Approximation Algorithms for Network Design and Cut Problems in Bounded-Treewidth

A dissertation submitted towards the degree
Doctor of Natural Sciences
of the Faculty of Mathematics and Computer Science
of Saarland University

by Daniel Vaz

Saarbrücken / 2020

Day of Colloquium:	8. December 2020
Dean of the Faculty:	Prof. Dr. Thomas Schuster
Chair of the Committee:	Prof. Dr. Markus Bläser
Reporters	
First reviewer:	Prof. Dr. Kurt Mehlhorn
Second reviewer:	Prof. Dr. Parinya Chalermsook
Third reviewer:	Prof. Dr. Robert Krauthgamer
Academic Assistant:	Dr. Sándor Kisfaludi-Bak

Abstract

Abstract This thesis explores two optimization problems, the *group Steiner tree* and *firefighter* problems, which are known to be NP-hard even on trees. We study the approximability of these problems on trees and bounded-treewidth graphs.

In the group Steiner tree, the input is a graph and sets of vertices called groups; the goal is to choose one representative from each group and connect all the representatives with minimum cost. We show an $O(\log^2 n)$-approximation algorithm for bounded-treewidth graphs, matching the known lower bound for trees, and improving the best possible result using previous techniques. We also show improved approximation results for group Steiner forest, directed Steiner forest, and a fault-tolerant version of group Steiner tree.

In the firefighter problem, we are given a graph and a vertex which is burning. At each time step, we can protect one vertex that is not burning; fire then spreads to all unprotected neighbors of burning vertices. The goal is to maximize the number of vertices that the fire does not reach. On trees, a classic $(1-1/e)$-approximation algorithm is known via LP rounding. We prove that the integrality gap of the LP matches this approximation, and show significant evidence that additional constraints may improve its integrality gap. On bounded-treewidth graphs, we show that it is NP-hard to find a subpolynomial approximation even on graphs of treewidth 5. We complement this result with an $O(1)$-approximation on outerplanar graphs.

Zusammenfassung Diese Arbeit untersucht zwei Optimierungsprobleme, von welchen wir wissen, dass sie selbst in Bäumen NP-schwer sind. Wir analysieren Approximationen für diese Probleme in Bäumen und Graphen mit begrenzter Baumweite.

Im *Gruppensteinerbaumproblem*, sind ein Graph und Mengen von Knoten (Gruppen) gegeben; das Ziel ist es, einen Knoten von jeder Gruppe mit minimalen Kosten zu verbinden. Wir beschreiben einen $O(\log^2 n)$-Approximationsalgorithmus für Graphen mit beschränkter Baumweite, dies entspricht der zuvor bekannten unteren Schranke für Bäume und ist zudem eine Verbesserung über die bestmöglichen Resultate die auf anderen Techniken beruhen. Darüber hinaus zeigen wir verbesserte Approximationsresultate für andere Gruppensteinerprobleme.

Im *Feuerwehrproblem* sind ein Graph zusammen mit einem brennenden Knoten gegeben. In jedem Zeitschritt können wir einen Knoten der noch nicht brennt auswählen und diesen vor dem Feuer beschützen. Das Feuer breitet sich anschließend zu allen Nachbarn aus. Das Ziel ist es die Anzahl der Knoten die vom Feuer unberührt bleiben zu maximieren. In Bäumen existiert ein lang bekannter $(1-1/e)$-Approximationsalgorithmus der auf LP Rundung basiert. Wir zeigen, dass die Ganzzahligkeitslücke des LP tatsächlich dieser Approximation entspricht, und dass weitere Einschränkungen die Ganzzahligkeitslücke möglicherweise verbessern könnten. Für Graphen mit beschränkter Baumweite zeigen wir, dass es NP-schwer ist, eine sub-polynomielle Approximation zu finden.

Acknowledgments

First and foremost, I would like to thank Parinya Chalermsook for being an excellent advisor and mentor. Thanks for guiding me through this journey, and always being there when I was in need of advice or help. I learned a lot from you, and always had fun doing it. Thank you also for always trying to bring us all together as much as possible, so we could focus on research and solving problems.

I am very grateful to Kurt Mehlhorn, who agreed to supervise my PhD and created a wonderful atmosphere for me and everyone in the group. Speaking to Kurt or listening to one of his talks was always captivating, and I hope I have the opportunity to do it again soon.

Research is not something I like to do alone, and so I want to thank everyone that I collaborated with during my PhD. I am happy that I got to work with so many bright and friendly people, who ensured that research was always enjoyable. A special thank you to Parinya, Syamantak, Bundit, Guy, Erik Jan, and Antonios, for being friends as well as collaborators.

I am grateful to the Max Planck Institute for Informatics for financing my PhD, and to all of the staff for trying their hardest to assist us in any way they could. Moreover, the Saarbrücken Graduate School of Computer Science played an important role, especially before the doctoral research phase, by allowing me to explore and decide on my research path. I also feel very fortunate for having had the opportunity to visit other academic institutions: thank you Universidad de Chile, Simons Institute and Aalto University for hosting me during my visits.

Thank you to everyone in the Algorithms and Complexity group. It was a pleasure being around such friendly and brilliant people, and there was never a dull day at work. A big thank you especially to Bhaskar, André, Attila, Andi, Michael, Pavel, and the monkeys. It was thanks to you that my time in Saarbrücken was so fun, and that I left with a lot more hobbies than I arrived with.

I cannot thank my family enough: for raising me and always feeding my curiosity, for supporting me in everything I do, and for always being there for me. Obrigado do fundo do meu coração.

Thank you to all of my friends, who may have played a small role in this work, but a significant role in my life.

Last but not least, thank you, Alina, for sharing my life and being the best companion I could ever ask for.

Contents

1 Introduction **1**
- 1.1 Algorithms for NP-hard Problems Beyond Trees . . . 2
- 1.2 Problems and Contributions . . . 3
- 1.3 Organization . . . 5

2 Notation and Preliminaries **7**
- 2.1 Graph Notation . . . 7
- 2.2 Treewidth and Tree Decompositions . . . 9
- 2.3 Computational Regimes for Coping with NP-hardness . . . 11
- 2.4 Standard Algorithmic Tools . . . 13

I Network Design on Bounded Treewidth Graphs **15**

3 Group Steiner Tree **17**
- 3.1 Problem Definitions and Results . . . 20
- 3.2 The Algorithm of Garg, Konjevod and Ravi . . . 21
- 3.3 Connectivity for GST on Bounded-Treewidth Graphs . . . 29
- 3.4 Solving GST on Bounded-Treewidth Graphs . . . 39
- 3.5 Group Steiner Forest on Bounded-Treewidth Graphs . . . 48
- 3.6 Directed Steiner Forest on Bounded-Treewidth Graphs . . . 53

4 Fault-Tolerant Group Steiner Tree **57**
- 4.1 Problem Definitions and Results . . . 60
- 4.2 Connectivity Lemma Based Approach . . . 61
- 4.3 Connectivity-K Mimicking Networks . . . 70

5 Conclusion and Open Problems **79**
- 5.1 Open Problems . . . 80

II Firefighter Problem **83**

6 Firefighter Problem on Trees **85**
- 6.1 Problem Definitions and Results . . . 87
- 6.2 Standard Linear Program and Preliminaries . . . 89
- 6.3 Integrality Gap Instances for the Standard LP . . . 90
- 6.4 Improving the Standard LP with Hartke's Constraints . . . 97

7 Firefighter Problem beyond Trees **117**
7.1 Problem Definitions and Results 118
7.2 The Firefighter Problem on Outerplanar Graphs 119
7.3 The Firefighter Problem on Bounded-Treewidth Graphs 128

8 Conclusion and Open Problems **135**
8.1 Open Problems 135

Appendix **139**

A Appendix for Part I **139**
A.1 Omitted proofs of Chapter 3 139
A.2 Algorithms 141

B Appendix for Part II **147**
B.1 Algorithms 147
B.2 Integrality instance for LP-Hartke ($\alpha = 2$) 148

List of Figures

3.1 Relation between various network design problems. 18

4.1 Example of different partitions of edges into paths from demands γ_1, γ_2. . 62

6.1 Simplified instance used to achieve integrality gap of $1 - 1/e$ for LP-FF. . 92
6.2 Instance with a non-integral extreme point for LP-FF. 100
6.3 Gadget used to achieve an integrality gap of 5/6 for LP-Hartke. 111
6.4 Example of the integrality gap instance for LP-Hartke, when $\alpha = 1$. . . 112
6.5 Simplified instance used to achieve integrality gap of 5/6 for LP-Hartke. 113

7.1 Example of delay and influence. 120
7.2 Construction of T_ϕ for 3 variables and one example clause, $x_1 \vee x_2 \vee x_3$. . 130

8.1 Example of $\Omega(n)$-completable instance with treewidth 2. 137

B.1 Full example of the integrality gap instance for LP-Hartke, when $\alpha = 2$. 148

List of Algorithms

A.1 Complete algorithm to approximate GST [GKR00]. 141
A.2 Modified algorithm for rounding an instance of Theorem 3.38. 142
A.3 Algorithm to approximately solve GSF. 143
A.4 Algorithm to approximately solve MDGSF. 144
A.5 Algorithm for computing a minor connectivity-K mimicking network. . . . 145
B.1 Greedy algorithm for k-completable and k-weak-completable graphs. . . . 147

CHAPTER 1

Introduction

Computers have revolutionized the world as we know it today, with numerous applications in all aspects of society. Unfortunately, many computational problems that are useful in practice have been proven to be NP-hard, and thus it is believed that they cannot be solved efficiently. There are three characteristics of an algorithm that are mostly considered regarding NP-hard problems: running time, solution quality, and generality. A consequence of the NP-hardness of a problem is that it is impossible (or incredibly difficult) to obtain an algorithm that has good running time, perfect solution quality, and is general.

We view this thesis as presenting research on approximation algorithms: we are interested in studying the existence of algorithms that solve a problem in its general form, and whose running time is bounded by a polynomial on the input length, at the cost of outputting a solution that is less than optimal, but guarantees that the solution is not much worse than the optimum. More formally, an $\alpha(n)$-approximation algorithm for a minimization problem, where $\alpha : \mathbb{Z}^+ \to \mathbb{R}$ is a function on the size of the instance, is an algorithm that outputs a feasible solution of cost at most $\alpha(n) \cdot \text{opt}$ for any instance of size n and with optimum cost opt. Similarly, an α-approximation algorithm for a maximization problem always outputs a feasible solution of cost at least $\alpha(n) \cdot \text{opt}$.

As part of our journey to obtain general approximation algorithms, we will also veer into the field of parameterized algorithms, which trade off generality with running time, by considering a parameter of the input instance; the higher the parameter, the more general is the algorithm and the higher the running time. Our goal is to design FPT algorithms, which given an instance of a problem of size n and a parameter k (which can be, for instance, the maximum degree, or the size of the optimum solution), find the optimum solution in time $f(k)\,\text{poly}(n)$, where f is any computable function. An FPT algorithm limits the combinatorial explosion to the chosen parameter, meaning that it runs in polynomial time with regard to the instance size.

Network design problems, a wide class of problems with many practical applications (see e.g. [GK11, He13, HKR$^+$11]), are often successfully handled using approximation algorithms [GK11, GK11, KV18]. These problems concern the task of finding subgraphs establishing connections between nodes, while satisfying certain properties, such as tolerance to link or node failures. Numerous classic graph problems are in this class: the traveling salesman problem, Steiner tree, matching problems, among others.

Many of these problems can be solved effortlessly on simple graphs such as trees, and, even on general graphs, have constant-approximation algorithms, which often match the best achievable guarantees. However, some problems are NP-hard even on trees, and become harder still on general graphs. In this thesis, we focus on two such problems, and how they can be approximated beyond trees.

The first problem we consider is the *group Steiner tree* (GST) problem, in which we

are given a base graph and a collection of groups of vertices, and are asked to find a minimum-cost way to connect a representative of each of the given groups. GST is a well-known network design problem that generalizes Steiner tree, and is known to be surprisingly hard. The second problem we consider is the *firefighter problem* (Max-FF), in which we are given a graph and a fire source, that is, a vertex that is "burning". This problem can be viewed as one-player game played on the input graph: in each turn, we can protect a vertex that is not burning, making it immune to fire; then, the fire spreads from all currently burning vertices to their neighbors that were not protected. The goal is to choose a strategy that maximizes the number of the saved vertices, which the fire does not reach. We are interested in these problems because of their rich complexity: besides being NP-hard to solve even on trees, there are some gaps in knowledge between the tree case and the general graph case.

In the group Steiner tree problem, existing results relied on a known technique to transform general graphs into trees while preserving distances in the graph [Bar96, FRT04]. These techniques, known as *metric tree embedding techniques*, work well in general, but they cannot be improved even on very simple graphs [CG04, GNR$^+$04]. In our work, we show new techniques that allow us to improve the approximation ratio for bounded-treewidth graphs, a generalization of trees for which metric tree embedding techniques could not improve the state of the art. In fact, our results are tight from an approximation point of view: our approximation guarantees match the best possible result for trees [HK03], and hence are tight for any generalization. Our results also apply in settings for which aforementioned techniques are not known, such as when the cost of the solution depends on the vertices in the solution, and not only on the edges. Additionally, we study an extension of group Steiner tree in which we want to connect the groups in a fault-tolerant manner, and provide a framework to solve problems in similar settings.

In the firefighter problem, we start by studying the problem on trees, and show that the standard linear program used to approximate the problem cannot yield any improved results. We also show how to augment the standard linear program so as to overcome the previous barrier, even though we do not show a general improved approximation algorithm. We then turn our attention to more general classes of graphs: we show an approximation algorithm for outerplanar graphs (see Section 2.1 for definitions). We also show that even in bounded-treewidth graphs, it is NP-hard to give any good approximation. Our results show that at a very low value of treewidth (between 2 and 5), the problem changes from being approximable, to not having any good approximations.

1.1 Algorithms for NP-hard Problems Beyond Trees

Many of the NP-hard graph problems that are studied in computer science, such as network design problems, tend to be much simpler when the input graph is a tree, and are often even solvable in polynomial time (see e.g. [CFK$^+$15]). These algorithms for trees are not necessarily helpful, but there are some approaches to go from trees to more general graphs.

One of these techniques are *metric tree embeddings* [Bar98, FRT04], which can be used to embed a graph into a tree, where the problem can then be solved. This technique is extremely general, and can be used for some problems where the objective depends

on distances between vertices (see [FRT04] for applications). For the group Steiner tree problem, this is the only currently-known approach to approximate the problem on general graphs [FRT04, GKR00]. Unfortunately, the use of this technique implies a loss of $\Omega(\log n)$ in the approximation ratio [CG04, GNR$^+$04], meaning that, to obtain better results, new techniques are needed.

In this thesis, we focus on a different way of generalizing trees, which uses the techniques developed on trees to obtain results on bounded-treewidth graphs.

The concept of treewidth was introduced by Robertson and Seymour [RS84] in their seminal work on graph minors (after being known under different names [BB73, Hal76]). Since then, treewidth has been widely used in parameterized algorithms (see for example [CFK$^+$15, Chapter 7] and references within), due to the fact that many problems studied and considered fundamental in computer science are tractable when the input graph has bounded treewidth. Furthermore, bounded-treewidth graphs have special significance as a stepping stone to planar graphs: starting with Baker [Bak94], several approximation algorithms (namely PTAS) for planar graphs have been obtained by reduction to the bounded-treewidth case (see [DHK05, DHK11, Kle05]).

This concept also finds applicability in practice. For example, logical inference on graphical models relies on treewidth to decide what methods to use [PDGF$^+$15], and control flow graphs arising in the compilation of structured programs are known to have low treewidth, which implies that they also have good register allocation [Tho98].

Intuitively, treewidth measures how "tree-like" a graph is, that is, how similar the structure of a graph is to a tree. More concretely, a tree decomposition groups vertices into bags, and arranges these bags into a tree structure, in a way that maintains certain desirable properties. Treewidth measures how large these bags must be, so that the graph can be organized into a tree. This underlying tree structure is extremely helpful, since it helps us to adapt the techniques used to solve problems on trees, such as recursion and dynamic programming, to more general graphs.

See Section 2.2 for more details on tree decompositions and treewidth.

1.2 Problems and Contributions

1.2.1 The Group Steiner Tree (GST) Problem

In this problem, we are given a graph with costs on the edges, and a collection of subsets of vertices that we call groups. Our goal is to find a minimum cost subtree of the graph that connects to (at least) one vertex of each group, which we call the representative. The group Steiner tree problem finds applicability in circuit design and broadcasting, in situations where there is some flexibility in the choice of elements to connect or their positioning [RW89]. Besides its applicability, the problem also attracts a great deal of theoretical interest.

After Reich and Widmayer proposed the problem in 1989 [RW89], Garg, Konjevod and Ravi [GKR00] presented an algorithm that outputs a solution whose cost is a factor of $O(\log n \log h)$ away from the optimum, when the input graph is a tree of size n, with h groups. By applying the metric tree embedding technique by Bartal [Bar96], improved on by Fakcharoenphol et al. [FRT04]), this result implies a $O(\log^2 n \log h)$-approximation to the problem in general graphs. Surprisingly, Halperin and Krauthgamer presented a

matching lower bound for the tree case: they showed that, even when the input graph is a tree, there is no $O(\log^{2-\varepsilon} h)$-approximation, under some reasonable complexity assumptions [HK03].

The main open problem regarding GST is whether we can achieve an $O(\log n \log h)$-approximation in general graphs. In our work, we show that, for a restricted class of graphs, the answer is affirmative: we present an $O(\log n \log h)$-approximation algorithm for graphs with bounded treewidth. This result strictly improves the best known approximation factor for bounded-treewidth graphs, since metric tree embedding techniques incur the loss of $\Omega(\log n)$ in the approximation factor, even on graphs with treewidth 2 [CG04, GNR+04]. It is our hope that our results can be used to obtain a better approximation ratio for planar graphs, and that our techniques may inspire further results in more general graph classes.

In this thesis, we show a simplified version of the results published at SODA'17 [CDL+17]. Furthermore, we present our results generalizing the approach to other group problems [CDE+18], such as a fault-tolerant variant of the problem, where the representative of each group must be connected by k edge-disjoint (or vertex-disjoint) paths.

1.2.2 The Firefighter Problem

The firefighter problem can be seen as a 1-player game on a graph G with a fire source s, which is initially *burning*. In each round, the player can choose a non-burning vertex to *protect*. Then, all the unprotected vertices neighboring a burning vertex start burning as well. The goal is to find a strategy that maximizes the number of non-burning vertices when the game ends, that is, when no new vertices can start burning. This problem models the spread of natural phenomena, such as forest fires, diseases, or ideas [FM09, Har95].

In general graphs, it has been shown that it is NP-hard to $n^{1-\varepsilon}$-approximate the problem [ACH+12]. For this reason, most of the work on this problem looks at more specific cases, such as when the input graph is a tree. In this situation, a simple greedy 2-approximation [HL00], as well as a $(1 - 1/e)$-approximation using randomized rounding of a linear program [CVY08] are known. Concurrently to our work, Adjiashvili et al. [ABZ17] obtained a PTAS ($(1 + \varepsilon)$-approximation for any $\varepsilon > 0$) for this problem. Their result also uses the standard linear program as part of the algorithm, but preprocesses the instance so that the rounding procedure outputs a better approximation.

In our work, we show that $1 - 1/e$ is the best approximation ratio achievable using the standard linear program, and we provide some evidence that extending it using constraints suggested by Hartke may improve the approximation ratio. Seeing as how rounding the linear program yielded the best approximation for a long time, and is still used as part of the PTAS by Adjiashvili et al. [ABZ17], we think that studying the limitations of the standard LP and improving them is an important question, which can lead to improved rounding algorithms or simplified PTAS.

In later work, we turn our attention to studying the complexity landscape of the problem: while the tree case is NP-hard, but can be well approximated, the general case does not have any good approximation. Our results are two-fold: first, we show that even on bounded-treewidth graphs (treewidth 5), computing an $n^{1-\varepsilon}$-approximation is

NP-hard, for any $\varepsilon > 0$; the above result also applies when, on graphs of treewidth w, fewer than $w/2 - 1$ vertices can be protected per turn. Our second result shows that on outerplanar graphs (planar graphs where all vertices are on the same face, see Section 2.1 for definitions), the problem has an $O(1)$-approximation. It is likely that this result can be generalized to other classes of graphs, especially k-outerplanar graphs. We leave this question as an open problem.

1.3 Organization

We start the thesis by presenting some notation and well-known results that are used throughout. The rest of the thesis is divided into two parts. In Part I, our results on the group Steiner tree problem (Chapter 3) and its fault-tolerant extension (Chapter 4) are described. In Part II, the firefighter problem is studied, starting with some results on trees (Chapter 6), and then the results on outerplanar graphs and bounded-treewidth graphs (Chapter 7). We finish each part with a conclusion and discussion of future work.

CHAPTER 2

Notation and Preliminaries

2.1 Graph Notation

Throughout this document, we assume a graph to be unweighted, undirected and simple unless otherwise specified. For most problems where the input is a graph, we will refer to the input graph as $G = (V, E)$, the number of vertices as $n = |V|$, and the number of edges as $m = |E|$.

If G is directed, we consider $E \subseteq V \times V$, and refer to the edges as (u, v). For undirected graphs, we will usually consider E to be a symmetric subset of $V \times V$, where $(u, v) \in E$ implies that $(v, u) \in E$; we denote the edge between u and v as (u, v), (v, u), or uv. In order to distinguish between results for directed and undirected graphs, we either use the notation uv for undirected graphs, or explicitly state when it is assumed that E is symmetric (though in most cases, this assumption is not necessary).

A weight or cost vector is a vector $w \in \mathbb{R}^E$ (edge weights) or $w \in \mathbb{R}^V$ (vertex weights). We use the notation w_e (resp. w_v) to mean the weight of the corresponding edge e (resp. vertex v), that is, the coordinate of w indexed by e (resp. v). For subsets of elements (vertices or edges), we use the notation $w(S)$ to mean the sum of the weights of those elements, $w(S) = \sum_{s \in S} w_s$, for $S \subseteq E$ (resp. $S \subseteq V$). Unless otherwise specified, weights are specified on edges.

We say a subgraph $p \subseteq G$ is a *path*, if it has vertex set $V(p) = \{p_1, p_2, \ldots, p_{|p|}\}$ and edge set $E(p) = \{(p_{i-1}, p_i) : i \in \{2, \ldots, |p|\}\}$. The *length* of a path p is $|E(p)|$, the number of its edges. For convenience, we may consider a path p as a tuple of vertices or of edges, according to context. Given two paths p, q, we denote by $p \circ q$ the concatenation of paths p and q, that is, the path $(p_1, p_2, \ldots, p_{|p|}, q_1, \ldots, q_{|q|})$, provided that the edge $p_{|p|}q_1$ exists (in G). The *shortest path metric* of a graph G with edge-weights $w \in \mathbb{R}^E$ is a metric $(V(G), d_{(G,w)})$, where $d_{(G,w)}(u, v)$ is the minimum weight $w(E(p))$ of a path p from u to v.

Given two graphs G_1, G_2, their union $G_1 \cup G_2$ is a multigraph whose vertex set is the union $V(G_1) \cup V(G_2)$, and which has an edge $uv \in G$ for every edge $uv \in G_1$, and for every edge $uv \in G_2$ (which means it might have parallel edges if G_1 and G_2 contain the same edge).

A *cut* (V_1, V_2) is a partition of V, and its *cutset* is the set of edges connecting V_1, V_2, denoted $E(V_1, V_2)$. Given disjoint $S_1, S_2 \subseteq V$, we say (V_1, V_2) is a cut between S_1 and S_2, or that (V_1, V_2) *separates* S_1 from S_2, if $S_1 \subseteq V_1$, $S_2 \subseteq V_2$. The *weight* of a cut is the total weight of the edges in its cutset (edges have weight 1 if not specified). A *mincut* separating S_1 from S_2 is a cut between S_1 and S_2 with minimum weight.

Directed graphs Let $G = (V, E)$ be a directed graph. E is the set of *arcs* (or simply edges) of G, and for every arc $e = (u, v) \in E$, we say that e is an *arc from u to v*, and u

and v are the *tail* and *head* of e, respectively.

The *in-degree* (resp. *out-degree*) of a vertex $v \in V$ is the number of arcs in E whose head (resp. tail) is v.

We say G is a *directed acyclic graph* (*DAG*) if it contains no directed cycles (see [Die12] for properties of these graphs). A DAG is *rooted* if there is exactly one vertex (the root) with in-degree 0.

An *arborescence* (also *out-arborescence*) is a directed, rooted tree where every edge points away from the root. An *in-arborescence* is similar, but with all the edges pointing towards the root.

Trees and arborescences When G is a tree or an (in or out)-arborescence with root r, we introduce additional notation. We will define all the notation with respect to an out-arborescence. The definitions for trees and in-arborescences can be obtained from the corresponding out-arborescence, by directing all of the edges away from the root.

The *parent* $p(v)$ of v in G is the tail of the (unique) arc to v. The set of *children* of v, denoted $C(v)$, contains the heads of the arcs from v, that is, the set of all vertices u such that there is an arc from v to u. An out-arborescence is *d-ary*, for some $d \in \mathbb{Z}_{\geq 1}$ if every vertex has at most d children.

We say that u is an *ancestor* of v if there is a directed path from u to v. u is the *ℓ-th ancestor* of v, denoted $p^{(\ell)}(v)$ if u is the unique vertex such that a path of length ℓ between u and v exists ($p(v)$ is the first ancestor of v).

The *subtree of G rooted at v*, denoted G_v, is the subgraph of G induced by all the vertices reachable from v, that is, all vertices u such that there is a path from v to u.

The *depth* of a vertex v is the length of the unique path from the root to v. The *height* of v, or equivalently, the height of G_v, is the maximum length of a path from v to a leaf in G_v. The *depth* or *height* of a tree G is the height of the root r.

Given a set of vertices (for example, that satisfy a certain condition), a *topmost* or *highest* vertex in the set is a vertex that has minimum depth among vertices in the set. Similarly, a *lowest* vertex is a vertex that has maximum depth among vertices in the set. Following this notation, the *lowest common ancestor* of vertices u and v is the maximum-depth vertex that is an ancestor of both u and v.

All of these definitions apply to DAGs as well, except that parents, ℓ-th ancestors, and lowest common ancestors may not be unique.

Proposition 2.1. *Let V be a set of vertices of size $|V| = n$. The number of possible forests on V is at most n^n.*

Proof. Every forest on V can be uniquely identified by a parent function $p : V \to V$ ($p(u) = v$ if v is the parent of u, $p(v) = v$ if v has no parent). Therefore, the number of forests on V is at most the number of possible functions $p : V \to V$, which is n^n. □

Proposition 2.2. *Let G be a tree with $\ell \geq 3$ leaves, such that every internal vertex of G has degree at least 3. Then $|V(G)| \leq 2\ell - 2$.*

Proof. Let the root r of G be an internal vertex. Removing r from G will leave us with subtrees $G_1, G_2, \ldots, G_k$, $k \geq 3$. If $\ell = 3$, then we must have $k = 3$ and each G_i is a single leaf, since each tree must have at least one leaf and there is only one possibility for trees with one leaf, so $V(G) \leq 4$.

Assume by induction that the statement holds for $\ell' < \ell$. Then

$$|V(G)| = 1 + \sum_{i=1}^{k} |V(G_i)| \leq 1 + \sum_{i=1}^{k} (2\ell_i - 2) = 2\ell - (2k - 1) \leq 2\ell - 2 \qquad \square$$

Planar and outerplanar graphs We use the notation and results presented by Diestel [Die12, Chapter 4]. A *planar graph* is a graph that can be embedded onto the plane ($\mathbb{R}^2$) in such a way that edges only intersect at their endpoints (i.e. there are no crossings). An embedding of a planar graph is also referred to as *plane graph*. The *faces* of an embedding are the contiguous regions of $\mathbb{R}^2 \setminus G$. The *outer face* is the unique face that is unbounded. Let f be a face of an embedding of G. The *subgraph of G induced by f*, $G[f]$, is the subgraph of G containing exactly all of the vertices and edges in the frontier (or boundary) of f .

An *outerplanar graph* is a planar graph such that the outer face of some embedding contains all of the vertices. We generally consider an arbitrary such embedding when talking about outerplanar graphs. A *k-outerplanar graph* is defined recursively as follows: a 1-outerplanar graph is simply an outerplanar graph; a k-outerplanar graph is a planar graph such that, for an appropriate embedding, removing the vertices and edges on the outer face results in a $(k-1)$-outerplanar graph.

2.2 Treewidth and Tree Decompositions

The concepts of treewidth and tree decomposition were proposed by Robert and Seymour in their seminal paper [RS84]. These concepts have been used ever since and have found numerous applications in the field of algorithms.

Before stating the formal definition, let us consider how treewidth might be useful as a parameter when designing algorithms. The set of graphs with treewidth 1 is simply the set of all trees. When considering problems on trees, many problems can be easily solved, because their structure allows for simple ideas to be used, such as recursion or bottom-up dynamic programming. For example, if we can obtain a solution for a tree from solutions for its subtrees, we can get a solution by "building it up", that is, starting from leaves, and then constructing solutions for progressively bigger subtrees, until we have a solution for the entire tree.

This approach works on trees for two reasons: (1) we have a tree structure, which allows us to use a bottom-up recursive strategy; (2) the "interface" of a subtree is a single vertex, that is, there is only one vertex connecting the subtree to the rest of the graph, which in many cases implies that dependence between the solution in the subtree and outside of it can be restricted to that single vertex.

The concept of tree decomposition pushes this idea a step further: we still want a tree-like structure, but we allow the interface of a subinstance (subtree in the previous case) to contain a small number of vertices. It is important that the number of vertices in the interface is indeed small, as we usually need to enumerate all the possibilities for the solution on these vertices.

Definition 2.3. Let G be an undirected graph. A *tree decomposition* is a pair $(\mathcal{T}, \mathcal{X})$ where $\mathcal{T}$ is a tree and $\mathcal{X} = \{X_t \subseteq V(G)\}_{t \in V(\mathcal{T})}$ is a collection of *bags* such that:

(1) $V(G) = \bigcup_{t \in V(\mathcal{T})} X_t$, that is, every $v \in V(G)$ is contained in some bag X_t;

(2) For any edge $uv \in E$, there is a bag X_t that contains both u and v, i.e. $u, v \in X_t$;

(3) For each vertex $v \in V(G)$, the collection of nodes t whose bags X_t contain v induces a connected subgraph of $\mathcal{T}$, that is, $\mathcal{T}[\{t \in V(\mathcal{T}) : v \in X_t\}]$ is a (connected) subtree.

We will use the term *node* to refer to an element $t \in V(\mathcal{T})$, and *bag* to refer to the corresponding subset X_t.

The *treewidth* of G, denoted $\mathrm{tw}(G)$, is the minimum *width* of any tree decomposition $(\mathcal{T}, \mathcal{X})$ for G. The width of $(\mathcal{T}, \mathcal{X})$ is given by $\max |X_t| - 1$. When working with directed graphs, we will consider the treewidth and tree decomposition of its undirected counterpart, that is, the undirected graph with the same edges (but without direction) as G.

Let G be a graph and $(\mathcal{T}, \mathcal{X})$ be its tree decomposition. We will say that each edge $uv \in E(G)$ *belongs* to a unique bag X_t, and write $e \in E_t$, if $t \in \mathcal{T}$ is the node closest to the root such that $u, v \in X_t$. For a subset $\mathcal{S} \subseteq V(\mathcal{T})$, we define $X(\mathcal{S}) := \bigcup_{t \in \mathcal{S}} X_t$. We also define G_t as the subgraph with vertices $X(\mathcal{T}_t)$ and edges $E(G_t) = \bigcup_{t' \in \mathcal{T}_t} E_{t'}$. For each $v \in V$, we denote by t_v the node closest to the root for which $v \in X_{t_v}$.

Another interpretation of Property (3) is that, for any $v \in V$, there is a single node $t_v \in \mathcal{T}$ where v *arises*, that is, such that v is in X_t but not in $X_{p(t)}$; for every other bag $X_{t'}$ containing v, it must have been passed on from the parent. We can use this property in a recursive algorithm by passing down some information about a vertex v to the children bags containing v. Since there is a single bag in which v arises, this makes sure that all bags containing v get the same information.

We will use the following result, which shows that a tree decomposition of G of width $O(tw(G))$ is computable in time $O(2^{O(tw(G))}n)$.

Theorem 2.4 ([BDD+16])**.** *There is an algorithm that, given a graph G, finds a tree decomposition $(\mathcal{T}, \{X_t\}_{t \in V(\mathcal{T})})$ such that $|X_t| \leq 5\,\mathrm{tw}(G)$ for all t, in time $O\big(2^{O(tw(G))}n\big)$.*

In order to simplify notation, we assume that $\mathcal{T}$ is a binary tree. Furthermore, it is helpful to require the height of $\mathcal{T}$ to be $O(\log n)$. Lemma 2.6, based on the following result of Bodlaender [Bod88], summarizes the properties we assume.

Theorem 2.5 ([Bod88, Theorem 4.2])**.** *Let $G = (V, E)$, with $n = |V|$ and $\mathrm{tw}(G) \leq w$.*

Then G has a tree-decomposition $(\mathcal{T}, \mathcal{X})$ with $\mathcal{T}$ a binary tree with depth at most $O(\log n)$, and the width of this decomposition is at most $3w + 2$.

In fact, the proof of Theorem 2.5 shows how to transform any given tree decomposition of G with width w into one with the properties above. This transformation runs in near-linear time in the size of $\mathcal{T}$.

Lemma 2.6. *Given a tree decomposition $(\mathcal{T}', \mathcal{X}')$ of width w, we can transform it into a tree decomposition $(\mathcal{T}, \mathcal{X})$ with the following properties:*

(1) the height of $\mathcal{T}$ is at most $O(\log n)$;

(2) each bag X_t satisfies $|X_t| \leq 3w + 2$;

(3) every non-leaf has exactly 2 *children;*

(4) every leaf bag has no edges ($E_t = \emptyset$ for leaf $t \in \mathcal{T}$).

