–248.
- Pommerening, Florian and Malte Helmert (2015). "A Normal Form for Classical Planning Tasks." In: *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015*. AAAI Press, pp. 188–192.
- Pommerening, Florian and Jendrik Seipp (2016). "Fast Downward Dead-End Pattern Database." In: *UIPC 2016 planner abstracts*, pp. 2–2.
- Pommerening, Florian, Gabriele Röger, and Malte Helmert (2013). "Getting the Most Out of Pattern Databases for Classical Planning." In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013*. IJCAI/AAAI, pp. 2357–2364.
- Pommerening, Florian, Gabriele Röger, Malte Helmert, and Blai Bonet (2014). "LP-Based Heuristics for Cost-Optimal Planning." In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014*. AAAI, pp. 226–234.
- Pommerening, Florian, Malte Helmert, Gabriele Röger, and Jendrik Seipp (2015). "From Non-Negative to General Operator Cost Partitioning." In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI 2015*. AAAI Press, pp. 3335–3341.
- Pommerening, Florian, Malte Helmert, and Blai Bonet (2017). "Higher-Dimensional Potential Heuristics for Optimal Classical Planning." In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI 2017*. AAAI Press, pp. 3636–3643.
- Pováda, Guillaume, Olivier Regnier-Coudert, Florent Teichteil-Königsbuch, Gérard Dupont, Alexandre Arnold, Jonathan Guerra, and Mathieu Picard (2019). "Evolutionary approaches to dynamic earth observation satellites mission planning under uncertainty." In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019*. ACM, pp. 1302–1310.
- Prosser, Patrick (1993). "Hybrid Algorithms for the Constraint Satisfaction Problem." In: *Computational Intelligence* 9, pp. 268–299.
- Puterman, Martin L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley.
- Quatmann, Tim and Joost-Pieter Katoen (2018). "Sound Value Iteration." In: *Computer Aided Verification - 30th International Conference, CAV 2018*. Vol. 10981. Lecture Notes in Computer Science. Springer, pp. 643–661.
- Reinefeld, Alexander and T. Anthony Marsland (1994). "Enhanced Iterative-Deepening Search." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16.7, pp. 701–710.
- Richter, Silvia and Matthias Westphal (2010). "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks." In: *Journal of Artificial Intelligence Research* 39, pp. 127–177.
- Rintanen, Jussi, Keijo Heljanko, and Ilkka Niemelä (2006). "Planning as satisfiability: parallel plans and algorithms for plan search." In: *Artificial Intelligence* 170.12-13, pp. 1031–1080.
- Ruml, Wheeler, Minh Binh Do, Rong Zhou, and Markus P. J. Fromherz (2011). "On-line Planning and Scheduling: An Application to Controlling Modular Printers." In: *Journal of Artificial Intelligence Research* 40, pp. 415–468.

- Russell, Stuart and Peter Norvig (2010). *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall.
- Sanner, Scott (2010). “Relational Dynamic Influence Diagram Language (RDDL): Language Description.” [http://users.cecs.anu.edu.au/~ssanner/IPPC\\_2011/RDDL.pdf](http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf).
- Sarraute, Carlos, Olivier Buffet, and Jörg Hoffmann (2012). “POMDPs Make Better Hackers: Accounting for Uncertainty in Penetration Testing.” In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2012*. AAAI Press.
- Seipp, Jendrik (2021). “Online Saturated Cost Partitioning for Classical Planning.” In: *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021*. AAAI Press, pp. 317–321.
- Seipp, Jendrik and Malte Helmert (2018). “Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning.” In: *Journal of Artificial Intelligence Research* 62, pp. 535–577.
- Seipp, Jendrik, Florian Pommerening, and Malte Helmert (2015). “New Optimization Functions for Potential Heuristics.” In: *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015*. AAAI Press, pp. 193–201.
- Seipp, Jendrik, Florian Pommerening, Silvan Sievers, and Martin Wehrle (2016). “Fast Downward Aidos.” In: *UIPC 2016 planner abstracts*, pp. 28–38.
- Shmaryahu, Dorin, Guy Shani, Jörg Hoffmann, and Marcel Steinmetz (2018). “Simulated Penetration Testing as Contingent Planning.” In: *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018*, pp. 241–249.
- Sievers, Silvan and Malte Helmert (2021). “Merge-and-Shrink: A Compositional Theory of Transformations of Factored Transition Systems.” In: *Journal of Artificial Intelligence Research* 71, pp. 781–883.
- Silva, João P. Marques and Karem A. Sakallah (1996). “GRASP - a new search algorithm for satisfiability.” In: *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996*. IEEE Computer Society / ACM, pp. 220–227.
- Smith, Trey and Reid G. Simmons (2006). “Focused Real-Time Dynamic Programming for MDPs: Squeezing More Out of a Heuristic.” In: *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*. AAAI Press, pp. 1227–1232.
- Speicher, Patrick, Marcel Steinmetz, Daniel Gnad, Jörg Hoffmann, and Alfonso Gerevini (2017). “Beyond Red-Black Planning: Limited-Memory State Variables.” In: *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017*, pp. 269–273.