Furthermore, this transformation runs in near-linear time in the size of $\mathcal{T}$.

Proof. Theorem 2.5 shows us how to transform $(\mathcal{T}', \mathcal{X}')$ so that it is a binary tree and satisfies Properties (1) and (2).

Let t be a node that does not satisfy Property (3); by Theorem 2.5, it must have exactly one child. We make a copy t' of t and make it the child of t. It can be easily seen that this does not affect the previous properties or contradicts the definition of tree decomposition. Furthermore, t' satisfies Property (4).

For any leaf node t that does not satisfy Property (4), we make two copies t', t'', and make them the children of t. Since $X_t = X_{t'} = X_{t''}$, no edges belong to t' and t''. □

2.3 Computational Regimes for Coping with NP-hardness

2.3.1 Approximation Algorithms

Approximation algorithms are a common approach to handle NP-hardness of an optimization problem: since an NP-hard problem cannot be solved to optimality (unless P = NP), we instead develop efficient algorithms that find a solution provably close to the optimum.

Let $\mathcal{P}$ be a minimization problem with objective function f. Given an instance I of $\mathcal{P}$, the goal of the problem is to find a feasible solution S for I that minimizes $f(S)$. Let $\mathrm{opt}(I)$ be the optimum value of f over all feasible solutions for an instance I, and $\mathrm{OPT}(I)$ be an arbitrary feasible solution minimizing f, that is, $f(\mathrm{OPT}(I)) = \mathrm{opt}(I)$. The definition is analogous for maximization problems.

Definition 2.7. Let $\mathcal{A}$ be an algorithm for a problem $\mathcal{P}$, $\alpha : \mathbb{Z}^+ \to \mathbb{R}$ be a function, and let $\mathrm{alg}(I) = f(\mathcal{A}(I))$ be the value of the solution $\mathcal{A}(I)$ returned by $\mathcal{A}$.

We say that $\mathcal{A}$ is an $\alpha(n)$-*approximation algorithm* for $\mathcal{P}$ if, for all instances I of $\mathcal{P}$,

$$\begin{aligned} \mathrm{opt}(I) \le \mathrm{alg}(I) &\le \alpha(n)\,\mathrm{opt}(I) && \text{(minimization, } \alpha(n) \ge 1\text{)} \\ \alpha(n)\,\mathrm{opt}(I) \le \mathrm{alg}(I) &\le \mathrm{opt}(I) && \text{(maximization, } \alpha(n) \le 1\text{)} \end{aligned}$$

The *approximation ratio* or *approximation factor* of $\mathcal{A}$ is the value $\alpha(n)$ closest to 1 such that $\mathcal{A}$ is an $\alpha(n)$-approximation algorithm for $\mathcal{P}$.

If $\mathcal{A}$ is a randomized algorithm, we consider the expected value of the solution, that is, $\mathrm{alg}(I) = \mathrm{E}[f(\mathcal{A}(I))]$.

If $\mathcal{P}$ is a maximization problem, we sometimes consider the ratio $\mathrm{opt}(I)/\,\mathrm{alg}(I)$ (that is, we say $\mathcal{A}$ is an $O(\log n)$-approximation instead of an $\Omega(1/\log n)$-approximation).

For some problems, we have a *polynomial time approximation scheme* (PTAS), which can output, in polynomial time, an $(1+\varepsilon)$-approximate solution for any constant $\varepsilon > 0$. PTAS allow us to get to within a very small percentage of the solution by increasing the running time of the algorithm.

Definition 2.8. Let $\mathcal{P}$ be a minimization problem.

A *polynomial-time approximation scheme* (PTAS) for $\mathcal{P}$ is a family of approximation algorithms $\{\mathcal{A}_\varepsilon\}$ such that, for every constant $\varepsilon > 0$, there is a $(1+\varepsilon)$-approximation algorithm $\mathcal{A}_\varepsilon$ running in polynomial time. For maximization problems, $\mathcal{A}_\varepsilon$ is a $(1-\varepsilon)$-approximation algorithm.

We say $\{\mathcal{A}_\varepsilon\}$ is an *efficient polynomial-time approximation scheme* (EPTAS) if, for every $\varepsilon > 0$, the running time of $\mathcal{A}_\varepsilon$ is $f(1/\varepsilon)\,\mathrm{poly}(n)$, where f is some computable function and n is the size of the input. If f is a polynomial, $\{\mathcal{A}_\varepsilon\}$ is a *fully-polynomial-time approximation scheme* (FPTAS).

The goal when studying the *approximability* of a problem is to determine the best approximation ratio of an algorithm to that problem. To that end, we also consider *hardness of approximation* results, which show what approximation ratios are unlikely. We say that a problem $\mathcal{P}$ is $\alpha(n)$-*inapproximable* under some complexity assumption (usually $\mathrm{P} \neq \mathrm{NP}$), if the assumption implies that there is no $\alpha(n)$-approximation algorithm for $\mathcal{P}$. If the assumption is $\mathrm{P} \neq \mathrm{NP}$, we also say that it is NP-hard to $\alpha(n)$-approximate $\mathcal{P}$.

For more details and examples, see [KV18, Vaz01, WS11].

2.3.2 Parameterized Complexity

Parameterized complexity is the field of computer science that aims to characterize the hardness of NP-hard problems according to additional parameters. The parameters can be anything from the size of the solution, to a property of the input graph, like the maximum degree or treewidth. A problem associated with a parameter is called a parameterized problem, and we can formally define it as follows (for decision problems).

Definition 2.9 ([CNP+11])**.** A *parameterized (decision) problem* $\mathcal{P}$ is specified by a language $L \subseteq \Sigma^* \times \mathbb{Z}_{\geq 0}$ containing all YES-instances of $\mathcal{P}$, where Σ is a fixed, finite alphabet. For an instance $(x, k) \in \Sigma^* \times \mathbb{Z}_{\geq 0}$, k is called the parameter.

Intuitively, a parameterized problem is simply a problem where each instance is augmented with a non-negative integer parameter. This definition also generalizes to optimization problems, as well as to multiple parameters.

We can now define relevant computation classes for parameterized problems. In this thesis, we are mostly interested in the classes of fixed-parameter tractable (FPT) and slice-wise polynomial (XP) problems.

Definition 2.10 ([CNP+11])**.** Let $\mathcal{P}$ be a parameterized (decision) problem.

We say $\mathcal{P}$ is *fixed-parameter tractable* (FPT) if there is an algorithm $\mathcal{A}$, a computable function $f : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ and a constant $c \in \mathbb{Z}_{\geq 0}$ such that, given an instance (x, k), $\mathcal{A}$ correctly decides if (x, k) is a YES-instance in time bounded by $f(k) \cdot (x + k)^c$.

We say $\mathcal{P}$ is *slice-wise polynomial* (XP) if there is an algorithm $\mathcal{A}$ and computable functions $f, g : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ such that, given an instance (x, k), $\mathcal{A}$ correctly decides if (x, k) is a YES-instance in time bounded by $f(k) \cdot (x + k)^{g(k)}$.

The class of all fixed-parameter tractable problems is denoted by FPT, and the class of all slice-wise polynomial problems by XP. An algorithm that certifies that $\mathcal{P}$ is in FPT or in XP is called an FPT or XP algorithm, respectively.

For optimization problems the definition is similar, with the difference being that $\mathcal{A}$ must compute an optimum solution, instead of deciding whether (x, k) is a YES-instance.

2.4 Standard Algorithmic Tools

2.4.1 Linear Programming

Linear programming is one of the most used techniques in approximation algorithms (see e.g. [WS11] or [Vaz01, Part II]). In this section, we will give a small introduction to linear programs and introduce some notation that will be used throughout this thesis. A more thorough introduction, as well as references to the results presented in this section can be found in the book by Bertsimas and Tsitsiklis [BT97].

Many interesting problems in combinatorial optimization have two interesting properties: (i) the objective function is linear in the solution (for example, the objective may be to minimize the sum of weights of the edges in the solution), and (ii) the feasibility constraints of the problem, that is, the rules to decide whether a solution is feasible, are linear, or can be expressed as linear constraints. Problems satisfying these constraints can be expressed as an (integer) linear program.

When formulating a linear program (LP), the first step is to choose the *decision variables*. The assignment to these variables is restricted by linear inequalities or equations called *constraints*. If an assignment satisfies all of the constraints of a linear program, we say this assignment is a *feasible solution*. Finally, an LP has a linear *objective function*. The goal of linear programming is referred to as *solving* the LP, and consists in finding an *optimum solution*, that is, a feasible solution that optimizes (either maximizes or minimizes) the objective function. The *value* of a feasible solution (resp. an LP) is the value of the objective function for that assignment (resp. for the assignment of the optimum solution). The *support* of a solution x, denoted $\text{supp}(x)$ is the set of variables that are assigned a non-zero value.

Definition 2.11. A *linear program* (LP) is a problem of the form

$$\min c^T x, \text{s.t. } Ax \geq b, x \geq 0 \tag{2.1}$$

where $x \in \mathbb{R}^n$ is the vector of n decision variables, and $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ encode the m constraints.

The value of the expression in (2.1) is the *value* of the LP. A vector $x \in \mathbb{R}^n$ is a *feasible solution* if $Ax \geq b, x \geq 0$; its *value* is $c^T x$. If the value of x is equal to the value of the LP, x is an *optimum solution*. The *support* of x is $\text{supp}(x) = \{i \in [n] : x_i > 0\}$.

All LPs can be written exactly as above. However, it might be easier to slightly adjust the expression for other problems, e.g. by changing min to max, $\geq$ to $\leq$, or (partially) removing the constraint $x \geq 0$.

If the variables are constrained to take only integer values, we call the linear program an *integer program* (IP). If only some of the variables must be integer, it is called a *mixed integer program*. Solving IPs is NP-hard, as they generalize many of the classic NP-hard problems. However, solving LPs can be done in polynomial time, which makes them a useful tool to approximate NP-hard problems. Specifically, there are known algorithms to solve LPs and obtain an *extreme point solution*, which is not only optimum, but also has the property that it cannot be written as a convex combination of feasible solutions.

2.4.2 Rounding Linear Programs

One technique that is used to solve problems using LPs is to first define and solve an LP *relaxation* to the problem. An LP is said to be a relaxation to a problem if any feasible solution to the problem can be encoded as a feasible solution to the LP. The easiest way to obtain an LP relaxation to a problem is to formulate the IP that encodes that problem, and then remove the integrality constraints to make it an LP.

After obtaining a relaxation and solving it, we can *round* the solution, that is, we can transform it into a feasible solution to the problem, in a way that increases its value by at most a factor of α. The solution that we obtain is then an α-approximation to the problem, because the value of the LP relaxation is at least as good as the value of the optimum solution. However, the value of the LP may sometimes be better than the value of the optimum solution. This motivates the definition of integrality gap

Definition 2.12. Let L denote an LP, and let $x^* \in \mathbb{R}^n$, $x^I \in \mathbb{Z}^n$ be an optimum solution and optimum integer solution to L, respectively.

The *integrality gap* of the LP is the ratio of the optimum value of an integer solution to the value of the LP, that is,

$$\text{intgap}(L) = \frac{c^T x^I}{c^T x^*}.$$

The best approximation we can get by rounding an LP is given by the integrality gap, since the ratio of any integer solution to the value of the LP is bounded by it.

In some cases, rounding may be accomplished by using the structure of the extreme point solutions for the specific LP. We say a solution is $1/k$-*integral* if all the variables take values in $(1/k)\mathbb{Z}$. An LP is $1/k$-integral if all its extreme point solutions are $1/k$-integral. If a solution (or LP) is 1-integral, we simply say it is *integral*. If a solution (or LP) is 1/2-integral, we say it is *half-integral*. We also define the concept of *integrality gap against* $1/k$-*integral solutions* as the ratio of the value of an optimum integer solution to the value of an optimum $1/k$-integral solution.

PART I

Network Design on Bounded Treewidth Graphs

This part of the thesis is the result of close collaboration with Parinya Chalermsook, Syamantak Das, Guy Even and Bundit Laekhanukit. It is mostly based on an article published in the *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)* [CDL$^+$17], and an article published in the proceedings of *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018)* [CDE$^+$18].

CHAPTER 3

Group Steiner Tree

The *group Steiner tree* (GST) problem was introduced by Reich and Widmayer [RW89] and is by now a cornerstone problem in combinatorial optimization and theoretical computer science, having received considerable attention over the past two decades [BDH+16, CEK06, CP05, DHK14, GKR00, HKK+07, HRZ01, NPS11, RW89]. In this problem, we are given an undirected graph $G = (V, E)$, edge or vertex costs, a root vertex r and a collection of h subsets of vertices, $S_1, \ldots, S_h \subseteq V$, called *groups*. The goal is to find a minimum-cost tree that, for each group S_i, $i \in [h]$, connects r to at least one vertex of S_i.

This problem admits a polynomial time $O(\log n \log h)$-approximation when the graph is a tree, due to the seminal result of Garg, Konjevod and Ravi [GKR00]. Using the work of Bartal [Bar96], and later Fakcharoenphol et al. [FRT04], we can convert any graph with edge lengths into a tree where distances are approximately preserved. This technique, referred to as *metric tree embedding*, leads to an increase of the cost by a factor of $O(\log n)$, which results in an $O(\log^2 n \log h)$-approximation for the problem in general graphs. This is the best possible result using this technique, as there are graphs where the cost of a solution increases by an $\Omega(\log n)$ factor [CG04, GNR+04].

Alternatively, a slightly worse approximation ratio can be achieved combinatorially, that is, avoiding the use of linear programming techniques: Chekuri et al. [CEK06] show a greedy algorithm for trees that achieves an approximation ratio of $O(\log^{1+\varepsilon} n \log h)$, where the running time depends exponentially on $1/\varepsilon$. Using the technique outlined above, this implies a combinatorial $O(\log^{2+\varepsilon} n \log h)$-approximation in polynomial time (for a constant $\varepsilon > 0$). We remark that tree embedding techniques do not currently exist for graphs with vertex costs, and thus there are no known polynomial-time polylogarithmic approximations for GST with vertex costs in general graphs.

On the hardness of approximation side, GST is known to generalize set cover. Set cover is hard to approximate to a factor of $(1-\varepsilon)\ln n$ for any $\varepsilon > 0$ [Fei98], which implies the same hardness for GST. Furthermore, Halperin et al. [HK03, HKK+07] showed that, unless NP $\subseteq$ ZPTIME$\left(n^{\text{polylog}\, n}\right)$, GST has no polynomial (or quasi-polynomial) time $O(\log^{2-\varepsilon} h)$-approximation. This bound is, in some sense, matched by an algorithm by Chekuri and Pál [CP05], which gives an $O(\log^2 h)$-approximation in quasi-polynomial time to GST in general graphs.

Beyond the results outlined above, others have explored variants of the problem: Naor et al. [NPS11] studied the online variant of GST and gave polylogarithmic-competitive algorithms for it; Demaine et al. [DHK14], and later Bateni et al. [BDH+16], focused on a specific subset of instances of pratical value, in which the graph is planar, and each group is concentrated in a face. These works culminated in a PTAS for this specialization of the problem. Regarding parameterized algorithms for the problem, little is known: the exact solution can be computed with running time $O(2^h \,\text{poly}(n))$ using dynamic

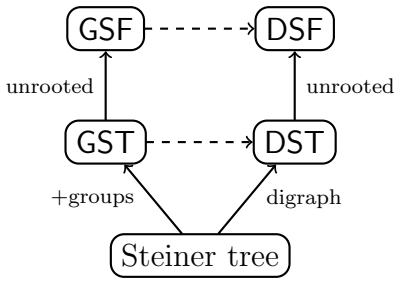

Figure 3.1: Relation between various network design problems. An arrow $A \to B$ means that B is a generalization of A. A dashed arrow indicates that A can be solved by a reduction to B.

programming [DW71]; Jones et al. study the problem on sparse graphs, parameterized by the number of non-terminals in the solution [JLR+17].

The main open question regarding the approximation of GST is whether we can achieve the same approximation ratio in general graphs as we can in trees. Current polynomial-time techniques, which rely on metric tree embeddings, cannot be improved: there are graphs with treewidth 2 which cannot be embedded into a tree without losing a factor of $\Omega(\log n)$ [CG04, GNR+04]. Furthermore, no such techniques are known for graphs with vertex costs. This suggests that, if we want to improve the approximation algorithms for GST, new techniques are needed.

Group and directed network design problems Many variants and generalizations of GST are studied in the literature (see e.g. [He13]). One closely related problem is the *directed Steiner tree* problem, which generalizes GST by encoding groups as sink vertices, with arcs from every vertex in a group. In this work, we are also interested in two other problems, group Steiner forest and directed Steiner forest, which generalize group Steiner tree and directed Steiner tree, respectively. Figure 3.1 represents the relation between all of the problems described below.

In the *directed Steiner tree* problem (DST) we are given a directed graph with edge costs, a root r and a set of h *terminals* T. The goal of the problem is to find the minimum-cost tree (out-arborescence) that connects the root to all the terminals. The DST problem generalizes GST via a simple reduction: we start from the input graph of GST, and turn its edges onto pairs of arcs in each direction; afterwards, one vertex for each group is created, and arcs are added from each terminal in the group to the corresponding vertex for the group, with cost 0. This reduction implies a lower bound of $\Omega(\log^{2-\varepsilon} h)$ for the approximation of DST; furthermore, many algorithms for the two problems share techniques, and sometimes are just slight adaptations of each other (see e.g. [HRZ01, CCC+99, CEK06, GLL19]). For a long time, the best approximation ratio for DST was $O(h^{\varepsilon}\varepsilon^{-2}\log h)$, due to the work of Charikar et al. [CCC+99]. If quasi-polynomial running time is allowed, their result implies an $O(\log^3 h)$-approximation. Very recently, Grandoni et al. [GLL19] improved the approximation ratio in quasi-polynomial running time to $O(\log^2 h/\log\log h)$, which is tight under the *projection game conjecture* [GLL19, HK03].

The *group Steiner forest* problem (GSF) is a generalization of GST that has also

received significant attention. In this problem, we are given a graph G and a collection of pairs of groups $\{(A_i, B_i)\}_{i\in[h]}$ (where $A_i, B_i \subseteq V$ are subsets of vertices). The goal is to find a minimum-cost subgraph $H \subseteq G$ that, for every $i \in [h]$, contains a path from some vertex $a \in A_i$ to some vertex $b \in B_i$.

Most of the research for GSF actually applies for a more general problem, the *directed Steiner forest* problem (DSF). DSF is a computationally harder problem, but as with GST and DST, it is possible to use the techniques developed for DSF (see, e.g., [BBM+13, CCC+99, CEG+11, CDK+17, FKN12]), together with tools available only for undirected graphs, to obtain polylogarithmic approximation algorithms for GSF. Notably, Chekuri et al. [CEG+11] gave a $O(\log^2 n \log^2 h)$-approximation algorithm for GSF that runs in polynomial time. This algorithm is obtained by using the known results for GST as a subroutine.

In the DSF problem, we are given a similar input to GSF, but the input graph is directed and the source-sink pairs are pairs of vertices. On general graphs, DSF has been shown to be as hard as the *label-cover* problem [DK99], which thus rules out polylogarithmic approximation guarantees unless NP $\subseteq$ DTIME($n^{\text{polylog}(n)}$). The early results for DSF gave algorithms whose approximation ratio depends on h, the number of source-sink pairs, while approximation ratios in terms of n, the number of vertices, were still $\Omega(n)$. The first algorithm that broke the bound of $\Omega(n)$ was given by Feldman, Kortsarz and Nutov [FKN12], with an approximation ratio of $n^{4/5+\varepsilon}$, for any $\varepsilon > 0$. This bound was later improved to $n^{2/3+\varepsilon}$ [BBM+13], $n^{3/5+\varepsilon}$ for unweighted graphs [CDK+17], and recently, $n^{0.5778+\varepsilon}$ for unweighted graphs [AB18].

Generalizations of GSF and GST to higher connectivity have also been studied [CGL15, GKR10, KKN12], see Chapter 4 for details.

Related work The approximation algorithm for GST in general graphs uses probabilistic metric tree embedding techniques, which approximate distances in a graph by a distribution over trees. For the purpose of approximating GST, we are interested in tree embeddings for which distances in the resulting trees increase by a small factor (called the *distortion*).

This technique was first introduced by Karp [Kar89], who presented a probabilistic embedding of cycles. Alon et al. [AKP+95] extended this technique to general metrics, with expected distortion of $2^{O(\sqrt{\log n \log\log n})}$. This factor was then improved by Bartal to $O(\log^2 n)$ [Bar96], and then to $O(\log n \log\log n)$ [Bar98], and finally to $O(\log n)$ by Fakcharoenphol et al. [FRT04].

This upper bound on distortion is known to be tight, as there are expander graphs for which any tree embedding must have distortion $\Omega(\log n)$. Gupta et al. [GNR+04] show that this lower bound also applies for tree embeddings of series-parallel graphs, which have treewidth 2. Carroll and Goel [CG04] expand on this result by showing the same distortion of $\Omega(\log n)$ when embedding graphs of bounded treewidth to families of graphs excluding a small fixed minor.

One of the main ideas in our algorithm is to formulate an LP to obtain a solution from a dynamic program to the Steiner tree problem. The technique of formulating an LP from a dynamic programming has been used before: Martin et al. [MRC90] showed that dynamic programs can be formulated as integral LPs (see also [Bü11, dFR01, d'E63, Man60] for stochastic dynamic programs). Gupta et al. [GTW13] also use a similar technique to

approximate the sparsest-cut problem. However, their algorithm performs rounding on sets of vertices simultaneously, while we embed the graph into a tree via the dynamic programming table, making our technique more general. Dynamic programs for Steiner tree on bounded treewidth were first presented by Chimani et al. [CMZ12], with running time $O\big(w^{O(w)}n\big)$, and later improved to $O\big(2^{O(w)}n\big)$ using several different techniques [BCK+15, CNP+11, FBN15].

Our results Our goal is to improve the best approximation factor achievable in polynomial time, from the current $O(\log^2 n \log h)$ to the factor $O(\log n \log h)$. We chose bounded-treewidth graphs as a first step towards this goal: not only do many results for trees generalize to this setting; but we also know that distortion for tree-embedding techniques is $\Theta(\log n)$ in the worst case [GNR+04].

We show that on bounded-treewidth graphs it is possible to surpass the results obtained using tree embedding techniques: we present an $O(\log n \log h)$-approximation algorithm for GST with running time $n^{O(\mathrm{tw}(G)\log \mathrm{tw}(G))}$, which is polynomial in n for constant treewidth. Our result extends to GST on graphs with vertex costs, with the same approximation guarantees and a running time of $n^{O(\mathrm{tw}(G)^2)}$.

The main technical contribution in this section is a framework that allows us to formulate GST as a dynamic program with extra constraints; and then solve this problem by rounding a linear program. This formulation is very similar to GST on trees, so we can solve it using the known algorithm by Garg et al. [GKR00]. In Chapter 4 we show some more uses of this technique. We expect that our framework can be used to solve GST, or even other problems, in further settings.

Organization The organization of this chapter is as follows: in Section 3.1 we introduce the formalism of the problems that we present in this chapter; in Section 3.2 we give a summary of the LP for GST and the algorithm of Garg et al. [GKR00]; in Section 3.3 we present a framework that allows us to more easily reason about the connectivity of graphs in bounded-treewidth; in Section 3.4 we show how to use this framework to solve group Steiner tree in bounded-treewidth graphs. Finally, we show some extensions to group Steiner forest and directed Steiner forest in Sections 3.5 and 3.6.

3.1 Problem Definitions and Results

For ease of presentation, we focus on the case of edge costs. The problems can be easily formulated also for vertex costs, and all of the results apply to both settings. When the use of vertex costs requires different techniques, we present these techniques and show how to apply them.

Problem 3.1: Group Steiner Tree (GST).

- Instance: $\big(G, c, r, \mathcal{S}\big)$, where
  - $G = (V, E)$ is a graph with edge costs $c : E \to \mathbb{R}$;
  - $r \in V$ is the *root* of the instance;
  - $\mathcal{S} = \{S_i\}_{i\in[h]}$ is a collection of h *groups* $S_i \subseteq V$, $i \in [h]$.
- Solution: a tree $F \subseteq E$ that, for every $i \in [h]$, connects r to some $v_i \in S_i$.

- GOAL: minimize the cost of F, $c(F) = \sum_{e \in F} c_e$.

Problem 3.2: Group Steiner Forest (GSF).

- INSTANCE: $\big(G, c, \mathcal{P}\big)$, where
  - $G = (V, E)$ is a graph with edge costs $c : E \to \mathbb{R}$;
  - $\mathcal{P} = \{(A_i, B_i)\}_{i \in [h]}$ is a collection of h *group pairs* $(A_i, B_i) \subseteq V \times V$, $i \in [h]$.
- SOLUTION: a forest $F \subseteq E$ that, for every $i \in [h]$, connects some $a_i \in A_i$ to some $b_i \in B_i$.
- GOAL: minimize the cost of F, $c(F) = \sum_{e \in F} c_e$.

Problem 3.3: Directed Steiner Forest (DSF).

- INSTANCE: $\big(G, c, P\big)$, where
  - $G = (V, E)$ is a directed graph with edge costs $c : E \to \mathbb{R}$;
  - $P = \{(a_i, b_i)\}_{i \in [h]}$ is a collection of h *terminal pairs* $(a_i, b_i) \in V \times V$, $i \in [h]$.
- SOLUTION: a forest $F \subseteq E$ that, for every $i \in [h]$, connects a_i to b_i.
- GOAL: minimize the cost of F, $c(F) = \sum_{e \in F} c_e$.

Theorem 3.4. *There is an $O(\log n \log h)$-approximation algorithm for GST with running time $n^{O(w \log w)}$, where w is the treewidth of the input graph.*

Theorem 3.5. *There is an $O(\log n \log h)$-approximation algorithm for GST on a graph with vertex costs, with running time $n^{O(w^2)}$, where w is the treewidth of the input graph.*

Theorem 3.6. *There is an $O(\log n \log^2 h)$-approximation algorithm for GSF with running time $n^{O(w \log w)}$ ($n^{O(w^2)}$ for vertex costs), where w is the treewidth of the input graph.*

Theorem 3.7. *There is an $O(w \log^2 n \log^2 h)$-approximation algorithm for DSF with running time $n^{O(w^2)}$, where w is the treewidth of the undirected version of the input graph.*

3.2 The Algorithm of Garg, Konjevod and Ravi

The algorithm of Garg, Konjevod and Ravi [GKR00] (GKR from now on) works by solving an LP relaxation of the problem, and then rounding the LP solution repeatedly. The solution obtained in each iteration costs the same as the LP solution in expectation, but covers each group with probability $\Omega(1/\log n)$. By repeating this experiment sufficiently ($O(\log n \log h)$ times), and taking the union of all of the solutions obtained, we can guarantee that all the groups are covered with high probability, while increasing the cost by a factor equal to the number of experiments.

This algorithm can then be used in conjunction with probabilistic tree embedding techniques [FRT04] to obtain an approximation algorithm for general graphs. We will start by presenting the reduction from general graphs to trees, then the standard LP for GST, which is used in GKR, and finally show the rounding algorithm and analyze its correctness. All of the results in this section are based on the paper by Garg et al. [GKR00], and are presented here for completeness.

3.2.1 Reduction to Trees

In this section, we show how to use probabilistic tree embeddings to solve GST on general graphs, by losing a factor of $O(\log n)$. To prove this result, formalized in Lemma 3.9, the theorem of Fakcharoenphol et al. [FRT04] (Theorem 3.8) is used. We provide a proof of the reduction for completeness.

Theorem 3.8 ([FRT04])**.** *Let (V, d) be an arbitrary metric. There is a distribution $\mathcal{D}$ over tree metrics that $O(\log n)$-approximates the metric (V, d), that is, for any $u, v \in V$:*

$$\mathrm{E}_{(T,d_T)\sim\mathcal{D}}\left[d_T(u,v)\right] \leq O(\log n)\, d(u,v) \tag{i}$$

$$d(u,v) \leq d_T(u,v) \qquad \forall (T,d_T) \in \mathcal{D} \tag{ii}$$

Furthermore, there is a polynomial-time algorithm that samples a tree metric (T, d_T) according to distribution $\mathcal{D}$.

Lemma 3.9 ([FRT04, GKR00])**.** *If there is an $\alpha(n, h)$-approximation algorithm for GST on trees, there is an $O(\alpha(n, h) \log n)$-approximation algorithm for GST on general graphs.*

Proof. Let $\left(G, c, r, \{S_i\}_{i\in[h]}\right)$ be an instance of GST, and let F^* be the optimum solution. Let $(V(G), d_c)$ be the shortest path metric of G with edge costs c, where $d_c(u, v)$ represents the minimum total cost $c(P)$ of a path P between u and v in G.

To obtain an $O(\alpha(n, h) \log n)$-approximation (in expectation) for this instance, we do the following:

- Sample a tree metric (T, d_T) according to distribution $\mathcal{D}$ that $O(\log n)$-approximates (G, d_c);
- Compute an $\alpha(n, h)$-approximate solution F_T for T with edge costs given by d_T, and the same root r and groups $\{S_i\}_{i\in[h]}$;
- Obtain a solution F for the original problem by adding to F, for each edge $uv \in F_T$, the shortest path from u to v in G (according to d_c).

We will now show that F is a feasible solution, and an $O(\alpha(n, h) \log n)$-approximate solution in expectation. Notice that, if two vertices u and v are connected in F_T, they must be connected in F as well: there must be a path $u = w_0, w_1, \ldots, w_\ell = v$ in F_T, and we add to F a path connecting each of the pairs (w_{i-1}, w_i), $i \in [\ell]$. Therefore, the concatenation of the paths in F for the pairs (w_{i-1}, w_i) must connect u to v in F. We conclude that F connects r to every group, since F_T must also connect every group.

In order to prove the approximation guarantees, we need to prove that $c(F) \leq c(F_T)$ and that $\mathrm{E}[\mathrm{opt}(T)] \leq O(\log n)\, \mathrm{opt}(G)$. Since we know that $c(F_T) \leq \alpha(n, h)\, \mathrm{opt}(T)$ by definition, the lemma follows. To prove the first claim, it is sufficient to use Property (ii) of Theorem 3.8, since each edge in $uv \in F_T$ gets replaced by a shortest path in F, and $d_c(u, v) \leq d_T(u, v)$. This implies that the union of all of these shortest paths must cost at most as much as all the edges in F_T.

To prove that $\mathrm{E}[\mathrm{opt}(T)] \leq O(\log n)\, \mathrm{opt}(G)$, notice that we can obtain a solution F_T^* for T by taking each edge $uv \in F^*$ and adding to F_T^* the shortest path between u and v

in (T, d_T). We denote the cost of F_T^* as $d_T(F_T^*)$. Using Property (i) of Theorem 3.8 and linearity of expectation, we conclude that

$$\mathrm{E}_{(T,d_T)\sim\mathcal{D}}\left[d_T(F_T^*)\right] \leq \sum_{uv\in F^*} \mathrm{E}\left[d_T(u,v)\right] \leq \sum_{uv\in F^*} O(\log n) d_c(u,v) = O(\log n) c(F^*)$$

We conclude that the expected cost of F is at most

$$\begin{aligned}\mathrm{E}_{(T,d_T)\sim\mathcal{D}}\left[c(F)\right] &\leq \mathrm{E}_{(T,d_T)\sim\mathcal{D}}\left[c(F_T)\right] \\ &\leq \alpha(n,h)\,\mathrm{E}_{(T,d_T)\sim\mathcal{D}}\left[\mathrm{opt}(T)\right] \\ &\leq O\big(\alpha(n,h)\log n\big)\,\mathrm{opt}(G)\end{aligned}$$ □

3.2.2 Linear Program for GST

There are several equivalent LPs for GST. The most common one is based on the standard network design LP; we refer to it as the cut-LP for GST. In this LP, there are variables x_e, $e \in E$, which indicate whether edge e is in the solution ($x_e = 1$) or not ($x_e = 0$).

Definition 3.10 (Cut-LP for GST – Cut-LP)**.**

$$\begin{aligned}\min \quad & \sum_{e\in E} c_e x_e \\ \text{s. t.} \quad & \sum_{e\in\delta(U)} x_e \geq 1 && \forall\, i \in [h], U \subseteq V \setminus S_i : r \in U \\ & x_e \geq 0 && \forall\, e \in E\end{aligned}$$

The intuitive meaning of Cut-LP is the following: if we consider any cut that separates the root r from a group S_i, there must be an edge crossing the cut, as otherwise r and S_i would not be connected. Despite its simplicity, it has an exponential number of constraints, which is sometimes an undesirable property. The flow-LP addresses this issue by formulating GST as the problem of finding a tree supporting a unit flow from the root to each group. We consider the flows to be directed; $\mathrm{out}(v)$ and $\mathrm{in}(v)$ refer to the sets of heads of arcs from v and tails of arcs to v, respectively.

Definition 3.11 (Flow-LP for GST – Flow-LP)**.**

$$\begin{aligned}\min \quad & \sum_{e\in E} c_e x_e \\ \text{s. t.} \quad & f_e^i \leq x_e && \forall\, i \in [h], e \in E \\ & f^i(\mathrm{in}(v)) - f^i(\mathrm{out}(v)) = 0 && \forall\, i \in [h], v \in V \setminus (r \cup S_i) \\ & f^i(\mathrm{in}(v)) - f^i(\mathrm{out}(v)) \geq 0 && \forall\, i \in [h], v \in S_i \\ & f^i(\mathrm{out}(r)) = 1 && \forall\, i \in [h] \\ & f_e^i \geq 0 && \forall\, i \in [h], e \in E \\ & x_e \geq 0 && \forall\, e \in E\end{aligned}$$

To formulate Flow-LP, we introduce additional variables f_e^i for every group S_i and every edge $e \in E$. f^i represents a flow from the root to the elements of the group S_i. In the integral case ($f_e^i \in \{0, 1\}$), this corresponds to a path from the root to a single element of S_i. The solution is then the set of edges necessary to support this flow. The constraints support this idea: the first constraint states that x_e must be at least as much as the flow going through e for any of the groups.

The remaining constraints are commonly known as *flow conservation* constraints, and simply force the ingoing flow to equal the outgoing flow, for each flow f^i. The two exceptions are the root (where there is an outgoing flow of 1), and the elements of the group S_i, that may receive flow, and thus may have outgoing flow smaller than the ingoing flow. Since flow conservation constraints apply to all vertices, including leaves, the flow leaving the root has to be fully received by elements of the group, that is, there is a unit flow between the root and each group.

We now show that both of the LPs above are relaxations to the GST problem.

Claim 3.12. *Both* Cut-LP *and* Flow-LP *are relaxations to GST. Furthermore, any integral solution of either LP corresponds to a feasible solution to GST.*

Proof. To prove the first part of the claim, it is sufficient to consider a solution F and show how to construct an LP solution corresponding to it. In either case, we can take x to be the indicator vector of a solution ($x_e = 1$ if and only if e is in the solution). Notice that the cost of x equals the cost of F, $c^T x = c(F)$. For Flow-LP, we take f^i to be the indicator vector of a path from the root to an element of S_i covered by F.

Regarding Cut-LP, for any cut $(U, V \setminus U)$ that separates r and a group S_i, there must be an edge e in F crossing the cut, as F contains a path from r to some $v_i \in S_i$. Since $e \in F$, $x_e = 1$, and therefore the constraints are satisfied. For Flow-LP, it is easy to see that the indicator vector of a path starting at r and ending at $v_i \in S_i$ must satisfy the flow conservation constraints. It must also be that $f_e^i \leq x_e$, as the path is contained in the solution F. In any case, the constructed solutions are feasible for both Cut-LP and Flow-LP.

The second part of the claim is an even stronger statement: it says that if there is an integral solution to one of the LPs, then we can convert it into a solution to GST with the same cost. Let x be an integral solution to Cut-LP. We will show how to convert it to an integral solution of Flow-LP, and then from there to a solution of GST.

Notice that for every cut separating the root and any group S_i, the sum of x_e for edges e crossing the cut is at least 1. Another way of stating this is that if we consider x as a vector of capacities of the edges, the min-cut separating r and S_i is at least 1, for every group $i \in [h]$. By the max-flow min-cut theorem [FF56], this implies that there is a unit flow f^i for every group from the root r to S_i. Since the capacities are integral, we can further assume that this flow is integral [FF56]. By definition, f^i satisfies flow conservation, and does not exceed the capacities x. This transformation does not change the cost of the solution. Now, to transform it into a feasible solution for GST, simply take $F = \{e \in E : x_e = 1\}$. Notice that $c(F) = c^T x$, and F must connect every group, as it contains a unit flow from the root to a vertex in each group. Therefore, integral solutions to the LPs above can be converted into a feasible solution to the problem. □

3.2.3 Rounding Algorithm

In this section, we describe the GKR algorithm, which $O(\log n \log h)$-approximates GST on trees. We present the algorithm and show its correctness; furthermore, we show a slight generalization of the main lemma in the analysis, which we require in Section 3.5.

Given a tree instance of GST, the GKR algorithm computes an optimal LP solution and rounds it repeatedly to ensure that every group is connected. The key step in the algorithm is the rounding, which produces a solution with the same expected cost as the LP solution, and where each group is connected to the root with probability $\Omega(1/\log n)$. We refer to this step as RoundGKR and state its properties in Lemma 3.13. We first show how to use this lemma to obtain the desired $O(\log n \log h)$-approximation; we then present a proof of the lemma based on the original work of Garg et al. [GKR00], as well as the elegant analysis of Rothvoß for the directed Steiner tree problem [Rot11].

The GKR algorithm starts by computing x, an optimal solution to the LP. Then, it rounds the solution independently $\ell \log n \log h$ times, for some integer ℓ, takes the union of all obtained solutions, and, if needed, finally connects any unconnected groups using a shortest path. Given an LP solution $x \in [0,1]^E$, RoundGKR computes a solution F as follows: for every edge e adjacent to the root, e is added to F independently at random with probability x_e; any other edge is added to F with probability $x_e/x_{p(e)}$ given that $p(e) \in F$. We assume that $x_e \leq x_{p(e)}$ so that probabilities are well defined; any solution can be preprocessed such that it satisfies this constraint, without affecting its feasibility. See Algorithm A.1 for more details on the GKR algorithm.

Let y_j, $j \in [h]$, represent the value of the minimum cut (or max flow) separating the root r from group S_j (with a maximum of 1). Formally,

$$y_j = \min\left(\{x(\delta(U)) \mid U \subseteq V \setminus S_j, r \in U\}, 1\right).$$

If x is a feasible solution for the Cut-LP, it follows that $y_j = 1$ for all groups S_j, $j \in [h]$. However, the lemma we show applies even when $y_j < 1$.

Lemma 3.13. *Let G be a tree, $x \in [0,1]^{E(G)}$ be any LP solution satisfying $x_e \leq x_{p(e)}$, $e \in E(G)$, and y_i be defined as above.*

RoundGKR(G, r, x) outputs a subtree $F \subseteq G$ with expected cost $c^T x$. Each group S_j is connected by F with probability at least $y_j/(\min(\text{height}(G), 12 \log |S_j|))$.

Theorem 3.14. *Let $\mathcal{I} = \left(G, c, r, \{S_i\}_{i\in[h]}\right)$ be a GST instance, where G is a tree.*

GKR computes a feasible solution to $\mathcal{I}$ with expected cost $O(\log n \log h)\, c(\text{OPT})$ in polynomial time, where $c(\text{OPT})$ is the cost of the optimal solution.

Proof. Let (x, f) be the optimum LP solution to Flow-LP, and let $\ell \geq 36$. We remark that $x_e \leq x_{p(e)}$: since all of the flow going through e goes through $p(e)$ too, an optimum solution will set $x_e = \max_i f_e^i \leq x_{p(e)}$. Furthermore,

$$\min(\text{height}(G), 12 \log |S_i|) \leq 12 \log |S_i| \leq 12 \log n$$

Let F_i be a random variable representing the output of the i-th call to RoundGKR, and $\hat{F} = \bigcup_{i\in[\ell \log n \log h]} F_i$. Fix a group S_j, $j \in [h]$. We represent as $S_j \cap \hat{F} = \emptyset$ the event

that S_j is not connected to the root in $\hat{F}$.

$$\begin{aligned}\Pr[S_j \cap \hat{F} = \emptyset] &\leq \left(1 - \frac{1}{12 \log n}\right)^{\ell \log n \log h} \\ &\leq e^{-\ell \log h / 12} \\ &\leq h^{-3}\end{aligned}$$

The first inequality follows from applying Lemma 3.13.

We conclude that the probability that a group is not connected to the root is at most h^{-3}. For groups that are not connected to the root, the algorithm connects them using a shortest path which costs at most $c(\mathrm{OPT}(\mathcal{I}))$ (the cost of the optimum solution cannot be lower than the shortest path to any single group). We can now bound the expected cost of $\hat{F}$:

$$\begin{aligned}\mathrm{E}[c(\hat{F})] &\leq \sum_{i=1}^{\ell \log n \log h} c(F_i) + \sum_{j \in [h]} \Pr[S_j \cap \hat{F} = \emptyset]\, c(\mathrm{OPT}(\mathcal{I})) \\ &\leq (\ell \log n \log h)\, c^T x + h \frac{1}{h^3}\, c(\mathrm{OPT}(\mathcal{I})) \\ &\leq \left(\ell \log n \log h + \frac{1}{h^2}\right) c(\mathrm{OPT}(\mathcal{I})) \\ &= O(\log n \log h)\, c(\mathrm{OPT}(\mathcal{I}))\end{aligned}$$

□

All that is left is to prove Lemma 3.13.

Proof of Lemma 3.13. Let F be a random variable representing the output of the function call to RoundGKR(G, r, x). The following claim shows that the probability that $e \in F$ is exactly x_e for all edges. By linearity of expectation this implies that

$$\mathrm{E}[c(F)] = \sum_{e \in E(G)} c_e \Pr[e \in F] = c^T x$$

Claim 3.15. *For any $e \in E(G)$, $\Pr[e \in F] = x_e$.*

Proof. We prove the claim by top-down induction. For edges adjacent to the root, the probability that $e \in F$ is x_e by the algorithm. For any other edge,

$$\Pr[e \in F] = \Pr[p(e) \in F] \Pr\left[e \in F \mid p(e) \in F\right] = x_{p(e)} \frac{x_e}{x_{p(e)}} = x_e$$

The second step follows by induction hypothesis, using the probabilities in the algorithm. □

We will now provide a lower bound for the probability that a group is connected. In order to simplify the analysis, we use the following lemma from Garg et al. [GKR00].

Lemma 3.16 (Lemma 3.3 in [GKR00], adapted)**.** *If x, x' are solutions that differ only in the capacity of a single edge e, and $x_e \geq x'_e$, then for any group S_i, the probability of including a vertex from S_i is no greater for x' than for x.*

Fix a group $S = S_j$, $j \in [h]$. Let $y = y_j$ and $H = \text{height}(G)$. We will start by showing that

$$\Pr[S \cap F \neq \emptyset] \geq \frac{y}{H}$$

where $S \cap F \neq \emptyset$ represents the event that F connects the root to S.

Let $f \leq x$ be a flow from r to S of value y. Since the minimum cut separating r from S has value (at least) y by definition, such a flow must exist. Using Lemma 3.16, we can iteratively reduce the values of the edges until we have $x' = f$, without increasing the probability of connecting S. This implies that any lower bound on the probability of success for f applies to x as well. From now on, we assume that $x = f$.

Let $X = |S \cap F|$ be a random variable that counts the number of terminals in S connected by the solution. Our goal is to lower bound $\Pr[X \geq 1] = \Pr[S \cap F \neq \emptyset]$. In order to bound this probability, we use the insight of Rothvoß [Rot11]:

$$\mathrm{E}[X] = 0 \Pr[X = 0] + \mathrm{E}[X \mid X \geq 1] \Pr[X \geq 1] = \mathrm{E}[X \mid X \geq 1] \Pr[X \geq 1]$$

Rearranging, we obtain

$$\Pr[X \geq 1] = \frac{\mathrm{E}[X]}{\mathrm{E}[X \mid X \geq 1]}$$

We first bound $\mathrm{E}[X]$. Using linearity of expectation,

$$\mathrm{E}[X] = \sum_{t \in S} \Pr[t \in F] = \sum_{t \in S} \Pr[p(t)t \in F] = \sum_{t \in S} x_{p(t)t} \geq y$$

The third step follows from Claim 3.15, and the last two steps from the assumption that $x = f$ and the fact that f is a flow of value y.

In order to upper bound $\mathrm{E}[X \mid X \geq 1]$, we use the following identity:

Lemma 3.17. *Let $(\Omega, \mathcal{F}, P)$ be a probability space, $A_i \in \mathcal{F}$ be events with indicator variable X_i and $X = \sum_i X_i$.*

Then $\mathrm{E}[X \mid X \geq 1] \leq \max_i \mathrm{E}[X \mid A_i]$.

The proof of this lemma follows by simple probabilistic arguments, and is deferred to Appendix A.1.1.

Consider the probability space derived from the execution of RoundGKR(G, r, x). Applying Lemma 3.17 to the events $\{t \in F\}$ for all $t \in S$, we obtain that

$$\mathrm{E}[X \mid X \geq 1] \leq \max_{t \in S} \mathrm{E}[X \mid t \in F]$$

Therefore, it is sufficient to prove that, for any $t \in S$, $\mathrm{E}[X \mid t \in F] \leq H$.

Let $t \in S$, let $t^{(\ell)}$ be the ℓ-th ancestor of t, and $C^{(\ell)}$ be the vertices $v \in S$ such that $t^{(\ell)}$ is the lowest common ancestor of v and t. We will prove that each set $C^{(\ell)}$ contributes a sum of at most 1 to the expectation, and hence the expectation is bounded by the height of the tree. For simplicity of notation, we write $x(v)$ to mean x_e, where e is the edge connecting v to its parent.

$$\begin{aligned}
\mathrm{E}[X \mid t \in F] &= \sum_{v \in S} \Pr[v \in F \mid t \in F] \\
&= \sum_{\ell \in [H]} \sum_{v \in C^{(\ell)}} \Pr[v \in F \mid t^{(\ell)} \in F] \\
&= \sum_{\ell \in [H]} \sum_{v \in C^{(\ell)}} \frac{x(v)}{x(t^{(\ell)})} \qquad (1) \\
&= \sum_{\ell \in [H]} \frac{1}{x(t^{(\ell)})} \sum_{v \in C^{(\ell)}} x(v) \\
&\leq \sum_{\ell \in [H]} \frac{1}{x(t^{(\ell)})} x(t^{(\ell)}) \qquad (2) \\
&= H
\end{aligned}$$

Step (1) follows by using Claim 3.15 on the path from v to $t^{(\ell)}$. Step (2) follows because x is a flow, and hence all of the flow into $C^{(\ell)}$ flows through $t^{(\ell)}$.

We obtain that

$$\Pr[X \geq 1] = \frac{\mathrm{E}[X]}{\mathrm{E}[X \mid X \geq 1]} \geq \frac{y}{H}$$

as desired.

We now repeat the proof with slight changes to show that

$$\Pr[X \geq 1] \geq \frac{y}{12 \log |S|}$$

Using Lemma 3.16, we further reduce x to be a flow from r to S where the flow to each vertex in S is either 0 or greater than $y/(2|S|)$. We achieve this by decreasing the flow along the path from the root to any vertex receiving at most $y/(2|S|)$; in doing so, we lose at most $y/2$ units of flow in total, and hence x is a flow of value at least $y/2$.

As above, we have

$$\mathrm{E}[X] = \sum_{t \in S} x_{p(t)t} \geq y/2$$

We now prove that, for any $t \in S$, $\mathrm{E}[X \mid t \in F] \leq 2(\log |S| + 2)$. Let

$$\hat{C}^{(\ell)} = \left\{ v \in S \;\middle|\; x(\mathrm{lca}(t, v)) \in \left]2^{-\ell-1}, 2^{-\ell}\right] \right\},$$

for $\ell \in \{0, \ldots, \lceil \log |S| \rceil\}$. In other words, we group the ancestors by flow passing through them. We remark that the union of $\hat{C}^{(\ell)}$ covers the range $\left]2^{-\lceil \log |S| \rceil - 1}, 1\right] \supseteq \left]1/(2|S|), 1\right]$, which includes all the terminals with incoming flow in x.

By a similar argument as in the first case, we get

$$\begin{aligned}
\mathrm{E}[X \mid t \in F] &= \sum_{v \in S} \Pr[v \in F \mid t \in F] \\
&= \sum_{\ell \in [H]} \sum_{v \in \hat{C}^{(\ell)}} \Pr[v \in F \mid \mathrm{lca}(t, v) \in F] \\
&= \sum_{\ell \in [H]} \sum_{v \in \hat{C}^{(\ell)}} \frac{x(v)}{x(\mathrm{lca}(t, v))} \\
&\leq \sum_{\ell \in [H]} \frac{1}{2^{-\ell-1}} \sum_{v \in \hat{C}^{(\ell)}} x(v) \\
&= \sum_{\ell \in [H]} 2^{\ell+1}\, 2^{-\ell} \\
&\leq 2(\lceil \log |S| \rceil + 1)
\end{aligned}$$

Applying the formula for $\Pr X \geq 1$, we conclude that

$$\Pr[X \geq 1] = \frac{\mathrm{E}[X]}{\mathrm{E}[X \mid X \geq 1]} \geq \frac{y/2}{2(\log |S| + 2)} \geq \frac{y}{12 \log |S|}$$

This concludes the proof. □

3.3 Connectivity for GST on Bounded-Treewidth Graphs

Many problems, such as Steiner tree, can be solved easily using dynamic programming when the input graph is a tree. This dynamic programming algorithm can then be adapted to solve the problem on bounded-treewidth graphs as well, by adjusting the "boundary conditions" of the subproblems (see Section 3.3.4 for more details).

Unfortunately, it is more challenging to apply dynamic programming to GST. In fact, there is no known use of this technique to approximate the problem in polynomial time, even on trees. The main obstacle to the use of dynamic programming or other recursive techniques for GST is that subproblems cannot be solved independently, as vertices of each group can be present in multiple subproblems. While Chekuri et al. successfully used recursion to approximate GST on trees [CEK06], they did so by carefully balancing the cost of the solution with the number of groups covered in each subtree. It is not clear how to generalize this approach to bounded-treewidth graphs.

In this section, we are going to present the ideas necessary to transform GST into an instance of a different problem, which we can approximate. This transformation produces an instance whose size depends exponentially on the treewidth of the input graph. We start by giving a different perspective on dynamic programming in Section 3.3.1, which will help us develop understanding on how the algorithm works. In Section 3.3.2 we define the concept of connection sets and prove some properties of these objects. In Section 3.3.3 we present a result that allows us to easily argue about the connectivity of a solution in a dynamic program, and in Section 3.3.4, we show how to use this result to write a simple dynamic program for Steiner tree.

While dynamic programming algorithms for Steiner tree were known before [BCK+15, CMZ12, CNP+11, FBN15], our techniques are more easily used to solve GST, and can be generalized to other problems, including high-connectivity variants (see Chapter 4).

3.3.1 A Different Perspective on Dynamic Programming

Dynamic programming is a technique that is frequently used to solve a problem when it has *optimal substructure*, that is, when an optimal solution to a problem can be constructed from optimal solutions to its subproblems. By computing and storing the solution to each subproblem once, dynamic programming can compute the optimal solution in time polynomial in the number of subproblems, whereas using recursion can take exponential time for even the simplest problems.

Throughout this thesis, we are mostly interested in a specific (but very general) type of dynamic programming, in which subproblems are specified by a *subinstance* and a *state*. A subinstance is simply a smaller part of the original instance (e.g. a subgraph of the original graph), the idea being that we can compute solutions to a larger subinstance by combining solutions to smaller subinstances. In many situations, we do not get a feasible solution by simply combining solutions to smaller subinstances. The state specifies restrictions on the solutions in the subinstance, so that solutions can be combined. In other words, the state encodes an equivalence class of solutions for a subinstance, in the sense that all of these solutions corresponding to a state can be used interchangeably when combining them with solutions to other subinstances.

If the number of subinstances and the number of possible states for each subinstance are both bounded, we can obtain optimal solutions for each subinstance and each state, starting from solutions to smaller subinstances and combining these to obtain solutions to progressively bigger instances, until we have a solution for the general instance. This process is what we generally know as dynamic programming.

Let us consider a more concrete example, the maximum independent set problem on trees: in this problem, we are given a tree $G = (V, E)$, and we want to find the largest independent set $I \subseteq V$, that is, the largest subset I such that no two vertices in I share an edge of E. Let $r \in V$ be an arbitrary root. We are going to consider as subinstances all of the subtrees G_v (the subtree of G rooted at v), $v \in V$; the state will be a single bit b, which encodes whether v is contained in the solution ($b = 1$) or not ($b = 0$). Implicitly, we are saying that when combining a solution in G_v with others, it is only relevant to know whether v is contained in the solution, and the rest of the solution does not matter. We refer to the problem on G_v with state b as $P(v, b)$. To be more explicit, we can state $P(v, b)$ as the problem of finding a maximum independent set I in G_v such that $v \in I$, in the case that $b = 1$, and otherwise $v \notin I$.

For a vertex v with children $v_1, v_2, \ldots, v_\alpha$, we can obtain the optimal solution for $P(v, 1)$ by taking v together with the union of the solutions for all the subproblems $P(v_i, 0)$. Obtaining the solution for $P(v, 0)$ can be done similarly, by taking the union, for each child v_i of v, of the best solution for either $P(v_i, 0)$ or $P(v_i, 1)$. In any case, the solution to $P(v, b)$ is obtained from a combination $(P(v_1, b_1), P(v_2, b_2), \ldots, P(v_\alpha, b_\alpha))$ of one subproblem for each child. If $b = 0$, we can take arbitrary $b_i \in \{0, 1\}$, but if $b = 1$, we must take $b_i = 0$, $i \in [\alpha]$ (otherwise the solution would contain an edge from v to one of the children).

Since the number of subproblems is $2|V|$, and we can find the optimal solution for $P(v, b)$ efficiently from the solutions for the subproblems, we can solve maximum independent set on trees in polynomial time.

This leads to an abstract view of dynamic programming: a solution to a subproblem is determined by the choice of states for children subinstances, as well as the solutions for each of theses states. To solve a subproblem, we compute all the optimum solutions for each child subinstance and its state; then, using these optimum solutions, we consider all possible combination of states for the children subinstances. Among all of these combined solutions, the best one will be optimal for the subproblem.

When solving GST on bounded-treewidth graphs, we will use this abstraction to describe our algorithm. We will show that we can reformulate GST as the problem of assigning states to the subinstances so that the total cost is minimized and all groups are covered.

3.3.2 Connectivity and Connection Sets

In this section, we introduce the concept of connection sets, as well as some of their properties. Connection sets formalize the notion of connectivity of a graph, and we use them in the presentation and analysis of our algorithm.

Definition 3.18. Let $G = (V, E)$ be a (directed or undirected) graph. A (directed or undirected) *connection set* Λ over V is a subset of $S \times S$ that expresses connectivity in a graph, that is, $(u, v) \in \Lambda$ if there is a path connecting u to v in G.

Formally, a connection set $\Lambda \subseteq V \times V$ is a reflexive, transitive relation on V: $(v, v) \in \Lambda$ for all $v \in V$, since v is connected to itself by the empty path; and concatenating paths between u and w and w and v implies that there is a path between u and v, that is, connection sets are transitive. If a connection set is undirected it must also be symmetric, and thus it is an equivalence relation. Unless otherwise stated, we assume that a connection set is directed if it corresponds to a directed graph or if it is clearly directed from context; we assume it is undirected in all other cases.

The following operators are useful when handling connections sets.

Definition 3.19. Let V be a vertex set, $U \subseteq V$, and $Y \subseteq V \times V$.

The *(reflexive) transitive closure* $\mathrm{tc}(Y)$ of Y is the smallest connection set (reflexive transitive relation) that is a superset of Y.

The *projection* of Y onto U is defined as $Y|_U = Y \cap (U \times U)$.

The transitive closure of a set $Y \subseteq V \times V$ can be obtained by adding to Y all pairs (v, v) for $v \in V$, and all pairs (w, w') for which there is a sequence $(w = w_0, w_1, \ldots, w_q = w')$ such that $(w_{i-1}, w_i) \in Y$ for all $i \in [q]$.

The following properties of connection sets are used throughout our work. For convenience, we assume that a set of undirected edges is given as a set of pairs (u, v) *and* (v, u) for each edge uv whenever working with connection sets.

Lemma 3.20. *Let $G = (V, E)$, $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ be graphs.*

(1) The connection set of G over V is given by $\Lambda = \mathrm{tc}(E)$.

(2) Given two connection sets Λ_1, Λ_2 (for graphs G_1, G_2), $\Lambda = \mathrm{tc}(\Lambda_1 \cup \Lambda_2)$ is the connection set over $V_1 \cup V_2$ of the graph $G' = (V_1 \cup V_2, E_1 \cup E_2)$.

Proof. Let $u, v \in V$. We know that $(u, v) \in \Lambda$ if and only if there is a sequence $(u = w_0, w_1, \ldots, w_q = v)$ such that $(w_{i-1}, w_i) \in E$ for all $i \in [q]$, which is equivalent to there being a u-v-path in G.

For the second property, notice that if $(u, v) \in \Lambda_1$, $(v, w) \in \Lambda_2$, it is not necessarily true that $(u, w) \in \Lambda_1 \cup \Lambda_2$, even though there is a path between u and w in G'. Therefore, we need to take the transitive closure of the union to make it a connection set.

Let $u, v \in V_1 \cup V_2$. We have that $(u, v) \in \mathrm{tc}(\Lambda_1 \cup \Lambda_2)$ if and only if there is a sequence $(u = w_0, w_1, \ldots, w_q = v)$ such that $(w_{i-1}, w_i) \in \Lambda_1 \cup \Lambda_2$ for all $i \in [q]$. If $(w_{i-1}, w_i) \in \Lambda_1 \cup \Lambda_2$, then w_{i-1}, w_i are connected in either G_1 or G_2. By concatenating the paths for each pair (w_{i-1}, w_i), we obtain a path in G'. Similarly, given a u-v-path in G', we can divide it into contiguous subsequences in G_1 or G_2. The endpoints of these contiguous subsequences are connected in G_1 or G_2, and thus the pair is in Λ_1 or Λ_2. Therefore, there is a sequence as in the definition of transitive closure, and therefore, $(u, v) \in \mathrm{tc}(\Lambda_1 \cup \Lambda_2)$. □

The following lemma reasons about the number of possible connection sets over a set of n vertices. In our algorithms, we enumerate connection sets for small graphs, and therefore this lemma will be useful to determine the running time and memory needed.

Lemma 3.21. *Let V be a vertex set, with $n = |V|$.*
The number of possible connection sets of over V is:

- *at most n^n for undirected connection sets*
- *at most 2^{n^2} for directed connection sets*

Proof. An undirected connection set is an equivalence relation, and a directed connection set is a preorder relation. These types of relations are well known and the bounds above are simple to prove.

Since connection sets are subsets of $V \times V$, the number of possible connection sets for a vertex set V is at most 2^{n^2}. For undirected connection sets (equivalence relations), we are essentially partitioning V into equivalence classes. We can bound the number of possible different undirected connection sets by assigning each $v \in V$ to an equivalence class indexed by $[n]$. This will include all the valid undirected connection sets, so the number must be at most n^n. □

3.3.3 Connectivity Lemma

Let $G = (V, E)$ and let $(\mathcal{T}, X)$ be a tree decomposition of G satisfying Properties (3) and (4) of Lemma 2.6, i.e. each internal node has two children, no edge belongs to a leaf bag. These assumptions are not strictly required, but they simplify the presentation of the results. All of the results of this section can be generalized in case these assumptions are not used.

In this section, we will show a key concept that we use to obtain the dynamic program for Steiner tree. This concept can also be applied to fault-tolerant versions of Steiner tree, and allows for a simple analysis of the algorithms.

Consider a dynamic program for Steiner tree. In a feasible solution, all of the terminals must be connected to the root. Since we want to solve the problem in time polynomial in the number of terminals, we would like to avoid enumerating all the terminals covered in a certain subproblem. Intuitively, we would like to somehow invalidate all the subproblems that lead to infeasible solutions, where a terminal is not connected to the root. For this to work, we need to know, for each subproblem, whether each terminal in the corresponding subinstance is connected. This is global information: whether a terminal is connected depends also on edges in different (non-overlapping) subproblems. Think, for instance, about the edges incident to the root; in every subproblem associated with a terminal, we must somehow know that the right edge incident to the root is in the solution. Otherwise, the terminal might not be connected to the root.

We propose a framework to deal with this issue: by adding some information to the subproblems (encoded in the state), and by defining some local compatibility rules, we can ensure that the information contained in each subproblem is valid globally, that is, a set of local rules will enforce global consistency of the information stored in the subproblems. We find this result surprising: with these local rules, the choice of the state for a subinstance indirectly influences *all* of the other subproblems.

More formally, each subinstance corresponds to the subgraph G_t, for some node $t \in \mathcal{T}$ of the tree decomposition. Let $P(t, \Sigma)$ be a subproblem for G_t with state Σ. The state of $P(t, \Sigma)$ encodes the full connectivity information of the final solution restricted to X_t, that is, it encodes a connection set $\Delta \subseteq X_t \times X_t$, where $(x, y) \in \Delta$ implies that any solution that uses $P(t, \Sigma)$, must contain a path between x and y.

The following definition shows the rules that we can enforce on the state of subproblems (local) and its intended consequence (global).

Definition 3.22 (Local and Global Connectivity)**.**

Let $Y \subseteq E(G)$ and $Y_t = Y \cap E_t$.

We say that the pairs $\{(\Gamma_t, \Delta_t)\}_{t \in V(\mathcal{T})}$ satisfy the *local* (resp., *global*) *connectivity definition* for Y if, for every node $t \in V(\mathcal{T})$ (having left and right children t' and t'', respectively),

Local		**Global**
$\Gamma_t := \begin{cases} \emptyset & \text{if } t \text{ is a leaf of } \mathcal{T} \\ \operatorname{tc}(\Gamma_{t'} \cup \Gamma_{t''} \cup Y_t)\big\|_t & \text{otherwise} \end{cases}$		$\Gamma_t := \operatorname{tc}(Y \cap E(G_t))\big\|_t$
$\Delta_t := \begin{cases} \Gamma_t & \text{if } t = \operatorname{root}(\mathcal{T}) \\ \operatorname{tc}(\Delta_{p(t)} \cup \Gamma_t)\big\|_t & \text{otherwise} \end{cases}$		$\Delta_t := \operatorname{tc}(Y)\big\|_t$

where the projection operator on X_t is simplified as $|_t$.

The following lemma shows that the local and global connectivity definitions are, in fact, equivalent. We remark that the local definition only uses the state corresponding to a node t and its parent or children, whereas the global definition uses information about the entire solution. This lemma implies that, just by ensuring that the conditions in the local definition are satisfied, we can have access to global information in any subproblem.

Lemma 3.23. *Let $Y \subseteq E$ be a subset of edges and, for every $t \in V(\mathcal{T})$, (Γ_t, Δ_t) be a pair of connectivity sets.*

Then, the pairs (Γ_t, Δ_t) satisfy the local connectivity definition if and only if they satisfy the global connectivity definition.

We defer the proof of the lemma, as well as its extension to any combination of undirected or directed graphs, vertex or edge connectivity, and vertex or edge costs, to the end of Section 3.3, in order to focus on the application of this lemma to the design of dynamic programs.

3.3.4 A Dynamic Program for Steiner Tree

In this section, we present a dynamic program for Steiner tree in bounded-treewidth graphs, that we later use to solve GST. Such a dynamic program had already been presented by Chimani et al. [CMZ12], and algorithms with better running time are already known [BCK$^+$15, CNP$^+$11, FBN15]. We present our own construction for notational consistency and to simplify the presentation of the algorithm for GST. The size of the dynamic program, as well as the running time of the algorithm, are $O(nw^{O(w)})$, where w is the treewidth of G. In the following, we assume that $r \in X_t$ for every $t \in V(\mathcal{T})$.

We start by defining the subproblems of this dynamic program: our goal is to assign states (Γ, Δ) to each node $t \in \mathcal{T}$; to that end, we define a subproblem $P[t, \Gamma, \Delta]$ for every node $t \in \mathcal{T}$, and every possible pair of connectivity sets Γ, Δ for X_t. Intuitively, subproblem $P[t, \Gamma, \Delta]$ can be thought of as

> "Find the minimum-cost tree in G_t that (i) connects all the pairs in Γ; (ii) assuming the existence of paths between any pair in Δ, connects all the terminals in G_t to r ".

We store the value of the optimal solution in a table c, that is, for each subproblem $P[t, \Gamma, \Delta]$, we compute the optimal cost $c[t, \Gamma, \Delta]$.

We start by setting the initial values for leaf cells, as well as marking some cells as invalid, by setting $c[t, \Gamma, \Delta] = +\infty$. These cells represent situations in which the local definition does not apply, and therefore their use would prevent us from using Lemma 3.23. We run the following procedure to initialize the table.

Procedure 3.24: Initialization of the DP table.

(1) For every leaf node t and every pair (Γ, Δ), we set $c[t, \Gamma, \Delta] = 0$ if $\Gamma = \emptyset$ and set (t, Γ, Δ) as invalid otherwise.

(2) We mark the cells $(\text{root}(\mathcal{T}), \Gamma, \Delta)$ as invalid if $\Gamma \neq \Delta$.

(3) Let $v_i \in T$ be one of the terminals, and $t \in V(\mathcal{T})$ a node. We mark a cell (t, Γ, Δ) as invalid if $v_i \in X_t$ but $(r, v_i) \notin \Delta$.

For all of the other subproblems, we compute the optimal value by using the values of the children subproblems. Specifically, for a subproblem $P[t, \Gamma, \Delta]$, where t has children t_1, t_2, we are going to choose a subproblem for each of t_1, t_2, as well as a subset of edges of E_t to add to the solution.

Definition 3.25. Let $P[t_1, \Gamma_1, \Delta_1]$, $P[t_2, \Gamma_2, \Delta_2]$ be subproblems and $Y_t \subseteq E_t$ be a subset of edges of the bag. We say that (Γ, Δ) is *consistent* with $\big((\Gamma_1, \Delta_1), (\Gamma_2, \Delta_2)\big)$ via Y_t, denoted

$$(\Gamma, \Delta) \xleftrightarrow{Y_t} \big((\Gamma_1, \Delta_1), (\Gamma_2, \Delta_2)\big)$$

if the following conditions (closely related to the local connectivity definition of Section 3.3.3) hold:

$$\begin{aligned} \Gamma &= \operatorname{tc}(\Gamma_1 \cup \Gamma_2 \cup Y_t)|_t \\ \Delta_1 &= \operatorname{tc}(\Delta \cup \Gamma_1)|_{t_1} \\ \Delta_2 &= \operatorname{tc}(\Delta \cup \Gamma_2)|_{t_2} \end{aligned}$$

The optimum for $P := P[t, \Gamma, \Delta]$ is computed as the sum of the minimum cost for $P_1 := P[t_1, \Gamma_1, \Delta_1]$, $P_2 := P[t_2, \Gamma_2, \Delta_2]$ and Y_t where P is consistent with (P_1, P_2) via Y_t. Formally,

$$c[t, \Gamma, \Delta] = \min\Big\{ c[t_1, \Gamma_1, \Delta_1] + c[t_2, \Gamma_2, \Delta_2] + c(Y_t) \;\Big|\; (\Gamma, \Delta) \xleftrightarrow{Y_t} \big((\Gamma_1, \Delta_1), (\Gamma_2, \Delta_2)\big) \Big\}$$

The actual solution can be obtained recursively as $F_t = Y_t \cup F_{t_1} \cup F_{t_2}$.