- Speicher, Patrick, Marcel Steinmetz, Robert Künnemann, Milivoj Simeonovski, Giancarlo Pellegrino, Jörg Hoffmann, and Michael Backes (2018a). “Formally Reasoning about the Cost and Efficacy of Securing the Email Infrastructure.” In: *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018*, pp. 77–91.
- Speicher, Patrick, Marcel Steinmetz, Michael Backes, Jörg Hoffmann, and Robert Künnemann (2018b). “Stackelberg Planning: Towards Effective Leader-Follower State Space Search.” In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18)*. AAAI Press, pp. 6286–6293.
- Speicher, Patrick, Marcel Steinmetz, Jörg Hoffmann, Michael Backes, and Robert Künnemann (2019). “Towards automated network mitigation analysis.” In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019*, pp. 1971–1978.
- Stallman, Richard M. and Gerald J. Sussman (1977). “Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis.” In: *Artificial Intelligence* 9.2, pp. 135–196.
- Steinmetz, Marcel (2015). “Learning Dead Ends Through Explicit Conjunctions.” MA thesis.

- Steinmetz, Marcel and Jörg Hoffmann (2016). “Towards Clause-Learning State Space Search: Learning to Recognize Dead-Ends.” In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI 2016. AAAI Press, pp. 760–768.
- Steinmetz, Marcel and Jörg Hoffmann (2017a). “Critical-Path Dead-End Detection versus NoGoods: Offline Equivalence and Online Learning.” In: *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling*, ICAPS 2017. AAAI Press, pp. 283–287.
- Steinmetz, Marcel and Jörg Hoffmann (2017b). “Search and Learn: On Dead-End Detectors, the Traps they Set, and Trap Learning.” In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, IJCAI 2017, pp. 4398–4404.
- Steinmetz, Marcel and Jörg Hoffmann (2017c). “State space search nogood learning: Online refinement of critical-path dead-end detectors in planning.” In: *Artificial Intelligence* 245, pp. 1–37.
- Steinmetz, Marcel and Jörg Hoffmann (2018). “LP Heuristics over Conjunctions: Compilation, Convergence, Nogood Learning.” In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, IJCAI 2018, pp. 4837–4843.
- Steinmetz, Marcel and Álvaro Torralba (2019). “Bridging the Gap between Abstractions and Critical-Path Heuristics via Hypergraphs.” In: *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling*, ICAPS 2019. AAAI Press, pp. 473–481.
- Steinmetz, Marcel, Jörg Hoffmann, and Olivier Buffet (2016a). “Goal Probability Analysis in Probabilistic Planning: Exploring and Enhancing the State of the Art.” In: *Journal of Artificial Intelligence Research* 57, pp. 229–271.
- Steinmetz, Marcel, Jörg Hoffmann, and Olivier Buffet (2016b). “Revisiting Goal Probability Analysis in Probabilistic Planning.” In: *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling*, ICAPS 2016. AAAI Press, pp. 299–307.
- Steinmetz, Marcel, Jörg Hoffmann, Alisa Kovtunova, and Stefan Borgwardt (2022a). “Classical Planning with Avoid Conditions.” In: *Thirty-Sixth AAAI Conference on Artificial Intelligence*, AAAI 2022, *Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence*, IAAI 2022, *The Twelveth Symposium on Educational Advances in Artificial Intelligence*, EAAI 2022, pp. 9944–9952.
- Steinmetz, Marcel, Daniel Fiser, Hasan Ferit Eniser, Patrick Ferber, Timo P. Gros, Philippe Heim, Daniel Höller, Xandra Schuler, Valentin Wüstholtz, Maria Christakis, and Jörg Hoffmann (2022b). “Debugging a Policy: Automatic Action-Policy Testing in AI Planning.” In: *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling*, ICAPS 2022. AAAI Press, pp. 353–361.
- Suda, Martin (2014). “Property Directed Reachability for Automated Planning.” In: *Journal of Artificial Intelligence Research* 50, pp. 265–319.
- Tagorti, Manel, Bruno Scherrer, Olivier Buffet, and Joerg Hoffmann (2013). “Abstraction Pathologies In Markov Decision Processes.” In: *8èmes Journées Francophones sur la Planification, la Décision et l’Apprentissage pour la conduite de systèmes*. Actes des 8èmes Journées Francophones sur la Planification, la Décision et l’Apprentissage pour la conduite de systèmes.
- Tarjan, Robert (1972). “Depth-First Search and Linear Graph Algorithms.” In: *SIAM Journal on Computing* 1.2, pp. 146–160.
- Tarski, Alfred (1955). “A lattice-theoretical fixpoint theorem and its applications.” In: *Pacific Journal of Mathematics* 5.2, pp. 285–309.
- Teichteil-Königsbuch, Florent (2005). “Approche symbolique et heuristique de la planification en environnement incertain : optimisation d’une stratégie de déplacement et de prise d’information.” PhD thesis.
- Teichteil-Königsbuch, Florent (2012). “Stochastic Safest and Shortest Path Problems.” In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI 2012. AAAI Press, pp. 1825–1831.