The optimum c^* is the cost of best subproblem for $\operatorname{root}(\mathcal{T})$, that is,

$$c^* = \min_{\Gamma} c[\operatorname{root}(\mathcal{T}), \Gamma, \Gamma]$$

We now use the perspective of Section 3.3.1 to argue about the solution of this dynamic program. Consider the states for each subinstance in the optimum obtained by the dynamic program. Since each subproblem must choose a state for each child subinstance, we use exactly one subproblem $P[t, \Gamma_t, \Delta_t]$ for each node $t \in \mathcal{T}$. Furthermore, it can be seen that the cost of the solution is equal to the sum of $c(Y_t)$, for the Y_t used by the chosen subproblems.

We can now apply Lemma 3.23, with all of the chosen $Y_t \subseteq E_t$ plus Γ_t, Δ_t obtained from the chosen states for each subinstance. By the construction of the dynamic program, all of the conditions in the local connectivity definition are satisfied. Therefore, the global connectivity definition holds too, and we know that $\Delta_t = \operatorname{tc}(Y)|_t$. In particular, we know that for every terminal $v_i \in T$, there is a bag $X_t \ni v_i$, $t \in \mathcal{T}$. Since the solution cannot use invalid cells (otherwise the cost would be infinite), $(r, v_i) \in \Delta_t$, which implies that $(r, v_i) \in \operatorname{tc}(Y)|_t$, that is, there is a path in Y between r and v_i. We conclude that in the solution Y, every terminal is connected to the root r.

All that is left is to show that the optimum solution Y^* is found. Let $Y_t^* = Y^* \cap E_t$, $\Gamma_t = \operatorname{tc}(Y^* \cap E(G_t))|_t$, $\Delta_t = \operatorname{tc}(Y^*)|_t$. By Lemma 3.23, Γ_t, Δ_t also satisfy the local connectivity definition. Therefore, all of the subproblems taken are valid and (Γ_t, Δ_t) is consistent with $\big((\Gamma_{t_1}, \Delta_{t_1}), (\Gamma_{t_2}, \Delta_{t_2})\big)$ via Y_t^*, for all $t \in \mathcal{T}$ with children t_1, t_2. Furthermore, we can prove that $c[t, \Gamma_t, \Delta_t] \leq c(F \cap G_t)$, for any solution that is consistent with state (Γ_t, Δ_t) for G_t. Indeed, the claim is trivial for leaves and, for internal nodes,

$$\begin{aligned} c(F \cap G_t) &= c(F \cap E_t) + c(F \cap G_{t_1}) + c(F \cap G_{t_2}) \\ &\geq c(F \cap E_t) + c[t_1, \Gamma_{t_1}, \Delta_{t_1}] + c[t_2, \Gamma_{t_2}, \Delta_{t_2}] \\ &\geq c[t, \Gamma_t, \Delta_t], \end{aligned}$$

where the first inequality follows by induction. This implies that, for the right choice of Γ, $c[\operatorname{root}(\mathcal{T}), \Gamma, \Gamma] \leq c(Y^*)$, that is, an optimum solution is found.

3.3.5 Proof of Lemma 3.23

In this section, we will prove a generalized version of Lemma 3.23, that can be used for edge and vertex costs. Additionally, it can also be used for edge or vertex connectivity, meaning that, in higher-connectivity problems, it can be used to find edge-disjoint or vertex-disjoint paths. In order to handle the case of vertex connectivity, we introduce a modified version of transitive closure that is compatible with the concept of internally disjoint paths.

Definition 3.26. Let V be a set of vertices, $Z \subseteq V$ a subset of vertices, and $Y \subseteq V \times V$ a set of edges.

We denote by $\mathrm{tc}^*_Z(Y)$ the set of all pairs $(u, v) \in V \times V$ such that there is a u-v-path in the graph $(Z \cup \{u, v\}, Y)$, that is, a path whose internal vertices are in Z, and whose edges are in Y. Formally,

$$\mathrm{tc}^*_Z(Y) = \left\{(u, v) \;\middle|\; \exists p = (u, w_1, \ldots, w_\ell, v),\ E(p) \subseteq Y;\ \forall i \in [\ell], w_i \in Z\right\}$$

We remark that it is not necessary for u, v to be vertices of Z in order for (u, v) to be in $\mathrm{tc}^*_Z(Y)$. In fact, this is a useful feature of tc^* when working with vertex-disjoint paths from u to v, as the endpoints are shared among all paths. However, it is important that all of the internal vertices of a path be contained in Z.

Using this definition, we keep track, for every node $t \in \mathcal{T}$, of which vertices in the corresponding bag X_t are allowed to be used in the solution, by using triples (Z, Γ^*, Δ^*), instead of the previously used pairs (Γ, Δ) as the state for t.

For the purposes of this section, we introduce the following definitions for local and global connectivity.

Definition 3.27 (Generalized Local and Global Connectivity)**.**

Let $Y \subseteq E(G)$, $W \subseteq V(G)$, $Y_t = Y \cap E_t$, $W_t = W \cap \left(X_t \setminus X_{p(t)}\right)$.

We say that the the triples $\{(Z_t, \Gamma^*_t, \Delta^*_t)\}_{t \in \mathcal{T}}$ satisfy the *local* (resp., *global*) *connectivity definition* for Y if, for every node $t \in V(\mathcal{T})$ (having left and right children as t' and t'', respectively),

Local

$$Z_t := \begin{cases} W_t & \text{if } t = \mathrm{root}(\mathcal{T}) \\ \left(Z_{p(t)} \cup W_t\right) \cap X_t & \text{otherwise} \end{cases}$$

$$\Gamma^*_t := \begin{cases} \emptyset & \text{if } t \text{ is a leaf of } \mathcal{T} \\ \mathrm{tc}^*_{Z_t}\left(\Gamma^*_{t'} \cup \Gamma^*_{t''} \cup Y_t\right)\big|_t & \text{otherwise} \end{cases}$$

$$\Delta^*_t := \begin{cases} \Gamma^*_t & \text{if } t = \mathrm{root}(\mathcal{T}) \\ \mathrm{tc}^*_{Z_t}\left(\Delta^*_{p(t)} \cup \Gamma^*_t\right)\Big|_t & \text{otherwise} \end{cases}$$

Global

$$Z_t := W \cap X_t$$

$$\Gamma^*_t := \mathrm{tc}^*_W\left(Y \cap E(G_t)\right)\big|_t$$

$$\Delta^*_t := \mathrm{tc}^*_W\left(Y\right)\big|_t$$

We then prove the following lemma, proving that the given local and global connectivity definitions are equivalent.

Lemma 3.28. *Let $G = (V, E)$ be a (directed or undirected) graph, and $(\mathcal{T}, X)$ its (undirected) tree decomposition satisfying Properties (3) and (4) from Lemma 2.6. Let*

$W \subseteq V$ be a subset of vertices, $Y \subseteq E$ be a subset of edges and, for every $t \in V(\mathcal{T})$, $(Z_t, \Gamma_t^*, \Delta_t^*)$ be a triple as in Definition 3.27.

Then, the triples $(Z_t, \Gamma_t^*, \Delta_t^*)$ satisfy the local connectivity definition if and only if they satisfy the global connectivity definition.

Before proving the lemma, we show how Lemma 3.23 follows. We will prove that, if we fix $W = V$, $W_t = X_t \setminus X_{p(t)}$ and $Z_t = X_t$, the definitions of Lemmas 3.23 and 3.28 are equivalent (see Table 3.1).

For this, it is sufficient to see that $\mathrm{tc}^*_W(F) = \mathrm{tc}^*_V(F) = \mathrm{tc}(F)$ for any set $F \subseteq V \times V$, and that the common vertices in $\Gamma^*_{t'}$, $\Gamma^*_{t''}$, and Y_t are all in X_t, thus

$$\mathrm{tc}^*_{Z_t}\left(\Gamma^*_{t'} \cup \Gamma^*_{t''} \cup Y_t\right)\big|_t = \mathrm{tc}\left(\Gamma^*_{t'} \cup \Gamma^*_{t''} \cup Y_t\right)\big|_t$$

Similarly, since $\Delta^*_{p(t)}$ and Γ^*_t only intersect in X_t,

$$\mathrm{tc}^*_{Z_t}\left(\Delta^*_{p(t)} \cup \Gamma^*_t\right)\big|_t = \mathrm{tc}\left(\Delta^*_{p(t)} \cup \Gamma^*_t\right)\big|_t$$

We conclude that if $Z_t = X_t$, then $\Gamma^*_t = \Gamma_t$ and $\Delta^*_t = \Delta_t$, and the proof follows.

For reference, we show how to use this lemma for the cases of edge connectivity, edge costs, and vertex connectivity, vertex costs in Table 3.1. Notice that in a problem that does not require disjoint paths, such as Steiner tree, edge connectivity and vertex connectivity are equivalent, and therefore one of the simplified cases of Table 3.1 is always sufficient. In more general problems, the more general version of the lemma may be required in combination with more general techniques. Examples of the use of this lemma in fault-tolerant settings can be seen in Chapter 4.

Connectivity	**Costs**	
Edge	Edge	$W = V$, $W_t = X_t \setminus X_{p(t)}$, $Z_t = X_t$
Vertex	Vertex	$Y = E$, $Y_t = E_t$

Table 3.1: Table demonstrating how to adapt Lemma 3.28 to different settings

The following technical lemma is useful when arguing about connectivity in tree decompositions, and will be needed to prove Lemma 3.28. It proves that if a path starts and ends in a bag X_t, but none of its internal vertices are contained in X_t, then the edges of the path must be contained in one subtrees of $\mathcal{T} \setminus t$ (that is, the path cannot "cross" over the bag X_t).

Lemma 3.29 (Path Lemma)**.** *Let G be any graph and $(\mathcal{T}, \mathcal{X})$ be a tree decomposition of G. Let $t \in V(\mathcal{T})$ and P be a simple path with at least 2 edges, whose endpoints x, y are the only vertices of P in X_t, that is, $V(P) \cap X_t \subseteq \{x, y\}$.*

Then there is a connected component $\mathcal{T}'$ in $\mathcal{T} \setminus t$ whose bags contain all the edges of P, that is, for every edge $(a, b) \in E(P)$, there is a node $t' \in V(\mathcal{T}')$ such that $(a, b) \in E_{t'}$.

Proof. We provide a simple proof by contradiction. Assume that there are two consecutive edges, $(a, b), (b, c) \in E(P)$ that are in different connected components of $\mathcal{T} \setminus t$ (otherwise, all edges must be in the same component). Since b is contained in two bags $X_{t'}$, $X_{t''}$ on different components of $\mathcal{T} \setminus t$, all bags corresponding to nodes on the path between t' and

t'' must also contain b, by definition of tree decomposition. Since t', t'' are in different components of $\mathcal{T} \setminus t$, then t is on the path between t' and t'', which implies that $b \in X_t$. Thus we reach a contradiction with the assumption of the lemma. □

Proof of Lemma 3.28. We remark that the function tc^* shares some properties with the usual definition of transitive closure, which are used throughout the proof:

Observation 3.30. *The function* tc^* *satisfies the following properties:*

- *Idempotence:* $\mathrm{tc}^*_Z(\mathrm{tc}^*_Z(Y)) = \mathrm{tc}^*_Z(Y)$
- *Monotonicity:* $\mathrm{tc}^*_{Z'}(Y') \subseteq \mathrm{tc}^*_Z(Y)$ *if* $Z' \subseteq Z$, $Y' \subseteq Y$

Equivalence for Z_t We prove that the two definitions for Z are equivalent by induction on the depth of the node (top-down). At the root, we have that $W_{\mathrm{root}(\mathcal{T})} = W \cap X_{\mathrm{root}(\mathcal{T})}$, so the equivalence holds.

For the induction step, let $t \in V(\mathcal{T})$ be a node other than the root. Then

$$\begin{aligned}\big(Z_{p(t)} \cup W_t\big) \cap X_t &= \big(Z_{p(t)} \cap X_t\big) \cup (W_t \cap X_t) \\ &\subseteq (W \cap X_t) \cup (W \cap X_t) \\ &= W \cap X_t\end{aligned}$$

The second step follows from the induction hypothesis, as well as the definition of W. We now prove the converse inclusion.

$$\begin{aligned}W \cap X_t &= \big(W \cap X_{p(t)} \cap X_t\big) \cup \big(W \cap \big(X_t \setminus X_{p(t)}\big)\big) \\ &\subseteq \big(Z_{p(t)} \cap X_t\big) \cup W_t \\ &= \big(Z_{p(t)} \cup W_t\big) \cap X_t\end{aligned}$$

We use the induction hypothesis, as well as the fact that $W_t \subseteq X_t$.

Equivalence for Γ^*_t We prove the statement by induction on the height of a node t (bottom-up). Since $E(G_t) = \emptyset$, both definitions are equivalent for every leaf t.

Let t be any node. By induction hypothesis, $\Gamma^*_{t'}, \Gamma^*_{t''} \subseteq \mathrm{tc}^*_W\left(Y \cap E(G_t)\right)$. Therefore,

$$\mathrm{tc}^*_{Z_t}\left(\Gamma^*_{t'} \cup \Gamma^*_{t''} \cup Y_t\right)\big|_t \subseteq \mathrm{tc}^*_W\left(Y \cap E(G_t)\right)\big|_t$$

Here, we use that $Z_t = W \cap X_t \subseteq W$.

To prove the converse inclusion, let $(u, v) \in \mathrm{tc}^*_W\left(Y \cap E(G_t)\right)\big|_t$. By definition, there must be a path p between u and v using internal vertices in W. Let $u = w_0, w_1, \ldots, w_\ell = v$ be all the vertices of p (in the correct order) that are also in X_t.

Each pair (w_i, w_{i+1}) is connected by a subpath of p. By Lemma 3.29, (w_i, w_{i+1}) is either an edge in Y_t, or the subpath is fully contained in $Y \cap E(G_{t'})$ or $Y \cap E(G_{t''})$, and uses internal vertices in W. In the first case, $(w_i, w_{i+1}) \in Y_t$, while in the remaining cases, (w_i, w_{i+1}) is in $\Gamma_{t'}$ or $\Gamma_{t''}$, by the induction hypothesis. We conclude that, since $w_i \in Z_t = W \cap X_t$, for $i \in [l-1]$, then (u, v) is in $\mathrm{tc}^*_{Z_t}\left(\Gamma^*_{t'} \cup \Gamma^*_{t''} \cup Y_t\right)\big|_t$.

Equivalence for Δ_t^* We now prove that both definitions for Δ^* are equivalent, by using induction on the depth of the nodes (top-down). For the node $\text{root}(\mathcal{T})$, we have $E(G_{\text{root}(\mathcal{T})}) = E(G)$, and thus the base case follows.

For the induction step, we remark that $\Delta^*_{p(t)}, \Gamma^*_t \subseteq \text{tc}^*_W(Y)$ by the induction hypothesis together with the statement of the lemma for Γ^*. Therefore,

$$\text{tc}^*_{Z_t}\left(\Delta^*_{p(t)} \cup \Gamma^*_t\right)\big|_t \subseteq \text{tc}^*_W(Y)|_t$$

For the reverse inclusion, we fix a pair $(u, v) \in \text{tc}^*_W(Y)|_t$, and a path p that connects u to v in Y using internal vertices in W. Further, let $u = w_0, w_1, \ldots, w_\ell = v$ be all the vertices of p (in the correct order) that are also in X_t.

By Lemma 3.29, (w_i, w_{i+1}) is either an edge in Y_t, or the subpath p' of p connecting w_i and w_{i+1} is contained either in $Y \cap E(G_{t'})$, $Y \cap E(G_{t''})$ or $Y \cap (E \setminus E(G_t))$. For all but the last case, $(w_i, w_{i+1}) \in \Gamma^*_t$, by definition. In the remaining case, it must be that the vertices of p' are also contained in $X(\mathcal{T} \setminus \mathcal{T}_b)$. Specifically, because the nodes whose bags contain a given vertex must form a connected component of $\mathcal{T}$, $w_i, w_{i+1} \in X_{p(t)}$, which implies $(w_i, w_{i+1}) \in \text{tc}^*_W(Y)|_{p(t)} = \Delta^*_{p(t)}$.

In any case, since $w_i \in Z_t = W \cap X_t$, for $i \in [l-1]$, we conclude that every pair (w_i, w_{i+1}) and thus (u, v), are contained in $\text{tc}^*_{Z_t}\left(\Delta^*_{p(t)} \cup \Gamma^*_t\right)\big|_t$. □

3.4 Solving GST on Bounded-Treewidth Graphs

As we have seen in Section 3.3.4, we can solve Steiner tree optimally on bounded-treewidth graphs, using dynamic programming and the fact that the problem has optimal substructure. For GST, the situation is more complicated: if we consider subproblems that correspond to covering a specific set of groups, the problem has optimal substructure and we can solve using dynamic programming. In that case, the best-known running time is $O(2^h \operatorname{poly}(n))$. If we want to avoid an exponential dependency on h, the problem does not seem to have optimal substructure, and thus dynamic programming does not seem applicable.

Let us briefly consider the dynamic programming for Steiner tree. It splits a problem into subproblems by considering two almost-disjoint subgraphs (their intersection is contained in X_t). This is possible for Steiner tree because all of the terminals must be covered, and therefore an optimal solution for the subproblems covers each of the terminals in the subgraph. If we instead have groups, now both of the subproblems may have elements of the same groups. Therefore, an optimal solution for the problem may cover each group on one of the two subgraphs. This means that there is no longer an "optimal solution" for the individual subproblems: there are dependencies between the subproblems that we do not know how to encode in the state using $o(2^h)$ possibilities. In fact, it seems unlikely that such a strategy would work, as we know that even for trees (with treewidth 1), we cannot compute the optimal solution in polynomial time [HK03].

We take a slightly different approach to solving this problem, one that is inspired by dynamic programming, but incorporates the knowledge of how to solve GST on trees. The idea is to take a dynamic program similar to the one for Steiner tree, but without specifying any terminals (hence no cells are marked invalid in Step (3) of Procedure 3.24). Even without terminals, every solution obtained from this dynamic program still satisfies

local (and therefore global) connectivity definitions, though it is not forced to connect any vertices.

The next step is to use a different algorithm to find a solution in the dynamic program, such that a fraction of the groups is covered, while keeping the cost low. We associate groups to the subproblems as follows: if the state of the subproblem implies that a group is covered, we say that the subproblem belongs to the group. Using the framework in Section 3.3, we simply need to check if (r, v_i) is in Δ, for some $v_i \in S_i$ which is part of the group. We are left with the problem of finding a solution that covers all groups, i.e. such that for each group there is one subproblem in the solution that covers it.

The key to solving this problem is that it is very similar to group Steiner tree itself (with some additional degree constraints). The input graph for the instances we obtain is a certain type of DAG with bounded maximum degree and logarithmic height. Therefore, we can transform it into a tree by creating copies of nodes with in-degree more than 1 (in a bottom-up fashion). Doing so creates a tree that has the same optimum for the group Steiner tree, but which is much larger than the DAG (the size of the tree is $n^{O(w \log w)}$). Solving the linear program and rounding it similarly to what is done in the GKR algorithm (Section 3.2) gives an $O(\log n \log h)$-approximation in polynomial time for bounded treewidth.

We now formalize these ideas and prove the correctness of the algorithm. In order to state our result formally, we introduce the concept of a solution tree, which represents a dynamic programming solution.

Definition 3.31 (Solution tree)**.**

Let $\widetilde{H}$ be a DAG with root $\tilde{r}$, and let its nodes be partitioned into *combination nodes* $\widetilde{H}_c$, and *subproblem nodes* $\widetilde{H}_p$.

We say an out-arborescence $T \subseteq \widetilde{H}$ is a *solution tree* if:

(1) It is rooted at $\tilde{r}$,

(2) Every node $\tilde{t} \in T$ has in-degree at most 1,

(3) Every combination node $\tilde{t}^c \in T \cap \widetilde{H}_c$ has full out-degree (i.e. all of its children are also in T),

(4) Every subproblem node $\tilde{t} \in T \cap \widetilde{H}_p$ (including the root $\tilde{r}$) has out-degree 1 in T (i.e. there is only one arc out of $\tilde{t}$).

Using this definition, we specify the following problem, which we reduce GST to.

Problem 3.32: Solution Tree Group Steiner Tree (STGST).

- Instance: $\big(H, c, r, \mathcal{S}, H_c, H_p\big)$, where
  - $H = (V, E)$ is a DAG with edge costs $c : E \to \mathbb{R}$;
  - $r \in V$ is the *root* of the instance;
  - $\mathcal{S} = \{S_i\}_{i \in [h]}$ is a collection of *groups* $S_i \subseteq V$, $i \in [h]$;
  - H_c, H_p partition $V(H)$, and are called the sets of *combination* and *subproblem* nodes, respectively.
- Solution: a solution tree $F \subseteq E$ that, for every $i \in [h]$, connects r to some $v_i \in S_i$
- Goal: minimize the cost of F, $c(F) = \sum_{e \in F} c_e$

3.4.1 Encoding GST on Bounded-Treewidth Graphs as a Tree

Theorem 3.33. *Let $\mathcal{I} = (G, c, r, \mathcal{S})$ be an instance of GST, where G has treewidth w.*

There is an STGST instance $\mathcal{I}' = \left(\widetilde{\mathcal{T}}, \tilde{c}, \tilde{r}, \{\widetilde{S}_i\}_{i\in[h]}, \widetilde{\mathcal{T}}_c, \widetilde{\mathcal{T}}_p\right)$, where $\widetilde{\mathcal{T}}$ is a tree, such that:

(1) $|V(\widetilde{\mathcal{T}})| = O(n^{O(w \log w)})$

(2) for every tree $F \subseteq E(G)$ there is a solution tree $\mathcal{Y}$ (and vice-versa), such that $c(F) = c(\mathcal{Y})$ and, for every $i \in [h]$, F connects r to S_i if and only if $\mathcal{Y}$ connects $\mathrm{root}(\widetilde{\mathcal{T}})$ *to $\widetilde{S}_i$.*

Furthermore, we can compute $\mathcal{I}'$ given $\mathcal{I}$, as well as F (resp. $\mathcal{Y}$) given $\mathcal{Y}$ (resp. F), in time $O\left(n^{O(w \log w)}\right)$.

The theorem also applies for GST with vertex costs; even then, the STGST instance in the reduction uses edge costs. The rest of this section is dedicated to proving this theorem. Throughout the section, we assume that $r \in X_t$ for every $t \in V(\mathcal{T})$ (we can simply add r to every bag).

Before we define $\widetilde{\mathcal{T}}$, we are going to define a DAG $\widetilde{H}$, and then show how to transform it into a tree. The structure of $\widetilde{H}$ is similar to the dynamic programming space in Section 3.3.4, and all the nodes are partitioned into combination nodes $\widetilde{H}_c$ and problem nodes $\widetilde{H}_p$. It consists of:

(1) Nodes $\tilde{t}[t, \Gamma, \Delta] \in \widetilde{H}_p$ for every $t \in \mathcal{T}$, $\Gamma, \Delta \in \mathcal{C}_t$;

(2) Combination nodes $\tilde{t}^c[\tilde{t}, \tilde{t}_1, \tilde{t}_2, Y_t] \in \widetilde{H}_c$ for every $\tilde{t} = \tilde{t}[t, \Gamma, \Delta]$, $\tilde{t}_i = \tilde{t}[t_i, \Gamma_i, \Delta_i]$, where t_1, t_2 are the children of t, and $Y_t \subseteq E_t$;

(3) Arcs $(\tilde{t}, \tilde{t}^c)$, $(\tilde{t}^c, \tilde{t}_1)$, $(\tilde{t}^c, \tilde{t}_2)$, for $\tilde{t}^c[\tilde{t}, \tilde{t}_1, \tilde{t}_2, Y_t] \in \widetilde{H}$ and

$$(\Gamma, \Delta) \xleftrightarrow{Y_t} ((\Gamma_1, \Delta_1), (\Gamma_2, \Delta_2));$$

(4) Root node $\tilde{r} \in \widetilde{H}_p$ and arcs $(\tilde{r}, \tilde{t})$, for all $\tilde{t} = \tilde{t}[\mathrm{root}(\mathcal{T}), \Gamma, \Gamma]$, with $\Gamma \in \mathcal{C}_{\mathrm{root}(\mathcal{T})}$.

All arcs except those of type $(\tilde{t}, \tilde{t}^c)$ have cost 0. The cost of an arc $(\tilde{t}, \tilde{t}^c)$ is $\tilde{c}(\tilde{t}, \tilde{t}^c) = c(Y_t)$.

This construction contains all the nodes we need, but may contain some nodes corresponding to invalid subproblems, or nodes not connected to the root. Therefore, we apply a pruning procedure:

Procedure 3.34: Pruning of $\widetilde{H}$.

(1) Remove all nodes $\tilde{t}, \tilde{t}^c \in \widetilde{H}$ that are not connected from the root, that is, nodes for which there is no directed path from the root in $\widetilde{H}$.

(2) Remove all nodes $\tilde{t} = \tilde{t}[t, \Gamma, \Delta]$, where t is a leaf and $\Gamma \neq \emptyset$.
Additionally, remove any combination node $\tilde{t}^c \in \widetilde{H}$, if there is an arc $(\tilde{t}^c, \tilde{t})$ and $\tilde{t}$ is removed.

(3) Repeatedly remove nodes $\tilde{t} = \tilde{t}[t, \Gamma, \Delta]$, where t is *not* a leaf, but $\tilde{t}$ has no children.
Additionally, remove any combination node $\tilde{t}^c \in \widetilde{H}$, if there is an arc $(\tilde{t}^c, \tilde{t})$ and $\tilde{t}$ is removed.

The DAG $\widetilde{H}$ already satisfies all of the desired properties in the statement of the theorem, except that it is not a tree. We will now prove these properties, before showing how to transform $\widetilde{H}$ into a tree.

We start by the size of $\widetilde{H}$: the number of nodes in $\widetilde{H}$ is $|V(\widetilde{H})| = O(nw^{O(w)})$, as there are at most $\sum_{t\in\mathcal{T}} |\mathcal{C}_t|^2 \leq O(nw^{O(w)})$ nodes $\tilde{t}$. The combination nodes $\tilde{t}^c$ are parameterized by a node $t \in \mathcal{T}$ (with children t_1, t_2), $\Gamma, \Delta \in \mathcal{C}_t$, $\Gamma_1, \Delta_1 \in \mathcal{C}_{t_1}$, $\Gamma_2, \Delta_2 \in \mathcal{C}_{t_2}$, and $Y_t \subseteq E_t$. Since Property (2) of Theorem 3.33 specifies that F is a tree, we can restrict Y_t to be a forest (a subset of a tree). Therefore, the number of possibilities for Y_t is at most $O(w^{O(w)})$ (by Proposition 2.1). Additionally, we know that $|V(\mathcal{T})| = O(n)$, and $|\mathcal{C}_t| \leq O(w^{O(w)})$, so we conclude that the number of combination nodes is $O(nw^{O(w)})$. More importantly, the out-degree of every combination node is 2; the out-degree of the root is at most $O(w^{O(w)})$; as each edge is parameterized by $\Gamma \in \mathcal{C}_{\mathrm{root}(\mathcal{T})}$; and the out-degree of any other node is $O(w^{O(w)})$, since the number of combination nodes for that same node $t \in \mathcal{T}$ is bounded by $O(w^{O(w)})$.

Consider now the groups $\widetilde{S}'_i$ for $\widetilde{H}$ to be all nodes $\tilde{t} = \tilde{t}[t, \Gamma, \Delta]$ such that $(r, v) \in \Delta$ for some $v \in S_i$. Formally,

$$\widetilde{S}'_i = \left\{ \tilde{t} \in \widetilde{H} : \tilde{t} = \tilde{t}[t, \Gamma, \Delta], (r, v) \in \Delta, v \in S_i \right\} \qquad \forall i \in [h]$$

Both $\widetilde{H}$ and the groups $\widetilde{S}'_i$ can be computed in time $O\big(nw^{O(w)}\big)$, since it is sufficient to enumerate all the problems and combination nodes in time $O\big(nw^{O(w)}\big)$, then add the edges to each combination node, and determine the groups, which both take time linear in the size of $\widetilde{H}$.

The following lemma proves that $\widetilde{H}$ satisfies Property (2) of Theorem 3.33.

Lemma 3.35. *For every tree $F \subseteq E(G)$ there is a solution tree $\mathcal{Y}$ (and vice-versa), such that $c(F) = c(\mathcal{Y})$ and for every $i \in [h]$, F connects r to S_i if and only if $\mathcal{Y}$ connects* $\mathrm{root}(\widetilde{\mathcal{T}})$ *to $\widetilde{S}'_i$.*

Proof. Let $F \subseteq E(G)$. We define $\Gamma_t := \mathrm{tc}\left(F \cap E(G_t)\right)|_t$, $\Delta_t := \mathrm{tc}\left(F\right)|_t$, for all $t \in \mathcal{T}$. We now take $\mathcal{Y}$ consisting of $\tilde{r}$, $\tilde{t}_t = \tilde{t}[t, \Gamma_t, \Delta_t]$ for all $t \in \mathcal{T}$, and $\tilde{t}^c[\tilde{t}_t, \tilde{t}_{t_1}, \tilde{t}_{t_2}, F \cap E_t]$ for all $t \in \mathcal{T}$ with children t_1, t_2. By Lemma 3.23, all the conditions are satisfied for the edges to exist between the nodes and for all nodes to be valid. Additionally, all the conditions in Definition 3.31 hold. Therefore, $\mathcal{Y}$ is a (feasible) solution tree.

The cost of $\mathcal{Y}$ is

$$c(\mathcal{Y}) = \sum_{t\in\mathcal{T}} c(\tilde{t}^c[\tilde{t}_t, \tilde{t}_{t_1}, \tilde{t}_{t_2}, F \cap E_t]) = \sum_{t\in\mathcal{T}} \mathrm{c}(F \cap E_t) = \mathrm{c}(F)$$

We will now show the converse implication. We claim that a solution tree $\mathcal{Y}$ assigns a unique state to each bag.

Observation 3.36. *Given a solution tree $\mathcal{Y} \subset \widetilde{H}$, $\mathcal{Y}$ must contain exactly one node $\tilde{t}_t = \tilde{t}[t, \Gamma_t, \Delta_t] \in \mathcal{Y}$, and one combination node $\tilde{t}^c_t = \tilde{t}^c[\tilde{t}_t, \tilde{t}_{t_1}, \tilde{t}_{t_2}, Y_t]$ for each $t \in \mathcal{T}$.*

Proof. Clearly the statement holds for $t = \mathrm{root}(\mathcal{T})$, since $\tilde{r} \in \widetilde{H}_p$, which implies that exactly one node $\tilde{t}_t = \tilde{t}[t, \Gamma_t, \Delta_t]$ is in $\mathcal{Y}$. Similarly, because $\tilde{t}_t \in \widetilde{H}_p$, there is only one $\tilde{t}^c[\tilde{t}_t, \tilde{t}_{t_1}, \tilde{t}_{t_2}, F \cap E_t]$ in $\mathcal{Y}$.

The statement follows by induction on the depth of the node, since if there is only one $\tilde{t}^c[\tilde{t}_t, \tilde{t}_{t_1}, \tilde{t}_{t_2}, F \cap E_t]$ in $\mathcal{Y}$ for a given $t \in \mathcal{T}$, then that implies that there is only one $\tilde{t}[t_1, \Gamma_{t_1}, \Delta_{t_1}]$ and one $\tilde{t}[t_2, \Gamma_{t_2}, \Delta_{t_2}]$ in $\mathcal{Y}$ as well. □

Using the observation above, we can then define F to be

$$\begin{aligned} F &= \bigcup_{t \in \mathcal{T}} Y_t \\ &= \bigcup \{Y : \tilde{t}^c[\tilde{t}, \tilde{t}_1, \tilde{t}_2, Y] \in \mathcal{Y}\} \end{aligned}$$

Similarly to the above, we can show that $\mathrm{c}(F) = c(\mathcal{Y})$. If F connects r to a vertex $v_i \in S_i$, then there must be a bag $X_t \ni v_i$ and $\tilde{t}[t, \Gamma_t, \Delta_t] \in \mathcal{Y}$. Using Lemma 3.23, $(r, v_i) \in \mathrm{tc}\,(F)|_t = \Delta_t$, which implies that $\tilde{t} \in \widetilde{S}'_i$, where $\tilde{t}$ is in the solution $\mathcal{Y}$. Conversely, if some $\tilde{t} = \tilde{t}[t, \Gamma_t, \Delta_t] \in \widetilde{S}'_i$ is in the solution $\mathcal{Y}$ (and hence connected to $\tilde{r}$), there must be some $v_i \in S_i$ such that $(r, v_i) \in \Delta_t$ (by definition of S'_i). By Lemma 3.23, $\Delta_t = \mathrm{tc}\,(F)|_t$, so if $(r, v_i) \in \Delta_t$, r is connected to v_i in F. □

All that is left is to show how to transform $\widetilde{H}$ into a tree. Our goal is achieved by making copies of nodes in $\widetilde{H}$ so that they have in-degree 1. The following procedure shows how to construct $\widetilde{\mathcal{T}}$.

Procedure 3.37.

(1) For every node $\tilde{t} \in \widetilde{H}$ (including root and combination nodes), add a node $\tilde{t}^{(p)}$ to $\widetilde{\mathcal{T}}$ for every path p in $\widetilde{H}$ from $\tilde{r}$ to $\tilde{t}$;

(2) Connect two nodes $\tilde{t}^{(p)}$, $\tilde{t}'^{(p')}$ if $p' = p \circ t$, i.e. p' is p with an edge to t at the end;

(3) The root of $\widetilde{\mathcal{T}}$ is the only copy of $\tilde{r}$, i.e. $\mathrm{root}(\widetilde{\mathcal{T}}) = \tilde{r}^{((\tilde{r}))}$;

(4) $\widetilde{S}_i = \{\tilde{t}^{(p)} : \tilde{t} \in \widetilde{S}'_i\}$.

The two representations are virtually equivalent; we can easily convert a solution tree $\mathcal{Y}$ in $\widetilde{H}$ to a solution tree $\mathcal{Y}'$ in $\widetilde{\mathcal{T}}$ by adding $\tilde{r}$ to $\mathcal{Y}'$, and then repeating the following: for every node $\tilde{t}' \in \mathcal{Y}'$, which is a copy of $\tilde{t} \in \mathcal{Y}$, add to $\mathcal{Y}$ the children of $\tilde{t}'$ in $\widetilde{\mathcal{T}}$ which are copies of the children of $\tilde{t}$. To convert a solution tree $\mathcal{Y}'$ in $\widetilde{\mathcal{T}}$ to $\mathcal{Y}$ in $\widetilde{H}$, we just need to take the original nodes that the nodes of $\mathcal{Y}'$ are copies of.

Using the correspondence between solution trees in $\widetilde{H}$ and in $\widetilde{\mathcal{T}}$, we can see that Lemma 3.35 implies that Property (2) in Theorem 3.33 holds for $\widetilde{\mathcal{T}}$. To prove that $|V(\widetilde{\mathcal{T}})| = O(n^{O(w \log w)})$, it is sufficient to consider that the degree in $\widetilde{\mathcal{T}}$ is bounded by $O(w^{O(w)})$ (because it is bounded also in $\widetilde{H}$), and that the height of $\mathcal{T}$, and therefore of $\widetilde{H}$ and $\widetilde{\mathcal{T}}$, is $O(\log n)$. We conclude that the number of nodes in $\mathcal{T}$ is

$$|V(\widetilde{\mathcal{T}})| = O\Big(w^{O(w)}\Big)^{O(\log n)} = n^{O(w \log w)}$$

To compute $\widetilde{\mathcal{T}}$, we simply enumerate all of its nodes (that is, all paths in $\widetilde{H}$). Therefore, we can compute $\widetilde{\mathcal{T}}$ from $\widetilde{H}$ in time linear in $\big|\widetilde{\mathcal{T}}\big|$. This concludes the proof of the theorem.

Vertex Costs

In order to adapt Theorem 3.33 to GST with vertex costs, we need to modify the states throughout the proof to match those of Lemma 3.28. We will briefly describe the changes necessary to the algorithm.

The main modification is to the nodes of the tree: instead of nodes $\tilde{t}[t, \Gamma, \Delta] \in \widetilde{H}_p$, we now have nodes $\tilde{t}[t, \Gamma, \Delta, Z] \in \widetilde{H}_p$ for every $t \in \widetilde{\mathcal{T}}$, $\Gamma, \Delta \in \mathcal{C}_t$, $Z \subseteq X_t$. Given nodes $\tilde{t}[t, \Gamma, \Delta, Z]$, $\tilde{t}[t_1, \Gamma_1, \Delta_1, Z_1]$, $\tilde{t}[t_2, \Gamma_2, \Delta_2, Z_2]$, where t_1, t_2 are the children of t in $\mathcal{T}$, we write:

$$(\Gamma, \Delta, Z) \xleftrightarrow{Y_t} \big((\Gamma_1, \Delta_1, Z_1), (\Gamma_2, \Delta_2, Z_2)\big)$$

if the following conditions hold:

$$\begin{aligned} \Gamma &= \mathrm{tc}^*_Z(\Gamma_1 \cup \Gamma_2 \cup Y_t)\big|_t \\ \Delta_1 &= \mathrm{tc}^*_{Z_1}(\Delta \cup \Gamma_1)\big|_{t_1} & \Delta_2 &= \mathrm{tc}^*_{Z_2}(\Delta \cup \Gamma_2)\big|_{t_2} \\ Z_1 \cap X_t &= Z \cap X_{t_1} & Z_2 \cap X_t &= Z \cap X_{t_2} \end{aligned}$$

We remark that, even though we do not check the exact same conditions for Z as in Lemma 3.28, our consistency constraints follow by setting $W_t = Z_t \setminus X_{p(t)}$.

We use this new condition when determining what arcs to add, that is, we add the arcs to and from a combination node if $(\Gamma, \Delta, Z) \xleftrightarrow{E_t} \big((\Gamma_1, \Delta_1, Z_1), (\Gamma_2, \Delta_2, Z_2)\big)$. Notice the use of E_t instead of Y_t, implying that the solution can use any edge. For the root node, we add arcs $(\tilde{r}, \tilde{t})$ for all $\tilde{t} = \tilde{t}[\mathrm{root}(\mathcal{T}), \Gamma, \Gamma, Z]$, with $\Gamma \in \mathcal{C}_{\mathrm{root}(\mathcal{T})}$, $Z \subseteq X_{\mathrm{root}(\mathcal{T})}$.

The cost of an arc $(\tilde{r}, \tilde{t})$ is $c(Z)$, and the cost of an arc $(\tilde{t}^c, \tilde{t}_1)$ (resp. $(\tilde{t}^c, \tilde{t}_2)$) is $\tilde{c}(\tilde{t}^c, \tilde{t}_1) = c(Z_1 \setminus Z)$ (resp. $\tilde{c}(\tilde{t}^c, \tilde{t}_2) = c(Z_2 \setminus Z)$). All other arcs have cost 0.

Finally, we define the groups as

$$\widetilde{S}'_i = \left\{\tilde{t} \in \widetilde{H} : \tilde{t} = \tilde{t}[t, \Gamma, \Delta, Z], (r, v) \in \Delta, v \in S_i \cap Z\right\} \qquad \forall i \in [h]$$

The remainder of the analysis is similar to the case of edge costs. For a given solution $W \subseteq V$, we define

$$\begin{aligned} Z_t &= W \cap X_t \\ \Gamma_t &= \mathrm{tc}^*_{Z_t}(E(G_t)\big|_t \\ \Delta_t &= \mathrm{tc}^*_{Z_t}(E)\big|_t \end{aligned}$$

Given states $\{(Z_t, \Gamma_t, \Delta_t)\}_{t \in \mathcal{T}}$, we can define a solution $W = \bigcup_{t \in \mathcal{T}} Z_t$. The proof of Lemma 3.35 follows from these definitions.

3.4.2 Approximating GST on Bounded-Treewidth Graphs

Now that we have showed how to obtain a tree instance that captures finding a solution for GST in bounded-treewidth graphs, we need to show how to efficiently obtain an approximate solution in this new instance. Our task is to find a solution tree in the graph that covers all groups, and has relatively good cost. We will show that simple changes to the LP and to the rounding procedure presented in Section 3.2 will be sufficient to obtain the desired results.

Specifically, we will add LP constraints for every combination node, with the meaning that, if its parent edge is picked, then every outgoing edge has to be picked as well; in LP terms, the fractional value of each outgoing edge is the same as the value of the parent edge. Due to the rounding algorithm (similar to GKR, presented in Section 3.2), this implies immediately that if a combination node is in the solution, all of its outgoing edges are too. For the remaining nodes, we will add constraints specifying that only one outgoing edge must be picked. We do this by stating that the sum of LP values of outgoing edges must equal the LP value of the parent edge. In order to ensure that these nodes have out-degree 1 in the solution, the rounding algorithm is changed so that it picks exactly one outgoing edge, with probability proportional to the LP values.

These are all the changes required to the algorithm, and as we will see, the analysis of the GKR algorithm (see Section 3.2) mostly applies in this case, with only one small change being necessary to account for the modified algorithm.

The following theorem formalizes these changes and shows that we can get a good approximate solution for GST on bounded-treewidth graphs.

Theorem 3.38. *Let* $\left(\widetilde{\mathcal{T}}, \tilde{c}, \tilde{r}, \{\widetilde{S}_i\}_{i\in[h]}, \widetilde{\mathcal{T}}_c, \widetilde{\mathcal{T}}_p\right)$ *be an instance of STGST, where* $\widetilde{\mathcal{T}}$ *is a tree.*

There is an algorithm that runs in time $\mathrm{poly}(|\widetilde{\mathcal{T}}|)$ *and outputs a solution tree* $\mathcal{Y} \subseteq \widetilde{\mathcal{T}}$ *sampled from a distribution* $\mathcal{D}$ *such that:*

(1) $\mathrm{E}_{\mathcal{Y}\sim\mathcal{D}}[c(\mathcal{Y})] \le c(\mathrm{OPT})$*, where* $c(\mathrm{OPT})$ *denotes the cost of the optimal solution*

(2) For any group $\widetilde{S}_i$*, the probability that the group is covered (with* α *constant) is*

$$\Pr_{\mathcal{Y}\in\mathcal{D}}\left[|\widetilde{S}_i \cap \mathcal{Y}| > 0\right] \ge \frac{1}{\alpha \,\mathrm{height}(\widetilde{\mathcal{T}})}$$

We will first show how to use this theorem to approximate GST on bounded-treewidth graphs. Apply Theorem 3.38 ℓ times, with $\ell = 3\alpha\,\mathrm{height}(\widetilde{\mathcal{T}})\log h$, to obtain solutions $\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_\ell$. Then, apply Property (2) of Theorem 3.33 to each $\mathcal{Y}_i$ to obtain a tree F_i in the original graph G. Finally, take the union of all F_i to get the solution $\hat{F} = \bigcup_i F_i$, removing extra edges if necessary to make it a tree.

Let us analyze the cost of $\hat{F}$: we know that $c(F_i) = c(\mathcal{Y}_i)$ by Theorem 3.33 and that $\mathrm{E}_{\mathcal{Y}\sim\mathcal{D}}[c(\mathcal{Y})] \le c(\mathrm{OPT})$. Therefore,

$$\mathrm{E}[c(\hat{F})] \le \ell\, c(\mathrm{OPT}) = O(\log n \log h)\, c(\mathrm{OPT})$$

Regarding whether $\hat{F}$ is feasible, we know from Theorem 3.33 that each group is covered in $\mathcal{Y}_i$ with probability $1/(\alpha\,\mathrm{height}(\widetilde{\mathcal{T}}))$. By Theorem 3.33, $\mathcal{Y}_i$ covering a group implies that F_i also covers it. Hence, if a group is covered by any $\mathcal{Y}_i$, it is covered in $\hat{F}$. Over ℓ samples, the probability that a group $\widetilde{S}_j$ is *not* covered (by any $\mathcal{Y}_i$) is at most

$$\begin{aligned}\left(1 - \frac{1}{\alpha\,\mathrm{height}(\widetilde{\mathcal{T}})}\right)^{\ell} &= \left(1 - \frac{1}{\alpha\,\mathrm{height}(\widetilde{\mathcal{T}})}\right)^{3\alpha\,\mathrm{height}(\widetilde{\mathcal{T}})\log h} \\ &\le e^{-3\log h} \\ &\le h^{-3}\end{aligned}$$

By union bound, the probably that any group is not covered is at most $hh^{-3} = h^{-2}$. Thus with high probability all the groups will be covered. If there is a group which is not covered, it can be covered by taking a shortest path from the root to the closest group element, without affecting the expected cost (due to the low probability that this occurs).

Now that we know how to use the rounding to obtain an approximate solution, we shall prove Theorem 3.38.

Proof of Theorem 3.38. We start by showing the changes that are necessary to the LP. In order to ensure that the solution obtained is a solution tree, we modify the LP as follows.

Definition 3.39 (Cut-LP for STGST (LP-STGST))**.**

$$\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e \\
\text{s. t.} \quad & \sum_{e \in \delta(U)} x_e \geq 1 && \forall\, U \subseteq V : \tilde{r} \in U, U \cap \widetilde{S}_i = \emptyset \text{ for some } i \\
& x_{(p(v),v)} = x_{(v,u)} && \forall\, v \in \widetilde{\mathcal{T}}_c, u \in C(v) && (1) \\
& x_{(p(v),v)} = \sum_{u \in C(v)} x_{(v,u)} && \forall\, v \in \widetilde{\mathcal{T}}_p : C(v) \neq \emptyset && (2) \\
& x_e \geq 0 && \forall\, e \in E
\end{aligned}$$

The above LP is similar to the Cut-LP in Section 3.2, with additional Constraints (1), (2). Constraint (1) ensures that, for every combination node $\tilde{t}$, if $\tilde{t}$ is added to the solution, then it has full out-degree. Constraint (2) similarly ensures the degree properties of subproblem nodes in $\widetilde{\mathcal{T}}_p$: if such a node $\tilde{t}$ is in the solution (and it is not a leaf), then exactly one outgoing edge should also be in the solution.

The rounding algorithm is similar to GKR rounding (see Section 3.2). One change is needed, in order to ensure the correct degree for subproblem nodes. Specifically, for a node $\tilde{t} \in \widetilde{\mathcal{T}}_p$ that is in the solution, whose single incoming edge is e, only one edge from $\tilde{t}$ is added to the solution. This edge is chosen randomly[1] from all the outgoing edges of $\tilde{t}$, each with probability $x_{e'}/x_e$. We show the modified algorithm in Algorithm A.2.

All that we need to show now is that this small change does not affect the analysis of the GKR algorithm. For combination nodes $\tilde{t} \in \widetilde{\mathcal{T}}_c$, we simply use the normal GKR rounding. By Constraint (1), $x_{e'} = x_e$, which means that, if e is in the solution $\mathcal{Y}$, then e' is picked with probability 1. As to subproblem nodes $\tilde{t} \in \widetilde{\mathcal{T}}_p$, we show that removing them from $\widetilde{\mathcal{T}}_p$ (which causes the algorithm to use normal GKR rounding) affects rounding negatively, that is, the rounding procedure performs better with nodes in $\widetilde{\mathcal{T}}_p$.

Fix a group $\widetilde{S}_i$. We will show that if we remove a single node $\tilde{t} \in \widetilde{\mathcal{T}}_p$ from $\widetilde{\mathcal{T}}_p$, the probability that $\widetilde{S}_i$ is covered does not increase. By repeatedly removing nodes from $\widetilde{\mathcal{T}}_p$, this set eventually becomes empty, and therefore our algorithm is reduced to simple GKR. Since the probability of covering $\widetilde{S}_i$ is non-increasing throughout this process, the lower bounds on the probability for GKR rounding also apply to our modified rounding.

[1] The edge can be picked by generating a real number $r \in [0, 1)$; each edge e_i is picked if $\sum_{j=1}^{i-1} x(e_j)/x_e \leq r < \sum_{j=1}^{i} x(e_j)/x_e$.

Lemma 3.40. *Let $\tilde{t} \in \widetilde{\mathcal{T}}_p$ be a topmost node in $\widetilde{\mathcal{T}}_p$ and $\widetilde{S}_i$ be a group. Then*

$$\Pr[\textsc{RoundMod}(\widetilde{\mathcal{T}}, \widetilde{\mathcal{T}}_p, x) \cap \widetilde{S}_i] \geq \Pr[\textsc{RoundMod}(\widetilde{\mathcal{T}}, \widetilde{\mathcal{T}}_p \setminus \tilde{t}, x) \cap \widetilde{S}_i]$$

Proof. Let $e = \big(p(\tilde{t}), \tilde{t}\big)$, $p(e) = \big(p(p(\tilde{t})), p(\tilde{t})\big)$ be its parent edge, and let $\mathcal{Y}$, $\mathcal{Y}'$ be random variables representing the solutions obtained from the calls to $\textsc{RoundMod}(\widetilde{\mathcal{T}}, \widetilde{\mathcal{T}}_p, x)$, and $\textsc{RoundMod}(\widetilde{\mathcal{T}}, \widetilde{\mathcal{T}}_p \setminus \{\tilde{t}\}, x)$, respectively. Let $\mathcal{Y}_e$, $\mathcal{Y}'_e$ be the solutions $\mathcal{Y}$, $\mathcal{Y}'$ restricted to the sub-tree consisting of e and descendant edges.

Notice that it is sufficient to prove that $\Pr[\mathcal{Y}_e \cap \widetilde{S}_i] \geq \Pr[\mathcal{Y}'_e \cap \widetilde{S}_i]$, since other parts of the solution ($\mathcal{Y} \setminus \mathcal{Y}_e$ and $\mathcal{Y}' \setminus \mathcal{Y}'_e$) are not affected by $\tilde{t}$, and therefore, the probability that a group gets connected is similar for $\mathcal{Y} \setminus \mathcal{Y}_e$ and $\mathcal{Y}' \setminus \mathcal{Y}'_e$.

Let $\mathcal{F}(\mathcal{Y})$ be the event that $\mathcal{Y} \cap \widetilde{S}_i = \emptyset$, that is, that the group $\widetilde{S}_i$ is *not* connected in $\mathcal{Y}$. Let $f_{e'} = \Pr[\mathcal{F}(\mathcal{Y}_{e'}) \mid e' \in \mathcal{Y}]$, $f'_{e'} = \Pr[\mathcal{F}(\mathcal{Y}'_{e'}) \mid e' \in \mathcal{Y}']$.

Conditioning on $p(e) \in \mathcal{Y}$, we obtain the probability of $\mathcal{F}(\mathcal{Y}_e)$ using the complement. $\overline{\mathcal{F}}(\mathcal{Y}_e)$ occurs if e is chosen, then one of the edges $e' \in C(e)$ is chosen, and for that edge, the event $\overline{\mathcal{F}}(\mathcal{Y}_{e'})$ occurs. Since only one edge $e' \in C(e)$ is picked, these events are disjoint. Formally,

$$\begin{aligned}
\Pr[\mathcal{F}(\mathcal{Y}_e) \mid p(e) \in \mathcal{Y}] &= 1 - \frac{x_e}{x_{p(e)}} \Pr[\overline{\mathcal{F}}(\mathcal{Y}_e) \mid e \in \mathcal{Y}] \\
&= 1 - \frac{x_e}{x_{p(e)}} \sum_{e' \in C(e)} \frac{x_{e'}}{x_e} \Pr[\overline{\mathcal{F}}(\mathcal{Y}_{e'}) \mid e' \in \mathcal{Y}] \\
&= 1 - \frac{x_e}{x_{p(e)}} \sum_{e' \in C(e)} \frac{x_{e'}}{x_e} (1 - f_{e'})
\end{aligned}$$

The probability that $\mathcal{F}(\mathcal{Y}'_e)$ occurs can be defined similarly: it is the complementary probability to $\overline{\mathcal{F}}(\mathcal{Y}'_e)$, which occurs if e is chosen, and then the complement of all the independent $e' \in C(e)$ either not being selected, or being selected and $\mathcal{F}(\mathcal{Y}'_{e'})$ occurring. Formally, we have,

$$\begin{aligned}
\Pr[\mathcal{F}(\mathcal{Y}'_e) \mid p(e) \in \mathcal{Y}'] &= 1 - \frac{x_e}{x_{p(e)}} \Pr[\overline{\mathcal{F}}(\mathcal{Y}'_e) \mid e \in \mathcal{Y}'] \\
&= 1 - \frac{x_e}{x_{p(e)}} \left(1 - \prod_{e' \in C(e)} \left(1 - \frac{x_{e'}}{x_e} \Pr[\overline{\mathcal{F}}(\mathcal{Y}'_{e'}) \mid e' \in \mathcal{Y}']\right)\right) \\
&= 1 - \frac{x_e}{x_{p(e)}} \left(1 - \prod_{e' \in C(e)} \left(1 - \frac{x_{e'}}{x_e} (1 - f'_{e'})\right)\right)
\end{aligned}$$

We need two more ingredients to complete the proof: the first is the observation that $f_{e'} = f'_{e'}$, for every $e' \in C(e)$, since both instances are similar under the condition that e' is in the solution; the second is the following simple inequality.

Claim 3.41. *Let $a_i \in [0, 1]$, $i \in [n]$. Then,*

$$\prod_{i=1}^{n} (1 - a_i) \geq 1 - \sum_{i=1}^{n} a_i$$

Proof. The proof follows by induction. Notice that, for $n = 1$, the left and right hand sides are equal to $1 - a_1$, and so the claim follows. Assume now that the claim holds for $n - 1$ elements. Then, by induction hypothesis, we have:

$$\begin{aligned}\prod_{i=1}^{n}(1-a_i) &= (1-a_n)\prod_{i=1}^{n-1}(1-a_i)\\ &\geq (1-a_n)\left(1-\sum_{i=1}^{n-1}a_i\right)\\ &= 1-a_n-\sum_{i=1}^{n-1}a_i+a_n\sum_{i=1}^{n-1}a_i\\ &\geq 1-\sum_{i=1}^{n}a_i\end{aligned}$$

□

Using this claim, we can show that $\Pr[\mathcal{F}(\mathcal{Y}'_e) \mid p(e) \in S'] \geq \Pr[\mathcal{F}(\mathcal{Y}_e) \mid p(e) \in S]$:

$$\begin{aligned}\Pr[\mathcal{F}(\mathcal{Y}'_e) \mid p(e) \in S'] &= 1-\frac{x_e}{x_{p(e)}}\left(1-\prod_{e'\in C(e)}\left(1-\frac{x_{e'}}{x_e}\big(1-f'_{e'}\big)\right)\right)\\ &\geq 1-\frac{x_e}{x_{p(e)}}\left(1-\left(1-\sum_{e'\in C(e)}\frac{x_{e'}}{x_e}\big(1-f'_{e'}\big)\right)\right)\\ &= 1-\frac{x_e}{x_{p(e)}}\sum_{e'\in C(e)}\frac{x_{e'}}{x_e}(1-f_{e'})\\ &= \Pr[\mathcal{F}(\mathcal{Y}_e) \mid e \in S']\end{aligned}$$

Finally, since $\Pr\left[\mathcal{Y}_e \cap \widetilde{S}_i \mid p(e) \notin \mathcal{Y}\right] = 0$ (also for $\mathcal{Y}'_e$), then

$$\begin{aligned}\Pr[\mathcal{Y}_e \cap \widetilde{S}_i] &= \Pr[p(e) \in \mathcal{Y}]\Pr[\mathcal{Y}_e \cap \widetilde{S}_i \mid p(e) \in \mathcal{Y}]\\ &= \Pr[p(e) \in \mathcal{Y}](1-\Pr[\mathcal{F}(\mathcal{Y}_e) \mid p(e) \in \mathcal{Y}])\\ &\geq \Pr[p(e) \in \mathcal{Y}']\big(1-\Pr[\mathcal{F}(\mathcal{Y}'_e) \mid p(e) \in \mathcal{Y}']\big)\\ &= \Pr[p(e) \in \mathcal{Y}']\Pr[\mathcal{Y}'_e \cap \widetilde{S}_i \mid p(e) \in \mathcal{Y}']\\ &= \Pr[\mathcal{Y}'_e \cap \widetilde{S}_i]\end{aligned}$$

This completes the proof of the lemma. □

The analysis of GKR [GKR00] is resilient to constraints added to the LP, and our modified rounding does not decrease the probability of covering a group. Therefore, the same analysis follows, which completes the proof of the theorem. □

3.5 Group Steiner Forest on Bounded-Treewidth Graphs

In this section, we will show how to adapt the ideas used in Section 3.4 to the group Steiner forest (GSF) problem. Despite its similarity to GST, this problem is harder

to solve because of its more general structure. In the non-group setting, the Steiner forest problem is NP-hard on graphs of treewidth 3 [Gas10], even though Steiner tree is fixed-parameter tractable when parameterized by the treewidth.

Our algorithm is inspired by the work of Chekuri et al. [CEG+11]. The main insight is that the solution is a forest, and therefore we can obtain a good solution by taking the union of multiple trees. Specifically, at each step, we find a good approximation of the tree with the best density, that is, the one that covers the most pairs of groups with the least cost. By repeating this process until all pairs are covered, we obtain a feasible solution to the problem, at the cost of a $(1 + \ln h)$ factor.

In their work, Chekuri et al. [CEG+11] use an adaptation of the GKR algorithm to give an $O(\log^2 n \log^2 h)$-approximation to the problem in general graphs. We show that our algorithm for GST on bounded-treewidth can also be adapted to the density setting, which leads to an $O(\log n \log^2 h)$-approximation to the problem in bounded-treewidth graphs.

Formally, we reduce the GSF problem to the problem of finding a tree with minimum density, which minimizes the ratio of the cost of the edges to the number of group pairs connected. We call this problem *minimum-density group Steiner forest* (MDGSF).

Problem 3.42: Minimum-Density Group Steiner Forest (MDGSF).

- INSTANCE: $\big(G, c, r, \mathcal{P}\big)$, where
  - $G = (V, E)$ is a graph with edge costs $c : E \to \mathbb{R}$;
  - $r \in V$ is the *root* of the solution;
  - $\mathcal{P} = \{(A_i, B_i)\}_{i \in [h]}$ is a collection of *group pairs* $(A_i, B_i) \subseteq V \times V$, $i \in [h]$.
- SOLUTION: a tree $F \subseteq E$, rooted at r.
- GOAL: minimize the density of F, $d(F) = c(F)/h(F)$, where
  - $c(F) = \sum_{e \in F} c_e$ is the cost of F;
  - $h(F) = \big|\{i \in [h] : F \text{ connects } a_i \in A_i \text{ to } b_i \in B_i\}\big|$ is the number of group pairs covered by F.

  If $h(F) = 0$, then $d(F) = +\infty$ by convention.
- *Remark:* We can assume that F is a tree by Observation 3.46, and therefore, we can assume that the root r is given.

The theorem below is a more complete version of Theorem 3.6, and formalizes the results of this section.

Theorem 3.43. *There is an $O(\log n \log h)$-approximation algorithm for MDGSF with running time $n^{O(w \log w)}$, where w is the treewidth of the input graph. This implies an $O(\log n \log^2 h)$-approximation algorithm for GSF with the same asymptotic running time.*

The proof of the theorem follows directly by combining the following two results.

Lemma 3.44. *Let $\big(G, c, \mathcal{P}\big)$ be an instance of GSF, and let $\mathcal{A}$ be an algorithm with approximation ratio $\alpha(n, h)$ for all instances of MDGSF of the form $\big(G, c, r, \mathcal{P}'\big)$, with $\mathcal{P}' \subseteq \mathcal{P}$, $r \in V(G)$.*

Then, we can compute an $(\alpha(1 + \ln h))$-approximation for GSF instance $\big(G, c, \mathcal{P}\big)$ in polynomial time and with at most $O(hn)$ calls to $\mathcal{A}$.

Theorem 3.45. *Let $\mathcal{I} = \big(G, c, r, \mathcal{P}\big)$ be an instance of MDGSF.*

There is an algorithm that runs in time $n^{O(w \log w)}$, where w is the treewidth of G, and that computes an $O(\log n \log h)$-approximate solution to $\mathcal{I}$.

The proof of Lemma 3.44 follows by standard counting arguments. We provide the proof for completeness.

Proof of Lemma 3.44. The algorithm for GSF (represented in Algorithm A.3) can be described as follows: Initialize the solution $\hat{F}$ as the empty set. While some group pairs are not yet covered, call $\mathcal{A}$ on G with the uncovered pairs, trying every possible root. From all the solutions obtained, take the one with the lowest density, and add it to $\hat{F}$. We denote by F_i the i-th such lowest-density solution that is added to $\hat{F}$. When all the groups are covered, output F.

Since the solution obtained by $\mathcal{A}$ always covers at least one of the group pairs, after at most h iterations, all the group pairs must be covered. Since we need to call $\mathcal{A}$ for every possible choice of root, we need to call $\mathcal{A}$ at most $h\,n$ times in total.

Let $c(F_i)$, $h(F_i)$ be the cost and number of group pairs covered by F_i, respectively. For convenience, we denote $\mathrm{OPT} := \mathrm{OPT}(G, c, \mathcal{P})$ to be an optimum solution for the GSF instance. We have that:

$$\begin{aligned}
c(\hat{F}) &= \sum_{i=1}^{q} c(F_i) = \sum_{i=1}^{q} h(F_i)\frac{c(F_i)}{h(F_i)} \\
&\leq \sum_{i=1}^{q} h(F_i)\alpha\frac{c(\mathrm{OPT})}{h - \sum_{j=1}^{i-1} h(F_j)} \\
&\leq \sum_{\ell=0}^{h-1} \alpha\frac{c(\mathrm{OPT})}{h-\ell} \\
&= \alpha c(\mathrm{OPT}) \sum_{\ell=1}^{h} \frac{1}{\ell} \\
&\leq \alpha c(\mathrm{OPT})(1 + \ln h)
\end{aligned}$$

The first inequality follows from the observation that before the i-th iteration $\hat{F}$ covers $\sum_{j=1}^{i-1} h(F_j)$ group pairs, and there are $h - \sum_{j=1}^{i-1} h(F_j)$ left. Since OPT covers these groups with cost $c(\mathrm{OPT})$, the density of F_i has to be at most α times the density of OPT on the remaining groups.

The second inequality follows by considering the density with which each group is covered: for the first $h(F_1)$ groups, the density is at most $\alpha c(\mathrm{OPT})/h$, for the next $h(F_2)$ groups, the density is at most $\alpha c(\mathrm{OPT})/(h - h(F_1))$ and so on. We can then upper bound the density of the ℓ-th group pair covered as $\alpha c(\mathrm{OPT})/(h-\ell)$, by replacing the denominator with a number that is at most as large.

With these two observations, the rest of the proof follows. □

3.5.1 Approximating Minimum-Density Group Steiner Forest

This section will focus on the proof of Theorem 3.45. The following observation shows that MDGSF always has an optimal solution that is a tree, and therefore we can simply search for a tree with approximately optimal density in our algorithm.

Observation 3.46. *For any instance of MDGSF, there is an optimal solution that is a tree.*

Proof. The observation follows in two steps: first, if the solution has cycles, we can repeatedly remove an edge from a cycle until the solution is a forest. This operation does not increase the cost, and does not affect connectivity between groups, so the density does not increase.

The second step is to consider the density of individual trees in the solution F. If we have multiple components F_i with density $d(F_i)$, cost $c(F_i)$, and $h(F_i)$ group pairs covered, we can take $i^* = \operatorname{argmin}_i d(F_i)$, and obtain that:

$$d(F) = \frac{\sum_i c(F_i)}{\sum_i h(F_i)} = \frac{\sum_i h(F_i)\, d(F_i)}{\sum_i h(F_i)} \geq \frac{\sum_i h(F_i)\, d(F_{i^*})}{\sum_i h(F_i)} \geq d(F_{i^*})$$

Therefore, if F is optimal, F_{i^*} is optimal too. □

Algorithm A.4 details the steps on how to get the desired approximation. The main idea is reduce to MDGSF to a variant of the STGST problem used in Theorem 3.33, and then apply an algorithm similar to the one by Chekuri et al. [CEG+11], which solves an appropriate LP relaxation and rounds it to obtain a solution. By using a rounding procedure similar to Section 3.4.2, we can guarantee that the density of the solution is a good approximation. We remark that to solve MDGSF, it is sufficient to repeat the rounding procedure $O(\log h)$ times, so that the solution covers a constant fraction of the groups covered by the LP solution.

We start by introducing an LP relaxation for the variant of STGST, which we call MDSTGSF. The difference between these two problems is that in MDSTGSF, our goal is to minimize the density of the solution. We denote as $\widetilde{A}_i$, $\widetilde{B}_i$ the groups in $\widetilde{\mathcal{S}}$ corresponding to A_i, B_i.

Definition 3.47 (Cut-LP for MDSTGSF – LP-STGSF)**.**

$$\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e & & \\
\text{s. t.} \quad & \sum_{e \in \delta(S)} x_e \geq y_i & & \forall\, S \subseteq V : \tilde{r} \in S, S \cap \widetilde{A}_i = \emptyset \text{ for some } i \\
& \sum_{e \in \delta(S)} x_e \geq y_i & & \forall\, S \subseteq V : \tilde{r} \in S, S \cap \widetilde{B}_i = \emptyset \text{ for some } i \\
& x_{(p(v),v)} = x_{(v,u)} & & \forall\, v \in \widetilde{\mathcal{T}}_c, u \in C(v) \qquad (1) \\
& x_{(p(v),v)} = \sum_{u \in C(v)} x_{(v,u)} & & \forall\, v \in \widetilde{\mathcal{T}}_p \qquad (2) \\
& \sum_{i=1}^{h} y_i = 1 & & \\
& x_e \geq 0 & & \forall\, e \in E
\end{aligned}$$

We can see that LP-STGSF is a relaxation of MDSTGSF in the following sense: for any solution tree F for the MDSTGSF, we can set $x = \chi_F / h(F)$, where $\chi_F \in \{0,1\}^E$ is

the indicator vector for F ($\chi_F(e) = 1$ if and only if $e \in F$), and $h(F)$ is the number of group pairs connected by F. We set $y_i = 1/h(F)$ if F connects A_i to B_i, and $y_i = 0$ otherwise. This solution (x, y) is feasible for LP-STGSF: degree constraints are satisfied, since the x is just the scaled-down indicator vector of F. Similarly, since A_i and B_i are connected, we know that every cut between them, or in this case, between them and the root is crossed by at least one edge of F. Therefore, $x(\delta(S)) \geq 1/h(F) = y_i$ for all such cuts. The sum of the values y_i must equal 1, since all the non-zero y_i have the same value, $1/h(F)$, and there are $h(F)$ non-zero such values. Finally, let us show that the objective value is the density of F:

$$\sum_{e \in E} c_e x_e = \sum_{e \in F} c_e \frac{1}{h(F)} = \frac{1}{h(F)} \sum_{e \in F} c(e) = d(F)$$

Let us now show how to round the solution. Let $\mathcal{B}_j = \left\{ i : 2^{-j} < y_i \leq 2^{-(j-1)} \right\}$, for $j \in [2 \log h]$. We will now show that by ignoring group pairs that are not in any $\mathcal{B}_j$, we don't lose too much "coverage", since these groups have very small values of y_i.

$$\begin{aligned}
\sum_{j \in [2 \log h]} \sum_{i \in \mathcal{B}_j} y_i &= \sum_j y(\mathcal{B}_j) - \sum \left\{ y_i : y_i \leq \frac{1}{h^2} \right\} \\
&\geq 1 - \frac{1}{h^2} \left| \left\{ y_i : y_i \leq \frac{1}{h^2} \right\} \right| \\
&\geq 1 - \frac{1}{h} \\
&\geq \frac{1}{2}
\end{aligned}$$

By an averaging argument, there is a j^* such that $\sum_{i \in \mathcal{B}_j} y_i \geq 1/(4 \log h)$. Our strategy will be to focus on $\mathcal{B}_{j^*}$, and prove that, by rounding the solution multiple times, we can obtain a solution that covers a constant fraction of the group pairs in $\mathcal{B}_j$.