- Teichteil-Königsbuch, Florent and Patrick Fabiani (2006). “Autonomous Search and Rescue Rotorcraft Mission Stochastic Planning with Generic DBNs.” In: *Artificial Intelligence in Theory and Practice, IFIP 19th World Computer Congress, TC 12: IFIP AI 2006*. Vol. 217. IFIP. Springer, pp. 483–492.
- Teichteil-Königsbuch, Florent, Ugur Kuter, and Guillaume Infantes (2010). “Incremental plan aggregation for generating policies in MDPs.” In: *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Volume 1-3*. IFAAMAS, pp. 1231–1238.
- Teichteil-Königsbuch, Florent, Vincent Vidal, and Guillaume Infantes (2011). “Extending Classical Planning Heuristics to Probabilistic Planning with Dead-Ends.” In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011*. AAAI Press, pp. 1017–1022.
- Torralba, Álvaro (2016). “SymPA: Symbolic Perimeter Abstractions for Proving Unsolvability.” In: *UIPC 2016 planner abstracts*, pp. 8–11.
- Torralba, Álvaro and Jörg Hoffmann (2015). “Simulation-Based Admissible Dominance Pruning.” In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*. AAAI Press, pp. 1689–1695.
- Torralba, Álvaro and Peter Kissmann (2015). “Focusing on What Really Matters: Irrelevance Pruning in Merge-and-Shrink.” In: *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015*. AAAI Press, pp. 122–130.
- Torralba, Álvaro, Jörg Hoffmann, and Peter Kissmann (2016). “MS-Unsat and SimulationDominance: Merge-and-Shrink and Dominance Pruning for Proving Unsolvability.” In: *UIPC 2016 planner abstracts*, pp. 12–15.
- Torralba, Álvaro, Patrick Speicher, Robert Künnemann, Marcel Steinmetz, and Jörg Hoffmann (2021). “Faster Stackelberg Planning via Symbolic Search and Information Sharing.” In: *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021*. AAAI Press, pp. 11998–12006.
- Trevizan, Felipe W., Sylvie Thiébaux, Pedro Henrique Santana, and Brian Charles Williams (2016). “Heuristic Search in Dual Space for Constrained Stochastic Shortest Path Problems.” In: *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016*. AAAI Press, pp. 326–334.
- Trevizan, Felipe W., Florent Teichteil-Königsbuch, and Sylvie Thiébaux (2017a). “Efficient solutions for Stochastic Shortest Path Problems with Dead Ends.” In: *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017*. AUAI Press.
- Trevizan, Felipe W., Sylvie Thiébaux, and Patrik Haslum (2017b). “Occupation Measure Heuristics for Probabilistic Planning.” In: *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017*. AAAI Press, pp. 306–315.
- Valmari, Antti (1989). “Stubborn sets for reduced state space generation.” In: *Advances in Petri Nets 1990 [10th International Conference on Applications and Theory of Petri Nets, Proceedings]*. Vol. 483. Lecture Notes in Computer Science. Springer, pp. 491–515.
- Van den Briel, Menkes, J. Benton, Subbarao Kambhampati, and Thomas Vossen (2007). “An LP-Based Heuristic for Optimal Planning.” In: *Principles and Practice of Constraint Programming - CP 2007*. Vol. 4741. Lecture Notes in Computer Science. Springer, pp. 651–665.
- Vinzent, Marcel, Marcel Steinmetz, and Jörg Hoffmann (2022). “Neural Network Action Policy Verification via Predicate Abstraction.” In: *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022*. AAAI Press, pp. 371–379.
- Wehrle, Martin and Malte Helmert (2014). “Efficient Stubborn Sets: Generalized Algorithms and Selection Strategies.” In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014*. AAAI, pp. 323–331.

- Wei, Junqing, John M. Dolan, Jarrod M. Snider, and Bakhtiar Litkouhi (2011). “A point-based MDP for robust single-lane autonomous driving behavior under uncertainties.” In: *IEEE International Conference on Robotics and Automation, ICRA 2011*. IEEE, pp. 2586–2592.
- Wiehr, Frederik, Anke Hirsch, Lukas Schmitz, Nina Knieriemen, Antonio Krüger, Alisa Kovtunova, Stefan Borgwardt, Ernie Chang, Vera Demberg, Marcel Steinmetz, and Jörg Hoffmann (2021). “Why Do I Have to Take Over Control? Evaluating Safe Handovers with Advance Notice and Explanations in HAD.” In: *ICMI '21: International Conference on Multimodal Interaction*. ACM, pp. 308–317.
- Wilhelm, Anna, Marcel Steinmetz, and Jörg Hoffmann (2018). “On Stubborn Sets and Planning with Resources.” In: *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018*, pp. 288–297.
- Younes, Håkan L. S., Michael L. Littman, David Weissman, and John Asmuth (2005). “The First Probabilistic Track of the International Planning Competition.” In: *Journal of Artificial Intelligence Research* 24, pp. 851–887.
- Zimmerman, Terry and Subbarao Kambhampati (2003). “Learning-Assisted Automated Planning: Looking Back, Taking Stock, Going Forward.” In: *AI Magazine* 24.2, pp. 73–96.