Let $\mathcal{B} = \mathcal{B}_{j^*}$. We use a variation of Theorem 3.38 to round the solution. Its proof, combined with Lemma 3.13, implies that if the min-cut separating the root from a group has value at least y_i, then the group is covered with probability $y_i / \text{height}(\widetilde{\mathcal{T}})$. Since $y_i \geq 1/2^{j^*}$ for $i \in \mathcal{B}$, we apply Theorem 3.38 $\ell = 3 \cdot 2^{j^*} \text{height}(\widetilde{\mathcal{T}})$ times to obtain solutions $\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_\ell$. Then, apply Property (2) of Theorem 3.33 to each $\mathcal{Y}_i$ to obtain a tree F_i in the original graph G. Finally, take the union of all F_i to get the solution $F = \cup_i F_i$, removing extra edges if necessary to make it a tree.

The expected cost, by Theorem 3.38, is at most $\ell c^T x = O(2^{j^*} \log n) d(\text{OPT})$. Furthermore, since $y(\mathcal{B}) \geq 1/(4 \log h)$ but $y_i \leq 1/2^{j^*-1}$ for $i \in \mathcal{B}$, then $|\mathcal{B}| \geq 2^{j^*}/(8 \log h)$. Therefore, if we connect a constant fraction α of the group pairs in $\mathcal{B}$, the density of the solution F is

$$d(F) \leq \frac{\ell c^T x}{\alpha |\mathcal{B}|} = \frac{3 \cdot 2^{j^*} c^T x \log n}{\alpha 2^{j^*}/(8 \log h)} = O(\log n \log h)\, c^T x = O(\log n \log h)\, d(\text{OPT})$$

All that is left to prove is that we connect a constant fraction of the group pairs in $\mathcal{B}$. Over ℓ samples, the probability that a group A_j or B_j is *not* covered (by any $\mathcal{Y}_i$) is

at most

$$\left(1 - \frac{y_j}{height(\widetilde{\mathcal{T}})}\right)^{\ell} = \left(1 - \frac{y_j}{height(\widetilde{\mathcal{T}})}\right)^{3 \cdot 2^{j^*} \text{height}(\widetilde{\mathcal{T}})} \leq e^{-3 \cdot 2^{j^*} y_j} \leq e^{-3} \leq \frac{1}{20}$$

Applying union bound, the probability that we do not simultaneously cover both A_j and B_j is $2/20$. Hence, we conclude that $9/10|\mathcal{B}|$ group pairs are covered in expectation, which implies that the density of F is

$$d(F) = O(\log n \log h) d(\text{OPT})$$

More formally, the probability that the cost exceeds $4 \cdot \ell \cdot c^T x$ is at most $1/4$, by Markov's law. Similarly, the probability that we leave more than $4/10|\mathcal{B}|$ groups uncovered is at most $1/4$. By inclusion-exclusion principle, with probability $1/2$, we cover at least $6/10|\mathcal{B}|$ group pairs with cost at most $4 \cdot \ell \cdot c^T x$. For convenience, we assume that at least one group pair gets connected. Otherwise, we can repeat the rounding until at least one group is covered.

We conclude that, with probability $1/2$

$$d(F) \leq \frac{4 \cdot \ell \cdot c^T x}{6/10|\mathcal{B}|} = O(\log n \log h) d(\text{OPT})$$

This concludes the proof of Theorem 3.45.

3.6 Directed Steiner Forest on Bounded-Treewidth Graphs

In this section, we will focus on solving directed Steiner forest on a graph G with bounded *undirected* treewidth, that is, the treewidth of the undirected graph obtained by removing directionality from edges. We will use the same approach to the problem as used by Chekuri et al. [CEG+11], who show that we can solve the problem by finding junction trees with good density (i.e. the ratio between total cost and number of pairs connected).

A *junction tree* is the union of an in-arborescence and an out-arborescence rooted at some vertex r. The idea behind junction trees is that there is a unique path from any vertex in the in-arborescence to any vertex in the out-arborescence, going through r. Thus, junction trees serve as an analogue to the trees that a group Steiner forest solution can be partitioned into.

Solving the problem can now be reduced to the following two tasks:

(1) Proving that a junction tree with good density exists;

(2) Finding a junction tree with approximately optimal density.

We will omit the details of the second part, as it is very similar to the algorithm of Section 3.5. Since Lemma 3.23 (local-global connectivity equivalence) also applies for directed graphs, there is an equivalent of Section 3.4.1 for directed graphs. Therefore,

we can use ideas similar to those of Section 3.5, where we want to find a solution that connects groups A_i to the root, and the root to groups B_i, and we consider all the groups to have size 1.

As to the first step, we will show that if the underlying undirected graph has treewidth w, there is always a junction tree of density $O(w \log n)d(\mathrm{OPT})$. Since we can find a junction tree which is an $O(\log n \log h)$-approximation to the density of the best junction tree, we can apply our algorithm to find a junction tree with density $O(w \log^2 n \log h)d(\mathrm{OPT})$.

Similarly to Section 3.5, by iteratively covering pairs using junction trees with good density, we can cover all the groups while incurring a loss of $O(\log h)$ in the approximation factor. Putting all the pieces together, we can find an $O(w \log^2 n \log^2 h)$-approximation to directed Steiner forest.

The goal of this section is to prove Theorem 3.7, which we recall below.

Theorem 3.7 (page 21). *There is an $O(w \log^2 n \log^2 h)$-approximation algorithm for DSF with running time $n^{O(w^2)}$, where w is the treewidth of the undirected version of the input graph.*

The following lemma, together with the techniques of Section 3.5, imply the Theorem 3.7.

Lemma 3.48. *Let (G, c, R) be an instance of DSF, where G is a directed graph with treewidth w and $R = \{(a_i, b_i) : i \in [h]\}$ are terminal pairs.*

There is a junction tree $J \subseteq G$ with density $O(w \log n)d(\mathrm{OPT})$, where OPT *is the subgraph $H \subseteq G$ which minimizes $d(H)$.*

Proof. Let $H^* \subseteq G$ be the optimal solution to DSF on G, and $(\mathcal{T}, X)$ be a tree decomposition as in Lemma 2.6. For each $i \in [h]$, we denote by $P(a_i, b_i)$ the shortest path in H^* between a_i and b_i, and by t_i the highest node in $\mathcal{T}$ such that X_{t_i} intersects $P(a_i, b_i)$. For each (a_i, b_i), $i \in [h]$, we say that X_{t_i} is the *head bag*, and $r_i \in X_{t_i} \cap P(a_i, b_i)$ (picked arbitrarily) is the *head vertex*.

We will now show that there is a subgraph $H = H^* \cap \mathcal{T}_{t_i}$ with density $O(\log n)d(H^*)$. Since $\mathcal{T}$ has height $O(\log n)$, there is a level ℓ of the tree that contains the head bags for $\Omega(h/\log n)$ pairs (by pigeon-hole principle). Let $X_{t_1}, X_{t_2}, \ldots, X_{t_p}$ be all the p head bags at level ℓ, and let $H_i = H^* \cap \mathcal{T}_{t_i}$ (for $i \in [p]$). We consider a unique copy of each bag, even if it is the head bag for multiple terminal pairs. We know that $\bigcup_{i \in [p]} H_i$ connects at least $\Omega(h/\log n)$ pairs, by the choice of ℓ and the definition of H_i. Since all of the H_i are subgraphs of H^*, their union costs as much as H^*, and therefore, the density of $\bigcup_{i \in [p]} H_i$ is at most $O(\log n)d(H^*)$. Furthermore, all the H_i are pairwise edge-disjoint, since all the sets $E(\mathcal{T}_{t_i})$ are disjoint, $i \in [p]$. By an averaging argument, we conclude that one of the H_i, $i \in [p]$, has density at most $O(\log n)d(H^*)$, as desired.

We can now finish the proof of the lemma. Since $|X_{t_i}| \leq O(w)$, there is a vertex $r_j \in X_{t_i}$ that is the head vertex for $\Omega(h(H_i)/w)$ pairs, where $h(H_i)$ is the number of pairs connected by H_i. Let $Q_j = \{q \in [h] : r_q = r_j\}$ be the set of pairs that have r_j as their head vertex, and $J = \bigcup_{q \in Q_j} P(a_q, b_q)$. We claim that J satisfies the conditions of the lemma.

We start by showing that J must be a junction tree, because it is the union of paths that all go through r_j. Let $P(a_q, b_q)$ be partitioned into P_{1q} and P_{2q}, where P_{1q} goes from

a_q to r_j and P_{2q} from r_j to b_q, for all $q \in Q_j$. To satisfy the definition of junction tree, we simply need that $J_1 = \bigcup_{q \in Q_j} P_{1q}$ is an in-arborescence rooted at r_j and $J_2 = \bigcup_{q \in Q_j} P_{2q}$ is an out-arborescence rooted at r_j. Indeed, the paths P_{1q} all go from some vertex to r_j, and paths P_{2q} go from r_j to some other vertex. Therefore, we can easily transform J_1 and J_2 into arborescences by removing extraneous edges. If J_1 (resp. J_2) has any cycles, remove the edge of the cycle outgoing from the vertex closest to (resp. from) r_j. Additionally, for every vertex of J_1 (resp. J_2), we remove outgoing (resp. ingoing) edges until only one is left. It can be easily verified that J_1 and J_2 are now arborescences, and thus $J = J_1 \cup J_2$ is a junction tree.

Regarding the density of J, if $r_q = r_j$, then $P(a_q, b_q) \subseteq H_i$. Therefore, $c(J) \leq c(H_i)$. Furthermore, J connects at least $\Omega(h(H_i)/w)$ pairs, so

$$d(J) \leq O\left(\frac{c(H_i)}{h(H_i)/w}\right) = O(w)d(H_i) \leq O(w \log n)d(H^*) \qquad \square$$

By proving the existence of junction trees with density $O(w \log n)d(\mathrm{OPT})$, we can use the techniques of Section 3.5 to find an $O(\log n \log h)$-approximate junction tree. These results imply an $O(w \log^2 n \log^2 h)$-approximation, which runs in time $n^{O(w^2)}$.

CHAPTER 4

Fault-Tolerant Group Steiner Tree

In network design, a setting of particular interest is that of fault-tolerant design. In many applications, we often want to ensure that the network stays operational, even when some failures occur (see e.g. [MSD+96] or [KV18, Chapter 20] and references within). For instance, it would be catastrophic if one server or cable failure would cause the internet to fail in an entire country. In the fault-tolerant setting, we require connectivity to be redundant, by having multiple paths which are independent from each other. If we have k independent paths between two nodes, connectivity is preserved even if the network suffers $k-1$ failures, as there must be at least one path which is not affected.

There are three variants of fault-tolerance that are usually considered in the literature: edge connectivity asks for resilience to edge failures, and thus independent paths must be edge-disjoint; vertex connectivity considers vertex failures, and asks for independent paths to be internally vertex-disjoint. Due to vertex connectivity being usually hard to approximate (see e.g. [CCK08]), Jain et al. [JMV+02] introduced the concept of element-connectivity. In this setting, we are given a set of terminals, and we are only interested in connectivity between the terminals. Motivated by the importance of the terminals in the network, Jain et al. take the assumption that these terminals cannot fail (or in any case, will be quickly repaired or replaced), and therefore independent paths can share terminals, but not edges or non-terminal vertices. This turned out to be an easier case to approach, while at the same time being useful in practice [FJW06, JMV+02].

One of the problems of interest in this field is the *survivable network design problem* (SNDP). In this problem, we are given a graph G with demands $r_{uv} \in \mathbb{Z}_{\geq 0}$, and we must find a minimum-cost subgraph that contains r_{uv} independent paths connecting u and v, for every $u, v \in V$. The term independent may mean edge-disjoint, vertex-disjoint, or element-disjoint, where the terminals are all the vertices participating in a pair with positive demand.

In a seminal paper, Jain [Jai01] gave a 2-approximation to the edge-connectivity variant of the problem. This result was later extended to element connectivity by Fleischer et al. [FJW06] and Cheriyan et al. [CVV06]. For the vertex-connectivity variant of the problem, Chuzhoy and Khanna present an $O(k^3 \log n)$-approximation to the problem [CK12]. Nutov studied SNDP with vertex costs, obtaining an $O(k \log n)$-approximation for the edge-connectivity problem [Nut10], and later as well for element-connectivity [Nut12]. From the perspective of hardness of approximation, Kortsarz et al. proved that the SNDP is $2^{\log^{1-\varepsilon} n}$-hard to approximate [KKL04], whereas Chakraborty et al. proved that the problem is $k^{\Omega}(1)$-hard to approximate [CCK08] (under some reasonable complexity assumptions).

While these results apply for the general SNDP problem, we are mostly interested in the rooted setting, where all the demands are between a root vertex r and another vertex. When considering edge connectivity, the setting where a set of terminals T have

uniform pairwise demands among themselves ($r_{uv} = k$ for $u, v \in T$) can be reduced to the rooted setting.

In this chapter, we study a generalization of rooted SNDP and GST, in which we want multiple edge-disjoint paths to connect the root and the groups. We will specifically focus on the edge connectivity variant of the *restricted group survivable network design problem* (GroupSNDP). In this problem, we are given a graph G with edge costs c, a root r, and a collection of h *groups* S_i, along with a *demand* k_i for each group. Our goal is to find a minimum-cost subgraph of G so that, for every group S_i, there is a vertex $v_i \in S_i$ such that there are k_i edge-disjoint paths between r and v_i.

The GroupSNDP problem focuses on a more difficult setting than SNDP, that of media broadcasting, such as in cable television or streaming service. In this case, we may want to connect a central server to local servers located in different communities. Each of these local servers can then redirect the signal or content to the clients in the area through a local network. In this scenario, we are given groups of vertices, and we simply need to connect a single representative vertex from each group. If we need a single path to each group, the problem turns into GST; for more details on this problem see Chapter 3.

The case in which the demands are at most 2 has been studied by Khandekar et al. [KKN12] and Gupta et al. [GKR10], culminating in an $\tilde{O}(\log^3 n \log h)$-approximation for the problem. Other than these results, there are no other approximation algorithms to the problem, even for demands of value 3. The problem is known to be as hard as the label cover problem on directed graphs [KKN12].

Chalermsook, Grandoni and Laekhanukit [CGL15] studied the relaxed group SNDP problem, where the k_i paths for a group S_i can connect r to different vertices in S_i (whereas in GroupSNDP they must connect the *same* vertex). In this setting, they give a bicriteria algorithm that outputs an $O(\log^2 n \log h)$-approximate solution while guaranteeing connectivity of at least $\Omega(k_i / \log n)$ for each group. They also extended their algorithm to the unrooted version of the problem, where demands are given between a source and a target group (thus generalizing group Steiner forest). For this problem they give an $O(\log^4 n \log h)$ bicriteria approximation that guarantees connectivity $\Omega(k_i / \log n)$ for each group. They also show that the problem is label-cover-hard, that is, unless NP has quasi-polynomial time algorithms, there is no $2^{\log^{1-\varepsilon} n}$-approximation to these problems.

Mimicking networks and vertex sparsification As part of our approach to solve SNDP on bounded-treewidth graphs, we use *vertex sparsifiers* to represent connectivity in the graph. Vertex sparsification is the problem of reducing the number of vertices in a graph so that some property of interest is preserved for a special subset of vertices, denoted *terminals*. In our case, we are interested in preserving the value of cuts between subsets of these terminals: given a graph G with capacities w and a set of terminals T, we want to find a smaller graph H with capacities w', where the minimum cut separating any two disjoint subsets of terminals is preserved. Such a graph is called a *mimicking network* of G.

The first construction for a mimicking network was given by Hagerup et al. [HKN$^+$98], with size 2^{2^k} (where k is the number of terminals). Subsequently, two independent works by Krauthgamer and Rika [KR13], and Khan and Raghavendra [KR14] proved that there are graphs whose mimicking networks have size $2^{\Omega(k)}$. While there is some research on

mimicking networks in specific graph classes [CSW+00, KR13, GHP17, KR17, KPZ19], as well as in an approximation setting [CLL+10, MM10, Chu12], it remains an open question to close this gap for general graphs.

Related work The research community has dedicated considerable attention to (unrooted) SNDP, including in specific graph classes. For the Steiner forest problem, which corresponds to SNDP with unit demands, results are known for low-treewidth graphs [BHM11, CNP+11] and Euclidean graphs [BKM15]. For the general demand case, results are also known for graphs of bounded genus [BDT14], and metric-cost graphs [CV07].

Borradaile et al. [BDT14, BZ17] studied the variant of the problem in which one is allowed to buy multiple copies of edges. For demands $k_i \in \{0, 1, 2, 3\}$, they give an EPTAS for this problem on planar graphs. For the k-*ECSS* problem, where demands are uniform (every vertex $v \in V$ has demand k), Czumaj et al. [CGS+04] showed a PTAS for $k = 2$ in unweighted planar graphs; Berger and Grigni [BG07] showed an exact algorithm for 2-ECSS running in time $2^{O(w^2)}n$ where w is the treewidth of the input graph. Using this result, they obtain a PTAS for 2-ECSS on planar graphs.

The problem of finding a vertex sparsifier that approximately preserves cut values has been extensively studied; we refer to such a sparsifier as *quality-q sparsifier* (meaning that the value of each cut is q-approximated). When no extra vertices are allowed (i.e. the sparsifier has size k), the best known result is a quality-$O(\log k / \log\log k)$ sparsifier [CLL+10, MM10]. In this setting, there is a lower bound of $\Omega\big((\log k / \log\log k)^{1/4}\big)$ on the quality of sparsifiers [MM10]. Chuzhoy presents a sparsifier of quality $O(1)$ and size $O(C^3)$ [Chu12], where C is the total capacity of edges incident on the terminals. This sparsifier can be computed in time $\text{poly}(n)\, 2^C$, making it especially useful when the capacity into terminals is low.

On certain classes of graphs, the size of mimicking networks can be vastly improved. On planar graphs, Krauthgamer and Rika [KR13, KR17] showed how to construct a mimicking network of size $O(k2^{2k})$, matching the lower bound of Karpov et al. [KPZ19]. When all the terminals are on the same face of a planar graph, Goranci et al. [GHP17] present a construction of size $O(k^2)$. Chaudhuri et al. [CSW+00] show that graphs of treewidth w have mimicking networks of size $O(k2^{2^w})$.

Our results Our first result is a generalization of the techniques of Chapter 3 to the fault-tolerant setting. The main obstacle to solving SNDP is that the solution must support, for each connectivity requirement, multiple edge-disjoint or vertex-disjoint paths. However, the solution cannot simply be partitioned so that each part supports a disjoint path for each demand. Instead, possibly conflicting partitions might be required for different demands, and thus any algorithm must either take several partitions into consideration, or consider the connectivity of a solution in a different way. We show that in a bounded-treewidth setting, the number of ways that a solution can be partitioned, from the point of view of a single bag, is bounded as a function of the treewidth. Therefore, we can enumerate, for each bag, the partitions of interest, and compute the optimum solution using dynamic programming.

Using our framework of Section 3.3, we formulate a dynamic program for rooted SNDP, which leads to an FPT algorithm with running time $O(nf(w, K))$, where w is

the treewidth of the input graph and K is the maximum connectivity demand. Using the arguments of Section 3.4, we give an $O(\log n \log h)$-approximation algorithm for GroupSNDP that runs in time $n^{g(w,K)}$, where $g(K,w) = O(\log f(K,w))$. We remark that the approximation ratio is the best possible, as implied by the hardness result of Halperin and Krauthgamer [HK03]. Unfortunately, $f(K,w)$ is doubly-exponential in K and w.

For our second result, we represent connectivity in a different way, which does not require the enumeration of partitions. To achieve this goal, we introduce a new structure, called *connectivity-K mimicking network*, which is a specialization of mimicking networks to bounded-connectivity problems. Connectivity-K mimicking networks differ from the standard mimicking networks in two ways: (i) all edges have capacity 1; (ii) we are only interested in K-connectivity (for a given K), and hence we only preserve cuts of size less than K. We present a construction for connectivity-K mimicking networks of size $3^K Kw$ for a graph with w terminals, and use this construction to reduce the running time of our algorithm for SNDP to $n\,2^{\exp(K)w\log w}$ and of the approximation algorithm for GroupSNDP to $n^{\exp(K)w\log w}$. This running time matches our algorithm for GST (when $K = 1$), and can be further improved if smaller constructions for connectivity-K mimicking networks exist.

The tradeoff for improving the running time is that this approach currently only works for edge connectivity and edge costs, in undirected graphs. On the other hand, the result based on Section 3.3 applies to any combination of vertex or edge costs, vertex or edge connectivity and directed or undirected graphs.

We remark that when the cost is polynomially bounded (e.g., in the Word RAM model with words of size $O(\log n)$), polynomial-time algorithms for SNDP follow from Courcelle's theorem [Cou90, BPT92] (albeit, with much larger running time). However, employing the theorem as a black-box does not allow us to design approximation algorithms for GroupSNDP.

Organization We show two different approaches to solve the problem: the first one is an extension of the connectivity-based approaches of Section 3.3, and generalizes to many different problems. The results based on this technique are described in Section 4.2. The second approach introduces a new cut-based sparsifier for fault-tolerant settings, and we show how to use this technique to solve GroupSNDP in Section 4.3.

4.1 Problem Definitions and Results

Problem 4.1: Survivable Network Design Problem (SNDP).

- INSTANCE: $\big(G, c, r, S, k\big)$, where
  - $G = (V, E)$ is a graph with edge costs $c : E \to \mathbb{R}$;
  - $r \in V$ is the *root* of the instance;
  - $S = \{s_i\}_{i\in[h]}$ is the set of *terminals* $s_i \in V$, $i \in [h]$;
  - $k \in \mathbb{Z}^h$ is the demand vector, where k_i is the *demand* for terminal s_i, $i \in [h]$.
- SOLUTION: a subgraph $F \subseteq E$ such that, for every $i \in [h]$, F contains k_i edge-disjoint paths from r to s_i
- GOAL: minimize the cost of F, $\mathrm{c}(F) = \sum_{e\in F} c_e$

Problem 4.2: Restricted Group SNDP (GroupSNDP).

- INSTANCE: $\big(G, c, r, \mathcal{S}, k\big)$, where
  - $G = (V, E)$ is a graph with edge costs $c : E \to \mathbb{R}$;
  - $r \in V$ is the *root* of the instance;
  - $\mathcal{S} = \{S_i\}_{i\in[h]}$ is a collection of *groups* $S_i \subseteq V$, $i \in [h]$;
  - $k \in \mathbb{Z}^h$ is the demand vector, where k_i is the *demand* for group S_i, $i \in [h]$.
- SOLUTION: a subgraph $F \subseteq E$ where, for every $i \in [h]$, there is $v_i \in S_i$ such that F contains k_i edge-disjoint paths from r to v_i
- GOAL: minimize the cost of F, $\mathrm{c}(F) = \sum_{e\in F} c_e$

The problems defined above are the edge-cost, edge-connectivity version. Vertex-cost and/or vertex-connectivity variants of both problems can be defined analogously (for vertex connectivity, the paths must be internally vertex-disjoint).

Our results are as follows. We define $\exp(x) = 2^{O(x)}$ throughout the chapter.

The two following results apply to edge-cost, vertex-cost, edge-connectivity, vertex-connectivity, directed and undirected variants of the problems.

Theorem 4.3. *There is an exact algorithm for SNDP, with edge or vertex costs and edge or vertex connectivity, with running time $n2^{\exp(Kw^2)}$, where w is the treewidth of the input graph, and K is the maximum demand.*

Theorem 4.4. *There is an $O(\log n \log h)$-approximation algorithm for GroupSNDP, with edge or vertex costs and edge or vertex connectivity, with running time $n^{\exp(Kw^2)}$, where w is the treewidth of the input graph, and K is the maximum demand.*

The two following results apply to the edge-cost, edge-connectivity, undirected variant of the problem.

Theorem 4.5. *There is an exact algorithm for SNDP with running time $n2^{\exp(K)w\log w}$, where w is the treewidth of the input graph, and K is the maximum demand.*

Theorem 4.6. *There is an $O(\log n \log h)$-approximation algorithm for GroupSNDP with running time $n^{\exp(K)w\log w}$, where w is the treewidth of the input graph, and K is the maximum demand.*

4.2 Connectivity Lemma Based Approach

In order to demonstrate the applicability of the techniques developed in Section 3.4, we will now show how to extend the same ideas to solve GroupSNDP. Similarly to GST, we will show that GroupSNDP on bounded-treewidth graphs can be encoded as a special tree instance, which is obtained from a dynamic program related to that of SNDP. Therefore, we start by presenting a dynamic program for SNDP and proving Theorem 4.3. Afterwards, we briefly discuss how to use the techniques of Section 3.4.2 to obtain a solution to the GroupSNDP problem, proving Theorem 4.4.

In order to simplify the presentation and better show the parallelism between this section and Section 3.4, we focus on the case of edge connectivity, and edge costs. We then describe the changes necessary to prove the general results.

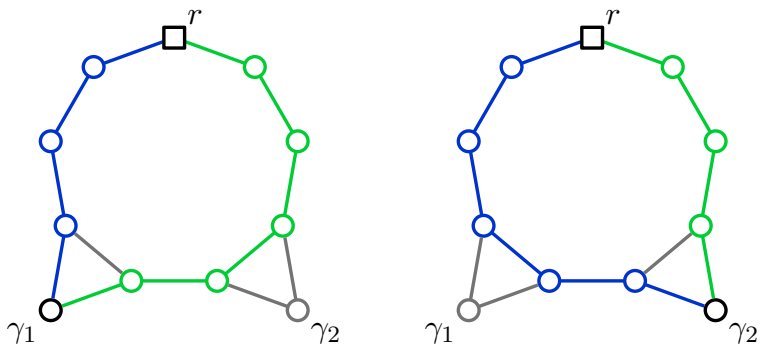

Figure 4.1: Example of different partitions of edges into paths from demands γ_1, γ_2. On the left (resp., right), two paths from r to γ_1 (resp., γ_2)

4.2.1 Dynamic Program for SNDP

In this section, we present a dynamic program for SNDP on bounded-treewidth graphs. In order to solve the problem, we use a similar strategy to the one we used for Steiner tree: we design a dynamic program that optimally chooses a state for each of the subinstances, which correspond to the nodes of the tree decomposition (see Sections 3.3.1 and 3.3.4 for more details).

Seeing as how in Steiner tree, the state is a pair (Γ, Δ), and SNDP is a high-connectivity version of this problem, we might think of using a scaled-up version of the same approach, such as,

$$(\Gamma_1, \Gamma_2, \ldots, \Gamma_K, \Delta_1, \Delta_2, \ldots, \Delta_K)$$

where $K = \max k_i$ is the maximum demand and each pair (Γ_i, Δ_i) would "increase" the connectivity by 1. Unfortunately, this strategy does not work, as this would be equivalent to partitioning our solution into edge-disjoint Steiner trees. While any solution obtained this way would be feasible, it is not guaranteed that the optimal solution is of that form, even for very simple cases. In Figure 4.1, we see an example of a solution in which the edges are partitioned differently for different terminals, and therefore, a global partition as above does not work.

Our main idea is to design states that store additional information, by storing connectivity information about all the possible partitions of our solution. In this way, any terminal can use a different partition of the edges, and we can ensure that all the demands are satisfied. We will show that, even though we may need to partition the solution differently for every terminal, there is a limited number of ways that these partitions can express themselves locally, and therefore the number of states is bounded by a function of the width of the tree decomposition and the maximum demand.

Let (G, c, r, S, k) be an instance of SNDP, and $(\mathcal{T}, X)$ be a tree decomposition of G satisfying the properties of Lemma 2.6. Furthermore, we add r to every bag X_t, $t \in V(\mathcal{T})$. We dedicate the rest of this section to the proof of Theorem 4.3, starting with edge connectivity and edge costs on undirected graphs.

Let t be a node of the tree decomposition of G. The possible states for G_t are of the form $\Psi \subseteq \mathcal{C}_t^{2K}$, where $\mathcal{C}_t$ represents the set of possible connection sets over X_t. In other words, the states for G_t will be a set of tuples of $2K$ connection sets. Each of these tuples of connection sets, $(\Gamma_1, \Gamma_2, \ldots, \Gamma_K, \Delta_1, \Delta_2, \ldots, \Delta_K)$, represents a possible partition of the edges of the solution into K sets $F_1, F_2, \ldots, F_K$, such that for each $i \in [K]$, (Γ_i, Δ_i) represent the connectivity of F_i, in the sense of Definition 3.22. We refer to the set of all partitions of a set A as $\text{part}(A)$.

Observation 4.7. *There are at most* $\exp(w^{O(wK)})$ *possible states for a subinstance* G_t, $t \in \mathcal{T}$, *where* w *is the width of the tree decomposition.*

Proof. The number of possible connection sets is $w^{O(w)}$ (by Lemma 3.21), which implies that the number of possible elements $(\vec{\Gamma}, \vec{\Delta})$ is $w^{O(wK)}$. Since Ψ is a subset of these elements, there are $\exp(w^{O(wK)})$ possible sets Ψ. □

Our goal is now to find a state for each subinstance of the problem, indexed by the nodes of the tree decomposition $\mathcal{T}$, in such a way that these states satisfy consistency constraints derived from Definition 3.22, and such that the cost of the solution is minimized. We can thus define a dynamic program with subproblems of the form $P[t, \Psi]$, for every $t \in \mathcal{T}$ and every possible state Ψ for t as described above. The intuitive meaning of $P[t, \Psi]$ is to

> "Find the minimum-cost subgraph F of G_t such that, for every element $(\vec{\Gamma}, \vec{\Delta}) \in \Psi$, F can be partitioned into $(F_1, \ldots, F_K) \in \text{part}(F)$, where, for every $j \in [K]$, F_j connects all the pairs in Γ_j, and, assuming the existence of paths between any pair in Δ_j, connects all the terminals in G_t to r with demand $d_t \geq j$".

Our goal is to compute $c[t, \Psi]$, the optimal cost for problem $P[t, \Psi]$.

As with the dynamic program in Section 3.3.4, we start by setting initial values for trivial subproblems, and marking some subproblems as invalid, that is, $c[t, \Psi] = +\infty$, according to Definition 3.22.

Procedure 4.8: Initialization of the DP table.

(1) For every leaf node t and every state Ψ, we set $c[t, \Psi] = 0$ if for every $(\vec{\Gamma}, \vec{\Delta}) \in \Psi$, $\Gamma_j = \emptyset$ for $j \in [K]$. Otherwise, we set (t, Ψ) as invalid.

(2) We mark the cells $(\text{root}(\mathcal{T}), \Psi)$ as invalid if there is $(\vec{\Gamma}, \vec{\Delta}) \in \Psi$ such that $\vec{\Gamma} \neq \vec{\Delta}$.

(3) Let s_i be one of the terminals, and $t \in V(\mathcal{T})$ be a node with $s_i \in X_t$. A cell (t, Ψ) is only valid if there is $(\vec{\Gamma}, \vec{\Delta}) \in \Psi$ such that for all $j \in [k_i]$, $(r, s_i) \in \Delta_j$. If this condition is not satisfied, we mark the cell as invalid.

For all the other subproblems, we compute the optimal value by using the values of children subproblems that are consistent with it.

Definition 4.9. We say that states Ψ, Ψ', Ψ'' for node t and its children t', t'' *are consistent via* Y_t, and denote it as

$$\Psi \xleftrightarrow{Y_t} \left(\Psi', \Psi''\right)$$

if for every element $(\vec{\Gamma}, \vec{\Delta}) \in \Psi$, there are elements $(\vec{\Gamma}', \vec{\Delta}') \in \Psi'$, $(\vec{\Gamma}'', \vec{\Delta}'') \in \Psi''$, and a partition $(Y_1, \ldots, Y_K) \in \text{part}(Y_t)$ such that $(\vec{\Gamma}, \vec{\Delta})$ $(\vec{\Gamma}', \vec{\Delta}')$, $(\vec{\Gamma}'', \vec{\Delta}'')$ *are consistent via partition* $(Y_1, \ldots, Y_K)$, that is, for every $j \in [K]$,

$$\begin{aligned}\Gamma_j &= \text{tc}(\Gamma'_j \cup \Gamma''_j \cup Y_j)\big|_t \\ \Delta'_j &= \text{tc}(\Delta_j \cup \Gamma'_j)\big|_{t'} \\ \Delta''_j &= \text{tc}(\Delta_j \cup \Gamma''_j)\big|_{t''}\end{aligned}$$

Additionally, it must also hold that for every $(\vec{\Gamma}', \vec{\Delta}') \in \Psi'$ (resp. $(\vec{\Gamma}'', \vec{\Delta}'') \in \Psi''$), there are elements $(\vec{\Gamma}, \vec{\Delta}) \in \Psi$, $(\vec{\Gamma}'', \vec{\Delta}'') \in \Psi''$ (resp. $(\vec{\Gamma}', \vec{\Delta}') \in \Psi'$) such that the conditions above hold.

We say a set of states $\{\Psi_t\}_{t \in \mathcal{T}}$ is *consistent* via Y if, for each internal node $t \in \mathcal{T}$ with children t', t'', states Ψ_t, $\Psi_{t'}$, $\Psi_{t''}$ are consistent via $Y_t := Y \cap X_t$.

Using this definition of consistency, we can now compute the optimal value for any subproblem, using the following recursion:

$$c[t, \Psi] = \min \left\{ c[t', \Psi'] + c[t'', \Psi''] + c(Y_t) \;\middle|\; \Psi \xleftrightarrow{Y_t} (\Psi', \Psi'') \right\}$$

The optimal solution is obtained from the best state for the root, that is, it is given by,

$$c^* = \min_{\Psi} c[\text{root}(\mathcal{T}), \Psi]$$

All that is left is to prove that this dynamic program captures an optimum solution to the problem, and that the solutions we obtain using the dynamic program are feasible. Specifically, we will prove that we can represent a solution F as a consistent set of states $\{\Psi_t\}_{t \in \mathcal{T}}$, and similarly, if we have a consistent set of states $\{\Psi_t\}_{t \in \mathcal{T}}$, we can obtain a solution to the problem, with the guarantees given by Ψ_t. These two properties together imply that, given an optimum solution F^* and the corresponding states $\{\Psi^*_t\}_{t \in \mathcal{T}}$, the cost of F^* in G_t is at least $c(F^* \cap G_t) \geq c[t, \Psi_t]$ for any $t \in \mathcal{T}$. Therefore, the cost of the solution obtained is at most the optimum cost. The following lemmas formalize these properties.

Lemma 4.10. *For every subgraph $F \subseteq E(G)$ there is a consistent set of states $\{\Psi_t\}_{t \in \mathcal{T}}$, such that*

- *the cost of the dynamic program solution $\{\Psi_t\}_{t \in \mathcal{T}}$ is exactly $\text{c}(F)$, and*
- *for every $\ell \leq K$ pairs of vertices $\{(a_1, b_1), (a_2, b_2), \ldots, (a_\ell, b_\ell)\}$ in the same bag (for some $t \in \mathcal{T}$, $a_1, b_1, \ldots, a_\ell, b_\ell \in X_t$), if there are edge-disjoint paths connecting each pair (a_i, b_i) in F, then there is $(\vec{\Gamma}, \vec{\Delta}) \in \Psi$ such that $(a_i, b_i) \in \Delta_i$ for $i \in [\ell]$.*

Proof. We start by showing how to transform F into states $\{\Psi_t\}_{t \in \mathcal{T}}$. Let $F \subseteq E(G)$, and let $Y_t = F \cap E_t$, for every $t \in \mathcal{T}$. For every node $t \in \mathcal{T}$, the set Ψ_t will contain one element for every possible partition of F into K parts. For each of these partitions, the corresponding elements in the sets Ψ_t will be consistent among all nodes $t \in \mathcal{T}$, which

will prove the consistency properties. Formally, for each partition $P = (P_1, \ldots, P_K) \in \text{part}(F)$, we define, for each node $t \in \mathcal{T}$,

$$\Gamma_j^{(t)}(P) = \text{tc}\left(P_j \cap E(G_t)\right)|_t$$
$$\Delta_j^{(t)}(P) = \text{tc}\left(P_j\right)|_t$$

Using the notation $\vec{\Gamma}^{(t)}(P) = (\Gamma_1^{(t)}(P), \ldots, \Gamma_K^{(t)}(P))$ (similarly for Δ), we define Ψ_t as:

$$\Psi_t = \left\{ \left(\vec{\Gamma}^{(t)}(P), \vec{\Delta}^{(t)}(P)\right) \mid P \in \text{part}(F) \right\}$$

By Lemma 3.23, the connection sets $\Gamma_j^{(t)}(P)$, $\Delta_j^{(t)}(P)$ satisfy the local definition of connectivity for fixed j and P. This immediately implies that all of the cells (t, Ψ_t) are valid, as they satisfy the validity constraints by definition of the elements in Ψ_t. Additionally, consistency is also satisfied, as $\left(\vec{\Gamma}^{(t)}(P), \vec{\Delta}^{(t)}(P)\right)$, $\left(\vec{\Gamma}^{(t')}(P), \vec{\Delta}^{(t')}(P)\right)$, and $\left(\vec{\Gamma}^{(t'')}(P), \vec{\Delta}^{(t'')}(P)\right)$ are consistent via partition $(P_1 \cap Y_t, P_2 \cap Y_t, \ldots, P_K \cap Y_t)$, by definition. The cost of the solution is $\sum_{t \in \mathcal{T}} c(Y_t) = \text{c}(F)$, since $Y_t = F \cap E_t$ and all the sets E_t form a partition of the edges of the graph. This concludes the transformation from F to $\{\Psi_t\}_{t \in \mathcal{T}}$.

Let $(a_1, b_1), (a_2, b_2), \ldots, (a_\ell, b_\ell)$ be pairs of vertices belonging to the bag X_t, $t \in \mathcal{T}$, $\ell \leq K$. We will now prove that if there are edge-disjoint paths connecting the pairs in F, then there is $(\vec{\Gamma}, \vec{\Delta}) \in \Psi$ such that $(a_i, b_i) \in \Delta_i$ for $i \in [\ell]$. Let $q_1, \ldots, q_\ell$ be the edge-disjoint paths connecting the pairs in F, such that q_i connects a_i to b_i. We take a partition of F consistent with the paths q_i, that is, we take $(P_1, \ldots, P_K) \in \text{part}(F)$ such that for all $i \in [\ell]$, $q_i \subseteq P_i$ (such a partition must exist: it is sufficient to start with $P_i = q_i$ and add extra edges to any part). Using $\left(\vec{\Gamma}^{(t)}(P), \vec{\Delta}^{(t)}(P)\right) \in \Psi_t$, we can now finish the proof: by definition, $\Delta_j^{(t)}(P) = \text{tc}\left(P_j\right)|_t$, and $(a_j, b_j) \in \text{tc}\left(P_j\right)|_t$, which concludes the proof. □

Lemma 4.11. *For every consistent set of states $\{\Psi_t\}_{t \in \mathcal{T}}$, there is a subgraph $F \subseteq E(G)$, such that*

(1) the cost of F, $\text{c}(F)$, is the cost of the dynamic program solution $\{\Psi_t\}_{t \in \mathcal{T}}$, and

(2) for every $\ell \leq K$ pairs of vertices $\{(a_1, b_1), (a_2, b_2), \ldots, (a_\ell, b_\ell)\}$ belonging to the same bag $(a_1, b_1, \ldots, a_\ell, b_\ell \in X_t$ for some $t \in \mathcal{T})$, if there is $(\vec{\Gamma}, \vec{\Delta}) \in \Psi$ such that $(a_i, b_i) \in \Delta_i$ for $i \in [\ell]$, then there are edge-disjoint paths connecting each pair (a_i, b_i) in F.

Proof. Let $\{\Psi_t\}_{t \in \mathcal{T}}$ be a set of states for each node of $\mathcal{T}$. For each internal node $t \in \mathcal{T}$, with children $t', t'' \in \mathcal{T}$, there must be a set of edges $Y_t \subseteq E_t$, such that

$$\Psi_t \xleftrightarrow{Y_t} (\Psi_{t'}, \Psi_{t''}).$$

We take $F = \bigcup_{t \in \mathcal{T}} Y_t$, which implies that the $\text{c}(F) = \sum_{t \in \mathcal{T}} c(Y_t)$, which is the cost of the dynamic programming solution dictated by the states $\{\Psi_t\}_{t \in \mathcal{T}}$. Notice that in this case we also have that $Y_t = F \cap E_t$, as $Y_t \subseteq E_t$, and the sets E_t are disjoint. This concludes the transformation from states to solution F.

We now prove Property (2) of the lemma. Let $(a_1, b_1), (a_2, b_2), \ldots, (a_\ell, b_\ell)$ be pairs of vertices belonging to the bag X_t, $t \in \mathcal{T}$, $\ell \leq K$, and let $\psi_t = (\vec{\Gamma}, \vec{\Delta}) \in \Psi_t$ be an element of Ψ_t such that $(a_j, b_j) \in \Delta_j$, for every $j \in [\ell]$. Using the consistency rules, we can now obtain consistent elements for every node in $\mathcal{T}$. This can be achieved using the consistency rules in the following two ways:

(1) if we have an element ψ_t for a node $t \in \mathcal{T}$, consistency dictates that there are elements $\psi_{p(t)} \in \Psi_{p(t)}$ and $\psi_\tau \in \Psi_\tau$ (where τ is the sibling of t), such that $\psi_{p(t)}$ is consistent with (ψ_t, ψ_τ). Thus we obtain elements for nodes $p(t)$ and τ;

(2) if we have an element ψ_t for a node $t \in \mathcal{T}$ with children t', t'', by consistency, there must be elements $\psi_{t'} \in \Psi_{t'}$ and $\psi_{t''} \in \Psi_{t''}$, such that ψ_t is consistent with $(\psi_{t'}, \psi_{t''})$. Thus we obtain elements for nodes t' and t''.

By repeating this process, we can obtain elements ψ_t for every node $t \in \mathcal{T}$, such that consistency rules are satisfied. Furthermore, consistency rules for ψ_t imply that for every node $t \in \mathcal{T}$, there is a partition of $\left(Y_1^{(t)}, \ldots, Y_K^{(t)}\right) \in \text{part}(Y_t)$ that satisfies consistency. Taking the union over all $t \in \mathcal{T}$ of the partition, $F_i = \cap_{t \in \mathcal{T}} Y_i^{(t)}$, we also get a partition $(F_1, \ldots, F_K)$ of F.

Let $\left(\vec{\Gamma}^{(t)}, \vec{\Delta}^{(t)}\right) = \psi_t$, for every node t. By fixing j, and applying Lemma 3.23 to the connection sets $(\Gamma_j^{(t)}, \Delta_j^{(t)})$ for every node $t \in \mathcal{T}$, we get that $(a_j, b_j) \in \Delta_j^{(t)}$ implies that there is a path in F_j connecting a_j and b_j. Since $(F_1, \ldots, F_K)$ is a partition of F, this implies that there are edge-disjoint paths connecting all the pairs in F. □

Vertex Costs, Vertex Connectivity and Directed Graphs

We will now show how to adapt the techniques of this section to solve SNDP with vertex costs, vertex connectivity, or in directed graphs. We refer to the undirected, edge-connectivity, edge-costs variant presented above as the *default* case. Regarding directed graphs, not much change is needed: the only difference is that connectivity sets are no longer symmetric, and therefore the number of possibilities increases to 2^{w^2} (see Lemma 3.21), implying a running time of $n\, 2^{\exp(w^2 K)}$ for SNDP.

For the vertex-cost or vertex-connectivity variants, we introduce the more general framework of Definition 3.27 and Lemma 3.28. Let t be a node of the tree decomposition of G. We now take states for G_t to be of the form (Z, Ψ), where $Z \subseteq V$, $\Psi \subseteq (2^{X_t})^K \times \mathcal{C}_t^{*2K}$, where $\mathcal{C}_t^*$ is the set of possible relations in X_t. Each of the elements of Ψ represents the connectivity of a partition of the solution, that is, each element of Ψ is a tuple $(Z_1, \ldots, Z_K, \Gamma_1, \Gamma_2, \ldots, \Gamma_K, \Delta_1, \Delta_2, \ldots, \Delta_K)$, where each triple $(Z_i, \Gamma_i, \Delta_i)$ has the meaning of Definition 3.27 for a single set of the partition.

Observation 4.12. *The number of possible states (Z, Φ) is $2^{\exp(w^2 K)}$.*

Proof. There are at most 2^w possible subsets of X_t, and 2^{w^2} relations in X_t. Therefore, $(2^{X_t})^K \times \mathcal{C}_t^{*2K}$ has at most $\exp(w^2 K)$ elements, and at most $2^{\exp(w^2 K)}$ subsets. □

A solution is now represented as a pair (F, W), $F \subseteq E$, $W \subseteq V$. We remark that, as in the algorithm for the vertex-cost variant in Section 3.4.1, the algorithm does not directly store the values of $W_t = W \cap (X_t \setminus X_{p(t)})$, but instead only stores the value of Z_t in the state. Setting $W_t = Z_t \setminus X_{p(t)}$, or, given W, setting $Z_t = W \cap X_t$, achieves

the same purpose. Depending on the variant of the problem considered, we may fix F or W, and may consider different ways of partitioning the solution. We will now present the general structure of the dynamic program, and then detail the changes to the edge-connectivity and vertex-connectivity variants. The algorithm is similar for both the edge-connectivity and vertex-connectivity variants, with the only difference being how to "partition" the edges and vertices of the solution. We purposefully postpone the definition of the functions part_E and part_V (representing the set of edge and vertex partitions, respectively).

We start by initializing some of the cells in the dynamic program, and marking some others as invalid, meaning their cost is set to be infinite (and cannot be changed). The conditions presented here depend on the function part_V, and hence differ between variants.

Procedure 4.13: Initialization of the DP table.

(1) For every node t and every state (Z, Ψ), we set cell (t, Z, Ψ) as invalid if for some $(\vec{Z}, \vec{\Gamma}, \vec{\Delta})$, $\vec{Z} \notin \text{part}_V(Z)$.

(2) For every leaf node t and every state (Z, Ψ), we mark (t, Z, Ψ) as invalid if for any $(\vec{Z}, \vec{\Gamma}, \vec{\Delta}) \in \Psi$, and any $j \in [K]$, $\Gamma_j \neq \emptyset$.

(3) We mark cells $(\text{root}(\mathcal{T}), Z, \Psi)$ as invalid if there is $(\vec{Z}, \vec{\Gamma}, \vec{\Delta}) \in \Psi$ such that $\vec{\Gamma} \neq \vec{\Delta}$.

(4) Let s_i be one of the terminals, and $t \in V(\mathcal{T})$ a node with $s_i \in X_t$. A cell (t, Z, Ψ) is only valid if $s_i \in Z$ and there is $(\vec{Z}, \vec{\Gamma}, \vec{\Delta}) \in \Psi$ such that for all $j \in [k_i]$, $(r, s_i) \in \Delta_j$. If this condition is not satisfied, we mark the cell as invalid.

(5) For every leaf node t and every state (Z, Ψ), if (t, Z, Ψ) is not marked invalid, we set $c[t, Z, \Psi] = 0$.

The initialization is mostly similar to the default case, except for the addition of Step (1), and the condition that $s_i \in Z$ in Step (4).

We now present the definition of consistency for both the edge-connectivity and vertex-connectivity variants. The general definition is as follows, using the generic part_E function, which will differ depending on the variant.

Definition 4.14. We say that states Ψ, Ψ', Ψ'' for node t and its children t', t'' *are consistent via* Y_t, and denote it as

$$(Z, \Psi) \xleftrightarrow{Y_t} \big((Z', \Psi'), (Z'', \Psi'')\big)$$

if it holds that

$$Z \cap X_{t'} = Z' \cap X_t \qquad\qquad Z \cap X_{t''} = Z'' \cap X_t$$

and for every element $(\vec{Z}, \vec{\Gamma}, \vec{\Delta}) \in \Psi$, there are elements $(\vec{Z}', \vec{\Gamma}', \vec{\Delta}') \in \Psi'$, $(\vec{Z}'', \vec{\Gamma}'', \vec{\Delta}'') \in \Psi''$, and a partition $(Y_1, \ldots, Y_K) \in \text{part}_E(Y_t)$ such that $(\vec{Z}, \vec{\Gamma}, \vec{\Delta})$, $(\vec{Z}', \vec{\Gamma}', \vec{\Delta}')$, $(\vec{Z}'', \vec{\Gamma}'', \vec{\Delta}'')$ *are consistent via partition* $(Y_1, \ldots, Y_K)$, that is, for every $j \in [K]$,

$$\Gamma_j = \text{tc}^*_{Z_j}(\Gamma'_j \cup \Gamma''_j \cup Y_j)\Big|_t$$

$$\Delta'_j = \text{tc}^*_{Z'_j}(\Delta_j \cup \Gamma'_j)\Big|_{t_1} \qquad\qquad \Delta''_j = \text{tc}^*_{Z''_j}(\Delta_j \cup \Gamma''_j)\Big|_{t_2}$$

$$Z'_j \cap X_t = Z_j \cap X_{t'} \qquad\qquad Z''_j \cap X_t = Z_j \cap X_{t''}$$

Additionally, it must also hold that for every $(\vec{Z}', \vec{\Gamma}', \vec{\Delta}') \in \Psi'$ (resp. $(\vec{Z}'', \vec{\Gamma}'', \vec{\Delta}'') \in \Psi''$), there are elements $(\vec{Z}, \vec{\Gamma}, \vec{\Delta}) \in \Psi$, $(\vec{Z}'', \vec{\Gamma}'', \vec{\Delta}'') \in \Psi''$ (resp. $(\vec{Z}', \vec{\Gamma}', \vec{\Delta}') \in \Psi'$) such that the conditions above hold.

We say a set of states $\{(Z, \Psi_t)\}_{t \in \mathcal{T}}$ is *consistent* via Y if, for each internal node $t \in \mathcal{T}$ with children t', t'', states (Z, Ψ_t), $(Z_{t'}, \Psi_{t'})$, $(Z_{t''}, \Psi_{t''})$ are consistent via $Y_t := Y \cap X_t$.

Now, given a definition of consistency for the states, we can compute the optimal value for any subproblem, using the following recursion:

$$c[t, Z, \Psi] = \min\Big\{ c[t', Z', \Psi'] + c[t'', Z'', \Psi''] + c(Y_t) + c(Z' \setminus Z) + c(Z'' \setminus Z) \\ \Big| \ (Z, \Psi) \xleftrightarrow{Y_t} \big((Z', \Psi'), (Z'', \Psi'')\big) \Big\}$$

The optimal solution is then obtained from the best state for the root, that is, it is given by,

$$c^* = \min_{Z, \Psi} c[\mathrm{root}(\mathcal{T}), Z, \Psi] + c(Z)$$

We are now ready to specify the final details:

- **Edge-connectivity:** if edge-disjoint paths are desired, set $\mathrm{part}_E(A) = \mathrm{part}(A)$ to be the set of all partitions of $A \subseteq E$; otherwise, set $\mathrm{part}_E(A) = \{(A, A, \ldots, A)\}$, that is, $\mathrm{part}_E(A)$ simply copies A to every index;
- **Vertex-connectivity:** if vertex-disjoint paths are desired, set $\mathrm{part}_V(B) = \mathrm{part}(B)$ to be the set of all partitions of $B \subseteq V$; otherwise, set $\mathrm{part}_V(B) = \{(B, B, \ldots, B)\}$, that is, $\mathrm{part}_V(B)$ copies B to every index;
- **Edge-costs:** set $c(Y)$, $Y \subseteq E$ as appropriate; otherwise, set $c(Y) = 0$, $Y \subseteq E$, which implies that we can fix $F = E$ and $Y_t = E_t$ (since edges are free);
- **Vertex-costs:** set $c(W)$, $W \subseteq E$ as appropriate; otherwise, set $c(W) = 0$, $W \subseteq V$, which implies that we can fix $W = V$ and $Z_t = X_t$ (since vertices are free).

Any combination of costs and connectivity may be taken, including edge costs and vertex costs simultaneously.

To finish the analysis of the result, we need to prove the analogues of Lemmas 4.10 and 4.11. We omit the full proofs of the lemmas, as they are mostly similar to the original, but highlight the differences required to make the proofs follow.

We transform a solution specified by $F \subseteq E$, $W \subseteq V$ into states $\{(Z_t, \Psi_t)\}_{t \in \mathcal{T}}$ as follows: $Z_t = W \cap X_t$; for each partition $P = (P_E, P_V)$, with $P_E = (F_1, \ldots, F_K) \in \mathrm{part}_E(F)$, $P_V = (W_1, \ldots, W_K) \in \mathrm{part}_V(F)$, we define, for each node $t \in \mathcal{T}$,

$$Z_j^{(t)}(P) = W_j \cap X_t$$
$$\Gamma_j^{(t)}(P) = \mathrm{tc}^*_{W_j}(F_j \cap E(G_t))\Big|_t$$
$$\Delta_j^{(t)}(P) = \mathrm{tc}^*_{W_j}(F_j)\Big|_t$$

Using the notation $\vec{Z}^{(t)}(P) = \big(Z_1^{(t)}(P), \ldots, Z_K^{(t)}(P)\big)$ (similarly for $\vec{\Gamma}^{(t)}(P)$, $\vec{\Delta}^{(t)}(P)$), we define Ψ_t as:

$$\Psi_t = \Big\{ \big(\vec{Z}^{(t)}(P), \vec{\Gamma}^{(t)}(P), \vec{\Delta}^{(t)}(P)\big) \mid P \in \mathrm{part}_E(F) \times \mathrm{part}_V(W) \Big\}$$

Conversely, given states $\{(Z_t, \Psi_t)\}_{t\in\mathcal{T}}$, we can set $W = \bigcup_{t\in\mathcal{T}} Z_t$, and set F as in Lemma 4.11. The rest of the analysis follows similarly to the default case.

4.2.2 Solving GroupSNDP

Our strategy to solve GroupSNDP is similar to that of Section 3.4.1: we are going to define an instance of STGST that represents our instance of the GroupSNDP problem. Remarkably, we embed both GST and GroupSNDP to the same problem, even though the fault-tolerance in the latter makes the problem much harder. The reason for this is that, in both cases, we can solve the problem by a dynamic program with extra group constraints, with these group constraints determining the problem (in this case STGST) that we embed to.

Since we know how to obtain an $O(\log n \log h)$-approximation to the STGST problem in size polynomial on the size of the instance, it is sufficient to prove an analogue of Theorem 3.33 for GroupSNDP.

Theorem 4.15. *Let $\mathcal{I} = \big(G, c, r, \mathcal{S}, k\big)$ be an instance of GroupSNDP, where G has treewidth w and let $K = \max_i k_i$.*

There is an STGST instance $\mathcal{I}' = \big(\widetilde{\mathcal{T}}, \tilde{c}, \tilde{r}, \{\widetilde{S}_i\}_{i\in[h]}, \widetilde{\mathcal{T}}_c, \widetilde{\mathcal{T}}_p\big)$, where $\widetilde{\mathcal{T}}$ is a tree, such that:

(1) $|V(\widetilde{\mathcal{T}})| = O(n^{w^{O(wK)}})$

(2) for every tree $F \subseteq E(G)$ there is a solution tree X (and vice-versa), such that $\mathrm{c}(F) = c(X)$ and, for every $i \in [h]$, F contains k_i edge-disjoint paths from r to some $v_i \in S_i$ iff X connects $\mathrm{root}(\widetilde{\mathcal{T}})$ to $\widetilde{S}_i$.

Furthermore, we can compute $\mathcal{I}'$ given $\mathcal{I}$, as well as F (resp. X) given X (resp. F), in time $O\big(n^{O(w \log w)}\big)$.

Proof. The proof of this theorem follows analogously to that of Theorem 3.33. We will mostly highlight the differences between these two proofs, and omit most details to avoid repetition.

As with the proof of Theorem 3.33, we start by defining a DAG $\widetilde{H}$, which mimics the dynamic program in Section 4.2.1. It consists of:

(1) Nodes $\tilde{t}[t, \Psi]$ for every $t \in \widetilde{\mathcal{T}}$, and every state Ψ for t;

(2) Combination nodes $\tilde{t}^c[\tilde{t}, \tilde{t}_1, \tilde{t}_2, Y_t]$ for every $\tilde{t} = \tilde{t}[t, \Psi]$, $\tilde{t}_i = \tilde{t}[t_i, \Psi_i]$, where t_1, t_2 are the children of t, and $Y_t \subseteq E_t$;

(3) Arcs $(\tilde{t}, \tilde{t}^c)$, $(\tilde{t}^c, \tilde{t}_1)$, $(\tilde{t}^c, \tilde{t}_2)$, for $\tilde{t}^c[\tilde{t}, \tilde{t}_1, \tilde{t}_2, Y_t] \in \widetilde{H}$ and

$$\Psi \xleftrightarrow{Y_t} (\Psi_1, \Psi_2);$$

(4) Root node $\tilde{r}$ and arcs $(\tilde{r}, \tilde{t})$, for all $\tilde{t} = \tilde{t}[\mathrm{root}(\mathcal{T}), \Psi]$, where for every $(\vec{\Gamma}, \vec{\Delta}) \in \Psi$, $\vec{\Gamma} = \vec{\Delta}$.

Pruning is performed similarly to Procedure 3.34, with a slight modification: a leaf node $\tilde{t} = \tilde{t}[t, \Psi]$ is removed if there is $(\vec{\Gamma}, \vec{\Delta}) \in \Psi$ such that $\Gamma \neq \emptyset^K$.

The groups $\widetilde{S}'_i$ are defined as follows: a node $\tilde{t} = \tilde{t}[t, \Psi]$ is in $\widetilde{S}'_i$ if there exists $v_i \in S_i \cap X_t$ and an element $(\vec{\Gamma}, \vec{\Delta}) \in \Psi$ such that, for every $j \in [k_i]$, $(r, v_i) \in \Delta_j$. Formally,

$$\widetilde{S}'_i = \left\{ \tilde{t}[t, \Psi] \in \widetilde{H} : (r, v_i) \in \bigcap_{j=1}^{k_i} \Delta_j, (\vec{\Gamma}, \vec{\Delta}) \in \Psi, v_i \in S_i \right\} \qquad \forall i \in [h]$$

The remainder of the proof follows similarly to Theorem 4.15, and by using Lemmas 4.10 and 4.11. Using similar arguments, we can prove that the size of the instance is $O\big(n^{w^{O(wK)}}\big)$. □

Vertex Costs, Vertex Connectivity and Directed Graphs

We now show how to adapt the results of this section to other variants of GroupSNDP, as we did in Section 4.2.1. We will show the necessary changes to obtain an analogue of Theorem 4.15 for any variant of costs, connectivity and directed or undirected graphs, with an increased size of $|V(\widetilde{\mathcal{T}})| = n^{\exp\left(w^2 K\right)}$.

The nodes and arcs of the graph are defined according to the new states (of the form (Z, Φ)), and the notion of consistency for these states. The resulting graph has nodes $\tilde{t}[t, Z, \Psi]$, $\tilde{t}^c[\tilde{t}, \tilde{t}_1, \tilde{t}_2, Y_t]$, and arcs as in the default case. For a combination node $\tilde{t}^c = \tilde{t}^c[\tilde{t}, \tilde{t}_1, \tilde{t}_2, Y_t]$, the arc $(\tilde{t}, \tilde{t}^c)$ has cost $c(Y_t)$; for every $\tilde{t} = \tilde{t}[t, Z, \Psi]$, and every parent $\tilde{t}' = \tilde{t}[t', Z', \Psi']$ of $\tilde{t}$, the arc $(\tilde{t}', \tilde{t})$ has cost $c(Z \setminus Z')$ (or simply $c(Z)$ if the parent of $\tilde{t}$ is the root node $\tilde{r}$). For a definition of $c(Y_t)$ and $c(Z \setminus Z')$ (resp. $c(Z)$) see Section 4.2.1.

Pruning is performed similarly to Procedure 3.34, using the steps of Procedure 4.13 except for Step (4). In other words, we remove nodes corresponding to invalid cells marked in (1)–(3), (5). Then, we remove nodes not connected from the root; nodes $\tilde{t} = \tilde{t}[t, Z, \Psi]$ where t is not a leaf but $\tilde{t}$ is; and combination nodes for which at least one of the children has been removed.

The groups $\widetilde{S}'_i$ are defined as follows: a node $\tilde{t} = \tilde{t}[t, Z, \Psi]$ is in $\widetilde{S}'_i$ if there exists $v_i \in S_i \cap Z$ and an element $(\vec{Z}, \vec{\Gamma}, \vec{\Delta}) \in \Psi$ such that, for every $j \in [k_i]$, $(r, v_i) \in \Delta_j$. Formally,

$$\widetilde{S}'_i = \left\{ \tilde{t}[t, Z, \Psi] \in \widetilde{H} : (r, v_i) \in \bigcap_{j=1}^{k_i} \Delta_j, (\vec{Z}, \vec{\Gamma}, \vec{\Delta}) \in \Psi, v_i \in S_i \cap Z \right\} \qquad \forall i \in [h]$$

The remainder of the proof follows similarly to the default case. Due to the increased number of states (see Observation 4.12), the size of the instance is $n^{\exp\left(w^2 K\right)}$.

4.3 Connectivity-K Mimicking Networks

In this section, we present a different method to represent connectivity within a bag, which improves our results. The main idea is to represent connectivity as a graph, which we call *connectivity-K mimicking network*, in such a way that routing paths in the connectivity-K mimicking network is almost equivalent to routing them in the original graph. We can then use mimicking networks as the states in our dynamic program, and using similar analysis, get a faster algorithm for GroupSNDP.

The biggest differences between these two approaches are the running time and the generality of the approach: the running time of the algorithm of Section 4.2 is doubly-exponential on the maximum demand and the treewidth, while the approach we present in this section is only single-exponential in the treewidth; however, we formulate connectivity-K mimicking networks with regard to edge cuts, and therefore can only apply this approach for edge-connectivity, edge-cost variants of SNDP and GroupSNDP.

The formal definition of connectivity-K mimicking network is as follows.

Definition 4.16. Let G be a graph with edge capacities $c \geq 1$, and let $X \subseteq V(G)$ be a subset of the vertices of G called *terminals*.

We say that H is a *connectivity-K mimicking network* for (G, X) if the following holds: for any disjoint subsets X_A, X_B of X, we have

$$\text{mincut}_H^K(X_A, X_B) = \text{mincut}_G^K(X_A, X_B)$$

where $\text{mincut}_G^K(X, Y) = \min\big(\text{mincut}_G(X, Y), K\big)$ is the minimum between K and the value of the minimum cut in G separating X and Y.

In our case, we are interested in querying how many edge-disjoint paths there are between two sets of vertices. Therefore, it is enough to consider the unweighted setting of this definition, that is, the case of $c = \mathbf{1}$, where all the capacities are 1.

In that case, we can always find a connectivity-K mimicking network whose size depends exponentially on the maximum connectivity and linearly on the number of terminals. This implies that our states have size $\exp(K)w$, which is a big improvement on the results of Section 4.2, where the states have size $w^{O(wK)}$.

We start by showing that small connectivity-K mimicking networks exist, in Section 4.3.1, then show how to compute one, in Section 4.3.2, and finally show how to use connectivity-K mimicking networks to obtain improved results for GroupSNDP, in Section 4.3.3.

4.3.1 Existence of Small Connectivity-K Mimicking Networks

In this section, we will show the existence of connectivity-K mimicking networks of size $3^K K|X|$, where X is the set of terminals. We start by introducing the notion of connectivity-K-linked graphs, and then show how to use these graphs to obtain connectivity-K mimicking networks.

Definition 4.17. Let G be a graph and $X \subseteq G$ be a set of degree-1 terminals.

We say G is *connectivity-K-linked* (with respect to X) if for every cut (A, B),

$$|E(A, B)| \geq \min(|X \cap A|, |X \cap B|, K)$$

A *violating cut* is a cut (A, B) that does not satisfy the inequality above.

Let $H \subseteq G \setminus X$ be a subgraph of G. We say H is *connectivity-K-linked with respect to its neighbors* if H is connectivity-K-linked with respect to the set of terminals $\partial(H)$, containing one terminal for each edge outgoing from H, that is,

$$\partial(H) = \{t_e \mid uv = e \in E(G), u \in H, v \notin H\}$$

The following theorem formalizes the result.

Theorem 4.18. *Let G be a graph, $X \subseteq V(G)$ be a set of terminals, and $K \in \mathbb{Z}_{\geq 1}$.*

There is a connectivity-K mimicking network of size $3^K K|X|$ for G with unit edge capacities $c = \mathbf{1}$ and terminal set X.

The proof of theorem follows from the two lemmas below. We postpone their proof in order to show how they imply Theorem 4.18.

Lemma 4.19. *Let G be a graph with degree-1 terminals X, and let H be a connected subgraph not containing any terminal in X.*

If H is connectivity-K-linked with respect to its neighbors, G/H is a connectivity-K mimicking network for G, where G/H denotes the graph obtained from G by contracting every edge in H.

Lemma 4.20. *Any graph G with a set of degree-1 terminals X and containing no connectivity-K-linked subgraph has size at most $3^K|X|$.*

Proof of Theorem 4.18. In order to simplify the presentation, Lemmas 4.19 and 4.20 assume that all the terminals have degree exactly 1. We can make G satisfy this assumption by adding, for each terminal $v \in X$, dummy vertices $v_1, \ldots, v_K$ that have a single incident edge connecting them to v. The new set of terminals X' is the set of all dummy vertices for all of the terminals.

To use the set of terminals X' instead of X when computing a cut, we simply consider all of the dummy vertices for each terminal of interest. In other words, if we want to compute the min-cut separating $A, B \subseteq X'$, we instead consider the sets $A', B' \subseteq X'$ containing the dummy vertices for each of the terminals in A or B, respectively. Since we are only interested in cuts of capacity at most K, K dummy vertices are sufficient.

We obtain our connectivity-K mimicking network by repeatedly contracting any connectivity-K-linked sets, until no more are left. Repeated application of Lemma 4.19 proves that such a graph is a connectivity-K mimicking network, and Lemma 4.20 establishes the size of the resulting graph as being at most $3^K|X'| \leq 3^K K|X|$. This concludes the proof of the theorem. □

We now prove Lemmas 4.19 and 4.20.

Proof of Lemma 4.19. Let $G' = G/H$ be obtained from G by contracting a connectivity-K-linked set H. We will show that $\text{mincut}^K_{G'}(X_A, X_B) = \text{mincut}^K_G(X_A, X_B)$.

Starting with $\text{mincut}^K_{G'}(X_A, X_B) \geq \text{mincut}^K_G(X_A, X_B)$, we can see that all the edges in G' are also in G, which implies that any cutset in G' is also in G. We conclude that the size of the minimum cut in G must be at most the size of the minimum cut in G', for any pair of terminals sets. In general, we can say that contraction of edges only ever increases connectivity, which implies the above.

We remark that, since the capacities are uniform, it is sufficient to consider sets X_A, X_B, $|X_A|, |X_B| \leq K$. Indeed, if $\text{mincut}^K(X_A, X_B) = \ell \leq K$ (in G or G'), then there are ℓ edge-disjoint paths connecting ℓ terminals from X_A to ℓ terminals from X_B. We can choose $X'_A \subseteq X_A$, $X'_B \subseteq X_B$, as the endpoints of these ℓ disjoint paths. Additionally, if $\ell < K$, we add an additional vertex, if possible, of X_A (resp. X_B) to X'_A (resp. X'_B); this

will make sure that connectivity-K-linkedness is preserved. In this way, we have that $|X'_A|, |X'_B| \leq K$, and $\text{mincut}^K(X_A, X_B) = \ell = \text{mincut}^K(X'_A, X'_B)$.

Let us now show that $\text{mincut}^K_{G'}(X_A, X_B) \leq \text{mincut}^K_G(X_A, X_B)$. Let X_A, X_B be any disjoint subsets of X and let $\ell = \text{mincut}_{G'}(X_A, X_B)$. Then there are ℓ edge-disjoint paths between the terminals in X_A and X_B in G'. Of these, $\ell' \leq \ell$ paths go through the vertex corresponding to H in G' (w.l.o.g. the paths are simple, otherwise we can replace them by shortest paths that only go through H once).

Now, we consider the terminals of $\partial(H)$ corresponding to the ingoing and outgoing edges for each of these ℓ' paths, that is, $Y_A \subseteq \partial(H)$ corresponding to the incoming edges of the paths into H, and $Y_B \subseteq \partial(H)$ corresponding to the outgoing edges. Since H is connectivity-K-linked, we can route paths between these terminal sets inside H if $\ell \leq K$, and only K of the paths if $\ell > K$. We can then augment $\min(\ell', K)$ of the paths in G' into paths in G by routing inside H. In any case, we can convert the ℓ paths in G' into $(\ell - \ell')$ paths in G not intersecting H plus $\min(\ell', K)$ paths routed through H, and therefore,

$$\begin{aligned}\text{mincut}^K_G(X_A, X_B) &\geq (\ell - \ell') + \min(\ell', K) \\ &\geq \min(K, \ell) \\ &= \text{mincut}^K_{G'}(X_A, X_B).\end{aligned}$$ □

Proof of Lemma 4.20. Let $N_K(w)$ be the maximum number of Steiner (non-terminal) nodes in such a graph G, with $w = |X|$. We will prove by double induction that

$$\begin{aligned} N_K(w) &\leq 3^{K-2}(w - 2(K-1)) && \text{if } w > 2(K-1) \\ N_K(w) &\leq 3^{K-2} && \text{if } w \leq 2(K-1) \end{aligned}$$

This proof can be divided into three parts:

(1) $N_2(w) \leq w - 2$, for $w \geq 3$

(2) $N_K(2K' + 1) = N_{K'}(2K' + 1)$, for all $K' < K$

(3) $N_K(w) = N_K(w_1 + \ell) + N_K(w_2 + \ell)$, where $w_1 + w_2 = w$ and ℓ is the number of edges in a violating cut that divides G into two subgraphs of size w_1, w_2.

Starting with Point (2), it is not possible to divide $2K' + 1$ into two sets such that both are at least of size $K' + 1$. This implies that

$$\min(|X \cap A|, |X \cap B|, K') = \min(|X \cap A|, |X \cap B|, K)$$

since either $|X \cap A| \leq K'$ or $|X \cap B| \leq K'$. Therefore, we can increase the value of K' in the definition of connectivity-K' linked to any other value $K > K'$ without affecting the statement.

The proof of Point (1) follows from these two simple facts and Proposition 2.2: (i) for $K = 2$, G is a forest; (ii) G does not contain any vertex of degree 2, unless both its neighbors are terminals. Fact (i) follows because any cycle is always connectivity-2-linked, so any graph that does not contain a connectivity-2-linked subgraph does not contain any cycles, and thus must be a forest. To prove Fact (ii), note that any such vertex v has one neighbor that is not a terminal. The incident edge connecting v to

its non-terminal neighbor is a connectivity-K-linked graph, for any K, so it can be contracted.

As to Point (3), if there is no violating cut, the graph is a connectivity-K-linked set and can be contracted (leading to 1 steiner node). If there is a violating cut, then we can cut each of the ℓ edges into two, that is, replace each edge by two terminals of degree 1, each connected to an endpoint of the edge. Then, we can inductively obtain the size of the instance by summing the sizes of the two parts.

There are now two possibilities (w.l.o.g. $w_1 \leq w_2$). If $w_1 + \ell \leq 2K - 1$, since $w_1 > \ell$,

$$\begin{aligned} N_K(w) &= N_K(w_1 + \ell) + N_K(w_2 + \ell) \\ &\leq N_K(2K-1) + N_K(w - w_1 + \ell) \\ &\leq N_{K-1}(2K-1) + N_K(w-1) \\ &\leq 3^{K-3}\big(2K - 1 - 2(K-2)\big) + 3^{K-2}\big(w - 1 - 2(K-1)\big) \\ &\leq 3^{K-3}3 + 3^{K-2}\big(w - 1 - 2(K-1)\big) \\ &\leq 3^{K-2}\big(w - 2(K-1)\big) \end{aligned}$$

Otherwise,

$$\begin{aligned} N_K(w) &= N_K(w_1 + \ell) + N_K(w_2 + \ell) \\ &\leq 3^{K-2}\big(w_1 + \ell - 2(K-1)\big) + 3^{K-2}\big(w_2 + \ell - 2(K-1)\big) \\ &\leq 3^{K-2}\big(w_1 + w_2 - 2(K-1) + 2\ell - 2(K-1)\big) \\ &\leq 3^{K-2}\big(w - 2(K-1)\big) \end{aligned}$$

In any case, by double induction on K and w, we conclude that the G has size at most $3^K|X|$. □

4.3.2 Computing Small Connectivity-K Mimicking Networks

We now present an algorithm that finds a good connectivity-K mimicking network for a graph G, and then prove its correctness (Theorem 4.21). The idea of the algorithm is to use the above lemma to compute a connectivity-K mimicking network. It starts by computing a violating cut (A, B), satisfying

$$|E(A,B)| < \min(|X \cap A|, |X \cap B|, K)$$

If such a cut does not exist, the graph is connectivity-K-linked, and therefore we can contract it onto a single vertex attached to all of the terminals. Otherwise, we remove the edges $E(A, B)$ of the cut, and for each edge, attach a degree-one terminal to each endpoint. Then, we recurse, on both sides of the graph, and finally join the solutions by connecting by an edge the neighbors of matching terminals created in the previous step. In other words, in the previous step we removed each edge $uv \in E(A, B)$ and added terminals t_{uv}, t_{vu}. Now, we undo this operation, by removing t_{uv}, t_{vu} and connecting their neighbors with an edge. The algorithm is formally presented in Algorithm A.5.

Theorem 4.21. *Given a graph G with degree-1 terminals X, the algorithm described above (see Algorithm A.5) computes a connectivity-K mimicking network H of G with the same terminals, and of size at most $O(3^K|X|)$. Furthermore, H is a minor of G.*

Proof. It is clear that the algorithm runs in time $|X|^K \operatorname{poly}(n)$. Notice that, to compute a violating cut, we simply have to compute the min-cuts for every subsets of terminals $X \cap A$, $X \cap B$, which takes time $\operatorname{poly}(n)$ for each pair of terminal sets. Every time we recurse, we remove one of the edges of the original graph from consideration, and so we must terminate after at most $|E(G)|$ calls.

We will now prove that H is a minor of G. There are essentially two different operations performed by the algorithm:

- Splitting edges: when recursing, edges are split, but are afterwards re-joined. Therefore, this operation does not really change the graph.

- Returning a star when no violating cut exists: This can be thought of as contracting all the edges in G not incident to a terminal.

In both of these cases, the operations can be thought of as contractions of edges until the right result is achieved. Therefore, this does not violate the invariant that H is a minor of G. Since we only contract subgraphs that are connectivity-K-linked, the resulting graph must be a connectivity-K mimicking network of G by Lemma 4.19.

By careful consideration of Lemma 4.20, we can see that it actually applies even if the graph contains connectivity-K-linked sets. Indeed, the proof applies as long as we can either find a violating cut and recurse on both parts, or otherwise the graph is itself connectivity-K-linked. These are precisely the operations made by the algorithm, and thus the bounds of Lemma 4.20 apply to this algorithm. □

4.3.3 Using Connectivity-K Mimicking Networks to solve GroupSNDP

In this section, we prove Theorem 4.5. The proof follows the general structure of the proof presented in Section 4.2, but we use connectivity-K mimicking networks as the states. Our goal is to assign two connectivity-K mimicking networks to each subinstance, roughly corresponding to the connectivity in $E(G_t)$ and $E \setminus E(G_t)$.

We remark that this notation deviates from that in Section 3.3.3, in which we are interested in the connectivity in $E(G_t)$ and E (corresponding to Γ and Δ respectively). The reason for this change is that, in higher connectivity, we want to avoid counting connections multiple times, if only one connection exists. For example, in the rule $\Delta_t = \operatorname{tc}\left(\Delta_{p(t)} \cup \Gamma_t\right)\big|_t$ in Definition 3.22, the same pair (u, v) can be both in $\Delta_{p(t)}$ and Γ_t, and we cannot determine whether they correspond to the same path or disjoint paths. Therefore, we consider the connectivity in disjoint sets of edges, and indeed in almost-disjoint graphs, that only intersect in X_t.

Two ingredients are needed to adjust this technique to our result: (i) consistency rules for the DP, and (ii) an oracle to determine how many edge-disjoint paths there are between two vertices u and v in the same bag.

The following equivalents of Definition 3.22 and Lemma 3.23 for connectivity-K mimicking networks provide these constructions. For convenience, we discard the use of Y in Definition 3.22, using instead all of the edges of the graph. We can then apply the lemma to the graph $(V(G), Y)$, where Y is any subset of edges.

Definition 4.22 (Local Connectivity)**.**

We say that the pairs of connectivity-K mimicking networks $\{(\mathcal{H}'_t, \mathcal{H}_t)\}_{t \in V(\mathcal{T})}$ satisfy the *local connectivity definition* if, for every node $t \in V(\mathcal{T})$ (having left and right children as t_1 and t_2, respectively),

$$\begin{aligned}
\mathcal{H}'_t &:= \begin{cases} (X_t, \emptyset) & \text{if } t \text{ is a leaf of } \mathcal{T} \\ \text{mimnet}_K\left(E_t \cup \mathcal{H}'_{t_1} \cup \mathcal{H}'_{t_2}, X_t\right) & \text{otherwise} \end{cases} \\
\mathcal{H}_{t_1} &:= \text{mimnet}_K\left(\mathcal{H}'_{t_2} \cup \mathcal{H}_t \cup E_t, X_{t_1}\right) \\
\mathcal{H}_{t_2} &:= \text{mimnet}_K\left(\mathcal{H}'_{t_1} \cup \mathcal{H}_t \cup E_t, X_{t_2}\right) \\
\mathcal{H}_{\text{root}(\mathcal{T})} &:= (X_{\text{root}(\mathcal{T})}, \emptyset)
\end{aligned}$$

Lemma 4.23. *Let $G = (V, E)$ be a graph, and $(\mathcal{T}, X)$ its tree decomposition satisfying Properties (3) and (4) from Lemma 2.6. For every $t \in V(\mathcal{T})$, let $(\mathcal{H}'_t, \mathcal{H}_t)$ be a pair as in Definition 4.22.*

Then, the triples $(\mathcal{H}'_t, \mathcal{H}_t)$ satisfy the local definitions iff for every $t \in V(\mathcal{T})$ and every pair of disjoint sets $S_1, S_2 \subseteq X_t$,

$$\begin{aligned}
\text{mincut}^K_{\mathcal{H}'_t}(S_1, S_2) &= \text{mincut}^K_{E(G_t)}(S_1, S_2) \\
\text{mincut}^K_{\mathcal{H}_t}(S_1, S_2) &= \text{mincut}^K_{E \setminus E(G_t)}(S_1, S_2).
\end{aligned}$$

Proof. We start by proving the equality for $\mathcal{H}'$ by bottom-up induction, and then the one for $\mathcal{H}$ by top-down induction.

Let $t \in \mathcal{T}$ be a node of the tree decomposition. Then $E(G_t) = \emptyset = \mathcal{H}'_t$, so equality immediately follows. Consider now an internal node t with children t_1, t_2, and assume that the claim follows for t_1, t_2.

Let $S_1, S_2 \subseteq X_t$, and F be the cutset for a mincut between S_1 and S_2 in $E(G_t)$. We will use $c_G(S_1, S_2) = \text{mincut}^K_G(S_1, S_2)$ for conciseness (in this proof only). Then

$$\begin{aligned}
&c_{G_t}(S_1, S_2) \\
&\quad = \min(K, |F|) \\
&\quad = \min(K, |F \cap E_t| + |F \cap E(G_{t_1})| + |F \cap E(G_{t_2})|) \\
&\quad \geq \min\left(k, c_{E_t}(S_1, S_2) + c_{E(G_{t_1})}(S_1 \cap X_{t_1}, S_2 \cap X_{t_1}\right) + c_{E(G_{t_2})}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})) \\
&\quad = \min\left(k, c_{E_t}(S_1, S_2) + c_{\mathcal{H}'_{t_1}}(S_1 \cap X_{t_1}, S_2 \cap X_{t_1}\right) + c_{\mathcal{H}'_{t_2}}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})) \\
&\quad \geq c_{\mathcal{H}'_t}(S_1, S_2)
\end{aligned}$$

The third inequality follows because each of the three terms corresponds to a min-cut between S_1 and S_2 for the respective edge sets. The fourth inequality follows by induction hypothesis, and the final one follows by definition of $\mathcal{H}'_t$. For this last step, we crucially use that $X_{t_1} \cap X_{t_2} \subseteq X_t$, which means that any cut for E_t, $\mathcal{H}'_{t_1}$ and $\mathcal{H}'_{t_2}$ uses disjoint edges and disjoint vertices outside of X_t. Though these edges may not necessarily exist in $\mathcal{H}'_t$, they provide an upper bound for the cut $c_{\mathcal{H}'_t}$, because $\mathcal{H}'_t = \text{mimnet}_K(E_t \cup \mathcal{H}'_{t_1} \cup \mathcal{H}'_{t_2}, X_t)$.

Analogously, we can prove that $c_{E(G_t)} \leq c_{\mathcal{H}'_t}$, by taking a set of edges F' of $\mathcal{H}'_t$ that realizes the minimum cut in that graph. The same steps then apply to prove the desired

inequality.

$$\begin{aligned}
c_{\mathcal{H}'_t}(S_1, S_2) &= \min(K, |F'|) \\
&= \min(K, |F' \cap E_t| + |F' \cap E(\mathcal{H}'_{t_1})| + |F' \cap E(\mathcal{H}'_{t_2})|) \\
&\geq \min\left(k, c_{E_t}(S_1, S_2) + c_{\mathcal{H}'_{t_1}}(S_1 \cap X_{t_1}, S_2 \cap X_{t_1}\right) + c_{\mathcal{H}'_{t_2}}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})) \\
&= \min\left(k, c_{E_t}(S_1, S_2) + c_{G_{t_1}}(S_1 \cap X_{t_1}, S_2 \cap X_{t_1}\right) + c_{G_{t_2}}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})) \\
&\geq c_{G_t}(S_1, S_2)
\end{aligned}$$

This concludes the first part of the proof.

We now prove that $\text{mincut}^K_{E \setminus E(G_t)}(S_1, S_2) = \text{mincut}^K_{\mathcal{H}_t}(S_1, S_2)$. For $t = r$, notice that $E \setminus E(G_t) = \emptyset = \mathcal{H}_t$, so the equality follows in this case. We now prove the equality for a node t_1 with parent t and sibling t_2.

Let $S_1, S_2 \subseteq X_{t_1}$, and F be the cutset for a mincut between S_1 and S_2 in $E \setminus E(G_{t_1})$. Then

$$\begin{aligned}
c_{E \setminus E(G_{t_1})}(S_1, S_2) &= \min(K, |F|) \\
&= \min(K, |F \cap E_t| + |F \cap (E \setminus E(G_t))| + |F \cap E(G_{t_2})|) \\
&\geq \min(K, c_{E_t}(S_1 \cap X_t, S_2 \cap X_t) + c_{E \setminus E(G_t)}(S_1 \cap X_t, S_2 \cap X_t) \\
&\qquad + c_{E(G_{t_2})}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})) \\
&= \min(K, c_{E_t}(S_1 \cap X_t, S_2 \cap X_t) + c_{\mathcal{H}_t}(S_1 \cap X_t, S_2 \cap X_t) \\
&\qquad + c_{\mathcal{H}'_{t_2}}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})) \\
&\geq c_{\mathcal{H}_{t_1}}(S_1, S_2)
\end{aligned}$$

Similarly to the proof above, we use the fact that $F \cap E_t$, $F \cap (E \setminus E(G_t))$, $F \cap E(G_{t_2})$ are cuts in the subgraphs E_t, $G \setminus E(G_t)$, $E(G_{t_2})$ respectively. The last step follows from the fact that the three terms correspond to cuts in E_t, $\mathcal{H}_t$ and $\mathcal{H}'_{t_2}$, and therefore their union forms a cut in $\mathcal{H}_t \cup E_t \cup \mathcal{H}'_{t_2}$. Since $\mathcal{H}_{t_1} = \text{mimnet}_K(\mathcal{H}_t \cup E_t \cup \mathcal{H}'_{t_2}, X_{t_1})$, the inequality follows. The converse follows similarly:

$$\begin{aligned}
c_{\mathcal{H}_{t_1}}(S_1, S_2) &= \min(K, |F|) \\
&= \min(K, |F \cap E_t| + |F \cap E(\mathcal{H}_t)| + |F \cap E(\mathcal{H}'_{t_2})|) \\
&\geq \min(K, c_{E_t}(S_1 \cap X_t, S_2 \cap X_t) + c_{\mathcal{H}_t}(S_1 \cap X_t, S_2 \cap X_t) \\
&\qquad + c_{\mathcal{H}'_{t_2}}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})) \\
&= \min(K, c_{E_t}(S_1 \cap X_t, S_2 \cap X_t) + c_{E \setminus E(G_t)}(S_1 \cap X_t, S_2 \cap X_t) \\
&\qquad + c_{E(G_{t_2})}(S_1 \cap X_{t_2}, S_2 \cap X_{t_2})) \\
&\geq c_{E \setminus E(G_{t_1})}(S_1, S_2)
\end{aligned}$$

This completes the proof. □

All that is left is to provide an oracle to decide if the solution has k_i edge-disjoint paths from the root r to a vertex v_i. Let t be a node such that $r, v_i \in X_t$. The number of edge-disjoint paths in a solution Y between r and v_i (up to k) is given by $\text{mincut}^K_Y(\{r\}, \{v_i\})$,

which by Lemma 4.23 is the same as $\text{mincut}^K_{\mathcal{H}'_t \cup \mathcal{H}_t}(\{r\}, \{v_i\})$, for $(\mathcal{H}'_t, \mathcal{H}_t)$ according to Definition 4.22 for graph (V, Y).

We conclude that we can use a similar dynamic program, where the states for G_t are all possible connectivity-K mimicking networks with terminals X_t. By Theorem 4.18, there is a connectivity-K mimicking network for any graph, with w terminals, of size $O(3^K K w)$. Since the number of edges is at most the square of the number of vertices, there are $O(\exp(9^K K^2 w^2))$ possible states for each node, which completes the proof.

The proof of Theorem 4.6 follows analogously to Section 4.2.2.

CHAPTER 5

Conclusion and Open Problems

In this part of the thesis, we present a framework that allows us to extend the ideas of dynamic programming to problems containing group constraints. To achieve this, we formulate the problem as a variant of GST, where solutions correspond to valid recursions in an underlying dynamic program, and the group constraints guarantee that, for each group, one of its terminals is added to the solution at some point in the recursion.

We start by showing a set of tools to handle connectivity on bounded treewidth, making it a simple task to design a dynamic program for Steiner tree, which we can also use as the underlying dynamic program in our algorithm for GST. These tools allow us to easily generalize Steiner tree and specify more complex connectivity requirements, as long as these occur between elements in the same bag, with the possible addition of a few extra elements (such as the root for Steiner tree).

We also show how to formulate the problem of finding a solution for GST as the problem of finding a dynamic programming solution for Steiner tree that covers one terminal for each group, and we show how to approximate this problem by treating it as a variant of GST itself. Our main result is an $O(\log n \log h)$-approximation algorithm for GST on bounded-treewidth graphs, obtained by combining the ideas described above. Inspired by the use of junction trees by Chekuri et al. [CEG+11], we use our techniques to obtain an $O(\log n \log^2 h)$-approximation for GSF on bounded treewidth and an $O(w \log^2 n \log^2 h)$-approximation for DSF on bounded (undirected) treewidth.

We then turn to the fault-tolerant setting, and show that our connectivity framework can also be used to approach high-connectivity problems, such as SNDP and GroupSNDP. Unfortunately, the immediate use of this technique requires running time that is double-exponential in the maximum connectivity demand and treewidth, since each terminal or group can partition the solution into disjoint paths differently, and the dynamic program must track all of the possible possibilities.

Our main results are a dynamic program for SNDP with running time $n\, 2^{\exp(Kw \log w)}$ and an $O(\log n \log h)$-approximation for GroupSNDP with running time $n^{\exp(Kw \log w)}$, which we obtain using our technique for solving dynamic programs with group constraints. We later improve these running times for the edge-connectivity variants to $n\, 2^{\exp(K) w \log w}$ and $n^{\exp(K) w \log w}$, respectively. These improved results use a mathematical object that we call connectivity-K mimicking network, which captures the connectivity of a graph up to a certain threshold, and is much smaller than the graph, with the size depending only on the number of terminals and the maximum connectivity. We think that this data structure is of independent interest and merits further study (see [LPS19] for a different application of the same concept).

We feel that our results for GST and GroupSNDP may be important steps towards understanding the approximability of these problems, as well as showing new ways to use dynamic programming techniques to obtain approximation algorithms.

5.1 Open Problems

There are still many open problems related to our results, both for the single connectivity case of GST and its high-connectivity extensions.

For GST, the most important open question concerns the approximability of the problem, namely whether the best approximation ratio for the problem is $O(\log n \log h)$ in general graphs, and whether a poly-logarithmic approximation ratio can be achieved for GST with vertex costs. Tree embedding techniques are currently not known for vertex costs, and, even for edge costs, imply the loss of an $\Omega(\log n)$ factor.

Open Problem 5.1. Is there a polynomial time $O(\log n \log h)$-approximation for GST in general graphs?

Open Problem 5.2. Is there a polynomial time $O(\text{polylog}\, n)$-approximation for GST in general graphs with vertex costs?

Regarding our results, it would be interesting to improve the running time of our algorithm to be FPT. It may also be possible to improve the term in the exponent of the running time from $w \log w$ to simply w. While the second question may follow from carefully combining our techniques with known FPT algorithms for Steiner tree [BCK$^+$15, CNP$^+$11, FBN15], the question of whether FPT running time is achievable requires the use of new techniques.

Open Problem 5.3. Is there an $O(\log n \log h)$-approximation algorithm for GST with running time $2^{O(w)}\,\text{poly}(n)$?

Besides the running time improvements to our algorithm, there might be other useful extensions of our results. With our current algorithm, we do not use the full expressiveness of the tree decomposition, since the leaves of our STGST contain information about the state of all the ancestors. This means that we might be able to use our algorithm in more general settings, such as in graphs with small balanced separator number (see e.g. [BGH$^+$95, Gru10]). Another possible extension is to provide an approximate embedding into the STGST problem, which would imply a loss in the approximation factor, but improve the running time. The ideal situation in this scenario would be to have an $O(1)$-approximate embedding with polynomial size.

Open Problem 5.4. Can the algorithm of Theorem 3.4 be used more generally, e.g. with a balanced separator tree instead of a tree decomposition?

Open Problem 5.5. Is there an $O(1)$-approximate embedding from GST to STGST, where the instance size is polynomial in the size of the original instance, and costs are accurate up to an $O(1)$-factor?

Another natural question is to ask in what graphs the same approximation ratio applies. Two natural classes of graphs are planar graphs and Euclidean graphs. For planar graphs, Baker's technique [Bak94] has been used to obtain PTAS for diverse problems [DHK05, DHK11, Kle05], by using algorithms for bounded-treewidth graphs. For Euclidean graphs, there are PTAS for Steiner tree that use dynamic programming, which might be amenable to our techniques. As an example, combining our techniques

with the algorithm of Arora [Aro98] yields an $O(\log n \log h)$-approximation for Euclidean GST, with running time $n^{O(\log \log n)}$ in Euclidean space of constant dimension (where n is the total number of points in the groups).

We remark that every tree metric can be embedded into a Euclidean space with distortion $O(\sqrt{\log \log n})$ [LMS98, Mat99]. Therefore, the group Steiner tree problem in a Euclidean space is hard to approximate to a factor of $\log^{2-\varepsilon} n$, for any constant $\varepsilon > 0$.

Open Problem 5.6. Is there an $O(\log n \log h)$-approximation algorithm for GST on planar graphs?

Open Problem 5.7. Is there an $O(\log n \log h)$-approximation algorithm for GST on Euclidean graphs?

Our techniques lie at the intersection of combinatorial and LP rounding algorithms. We think it might be interesting to look at other kinds of algorithms to approximate GST on bounded treewidth.

Open Problem 5.8. Can we obtain an $O(\log n \log h)$-approximation for GST on graphs with bounded treewidth, using combinatorial algorithms (i.e. without explicitly solving an LP)?

Open Problem 5.9. Can we obtain an $O(\log n \log h)$-approximation for GST on graphs with bounded treewidth, by rounding an LP (potentially from an LP hierarchy, see e.g. [Lau03])?

There are also some other problems that we can approach using similar techniques. For instance, the work of Gupta et al. for the sparsest cut problem [GTW13] can be seen as using a similar technique to ours, and thus shows that such techniques may have wide applicability. Examples of some related problems where the same technique might be used (further) are GSF and GST with degree-constraints.

Open Problem 5.10. Is there an $O(\log n \log h)$-approximation algorithm for GSF on bounded treewidth?

Open Problem 5.11. Is there a bicriteria $(O(\log n \log h), \text{polylog}(n, h))$-approximation algorithm for the bounded-degree GST problem on bounded treewidth?

For higher-connectivity problems, the most interesting question is whether we can achieve a running time that is single-exponential on the treewidth and maximum connectivity demand. One way to improve the running time is to obtain a connectivity-K mimicking network with size polynomial on the number of terminals and maximum connectivity, or, more generally, to prove that the number of relevant connectivity-K mimicking networks is only single-exponential. Another question is whether we can compute a connectivity-K mimicking network in running time near-linear on the size of the graph (and FPT on the maximum connectivity). Even though a positive answer would not lead to an improvement of our results, we find this to be a question of independent interest.

Open Problem 5.12. Is there an $O(\log n \log h)$-approximation algorithm for Group-SNDP with running time $n^{\text{poly}(K) w \log w}$?

Open Problem 5.13. Does every graph G with terminals X have a connectivity-K mimicking network of size $\text{poly}(K)|X|$?

Open Problem 5.14. Is there a set $\mathcal{M}$ of $\exp(\text{poly}(K)w))$ graphs such that every graph G with w terminals has a connectivity-K mimicking network in $\mathcal{M}$?

Open Problem 5.15. Is there an algorithm that computes a connectivity-K mimicking network of size $\exp(K)|X|$ with running time $\exp(K)\, n \,\text{polylog}\, n$?

Finally, there are other problems where the use of our techniques might lead to improved results. The most clear examples are the unrooted and relaxed variants of GroupSNDP, but other high-connectivity problems may also benefit from the use of our framework.

Open Problem 5.16. Is there an $O(\log n \log h)$-approximation algorithm for unrooted GroupSNDP?

Open Problem 5.17. Is there an $O(\log n \log h)$-approximation algorithm for relaxed GroupSNDP?

PART II

Firefighter Problem

This part is the result of close collaboration with Parinya Chalermsook and Erik Jan van Leeuwen. It is based on an article presented at the *Workshop on Approximation and Online Algorithms* in 2016 [CV16], as well as unpublished work [vLV].

CHAPTER 6

Firefighter Problem on Trees

The firefighter problem was introduced by Hartnell in 1995 [Har95] as a model for fire spreading on a graph. It can be described as a one-player game, where the goal is to strategically place firefighters, so as to maximize the number of vertices saved. At the beginning of the game, a vertex s is *burning*. At each time step, the player can pick some non-burning vertices to *protect*, which remain protected for the rest of the game. Then, the fire spreads from every burning vertex to its neighbors that are not protected, with the game stopping when the fire can no longer spread. We say that a vertex is *saved* if it is not burning at the end of the game, either because it was protected or because the fire was blocked off by protected vertices.

The firefighter problem can also model other spreading phenomena, such as infectious diseases, computer viruses, or even ideas. In any of these scenarios, protecting a vertex represents any kind of defensive action, such as vaccinations, quarantine, or disconnecting a node of the network. In the field of epidemiology, some recent studies have looked at interactions between individuals, modeled as a graph, and how diseases spread in this model [RK03, SW16, Cra15]. The firefighter problem and its algorithms might be useful to strategically use a limited stock of vaccines in such a way that contains the spread of a disease (see also [DH07, Har04]).

There are two important variants of the firefighters problem: in the maximization variant (Max-FF), we are allowed to protect one vertex per time step, and the objective is to maximize the number of saved vertices; in the minimization variant (Min-FF), we are given a terminal set $\mathcal{X} \subseteq V(G)$, we are allowed to pick B vertices per time step, and the goal is to save all terminals in $\mathcal{X}$, while minimizing the budget B.

In this chapter, we will focus on the Max-FF problem. Even though it is NP-hard to $n^{1-\varepsilon}$-approximate the problem in general graphs [ACH$^+$12], there are still many open questions on more specific settings, such as trees, grids or other sparse graphs (see e.g. [FM09]). On trees, the problem remains NP-hard, but better approximation results are known: Hartnell and Li give a simple 2-approximation algorithm [HL00]; an LP-rounding algorithm by Cai et al. achieves a $(1 - 1/e)$-approximation [CVY08], which Costa et al. [CDD$^+$13] extend to the case of multiple fire sources and firefighters; Iwaikawa et al. [IKM11] improve the approximation ratio slightly when the degree of the tree is bounded. Recently, Adjashvili et al. presented a PTAS for the problem on trees [ABZ17]. Their results use the standard LP for the problem as a subroutine, but do not bound the integrality gap. We believe that the integrality gap questions are interesting despite the known approximation guarantees.

In this chapter, we study Max-FF on trees, which we henceforth refer to as Tree-FF. Most of the results for Tree-FF heavily rely on the graph structure, and therefore do not usually generalize to other classes of graphs (for example, when graphs have cycles). We explore the approximability of Max-FF in more general graphs in Chapter 7.

Related work Tree-FF has several connections to classic optimization problems (that do not apply to Max-FF in general):

- **Set cover:** On the one hand, Tree-FF can be thought of as a maximum coverage problem, and is a special case of the maximum coverage problem with group budget constraints [CK04];
- **Submodular optimization:** In a similar vein, Tree-FF can also be modeled as a submodular optimization problem under matroid constraints [CCP+11], where the matroid constraints restrict the sets of vertices that can be protected;
- **Cut Problems:** Anshelevich et al. [ACH+12] discussed that solutions to the firefighter problem can be seen as a "cut over time", in which the cut must be produced gradually over many timesteps. In other words, in each time step t, the player is allowed to remove a set of vertices from the graph. The final goal is then to "disconnect" s from a set of vertices T.

King and MacGillivray show that Tree-FF is solvable in polynomial time if the input tree has degree at most three, with the fire starting at a vertex with degree at most 2 [KM10]. Cai et al. present an exact algorithm with running time $2^{O(\sqrt{n}\log n)}$ [CVY08]. The problem was also studied from the point of view of parameterized algorithms (e.g. [BCC+14, CVY08, CC14]) and on many special cases, e.g. when the tree has bounded pathwidth [CC14] as well as on bounded degree graphs [BCR13, CC14]. Chalermsook and Chuzhoy study the Min-FF problem and show an $O(\log^* n)$-approximation [CC10]. This result was later improved to an $O(1)$-approximation by Adjashvili et al. [ABZ17].

Another question of interest, particularly in the discrete mathematics community, is whether a fire can be stopped at all for graphs following a fixed structure. Of particular interest are infinite graphs, such as grids or regular graphs, where we want to distinguish between containment, i.e. a finite number of vertices burns, or non-containment, i.e. an infinite number of vertices burns (see [DH07, FHL+00, Fog03, Har95, MW03, WM02]).

The surviving rate of a graph is the expected fraction of vertices that burns if the fire source is chosen at random and vertices are protected optimally. Determining the surviving rate is a problem that has been studied for different classes of graphs, including trees [CW09], graphs of bounded treewidth [CCV+10] and planar graphs [Gor15]. Finbow et al. [FHL+00] look at the related question of determining which graphs maximize the surviving rate.

Our results Our goal is to develop a better understanding of Tree-FF from the perspective of linear programming. The known $(1-1/e)$-approximation was obtained from the standard LP by independent rounding. Cai et al. [CVY08] claimed that this result is the best possible for LP-respecting algorithms, that is, algorithms which only protect vertices that are in the support of the LP solution. However, many algorithms in other contexts are not LP-respecting (such as the algorithm by Chalermsook and Chuzhoy [CC10] for Min-FF), and hence the question of whether rounding the standard LP could lead to a better approximation ratio was open.

Our first result (formalized in Theorem 6.5) proves that this is not possible, that is, the integrality gap of the standard LP relaxation can be arbitrarily close to $(1-1/e)$. This result also generalizes for any constant budget $B \in \mathbb{Z}_{\geq 1}$, matching the algorithmic result

of Costa et al. [CDD+13]. Furthermore, if the degree of the input tree is upper bounded by a constant $d \in \mathbb{Z}_{\geq 4}$, we show an integrality gap result of $(1 - 1/e + O(1/\sqrt{d}))$. The best approximation ratio in this setting is $(1 - 1/e + \Omega(1/d))$ by Iwaikawa et al. [IKM11].

Motivated by the negative results above, we analyze a stronger LP relaxation, suggested by Hartke [Har04], that adds some constraints to the standard LP. We denote this new relaxation LP-Hartke. Even though Hartke showed experimentally that the new relaxation outperforms the standard LP, he did not provide any theoretical analysis for the new LP. We show some evidence that LP-Hartke is a stronger relaxation than the standard LP, namely:

- For the tractable instances studied by Finbow et al. [FM09], any extreme point of the new LP is integral. In contrast, the extreme point solutions of the standard LP are not integral in these instances.
- A family of instances, capturing the integrality gap instances of Theorem 6.5, admits a better approximation ratio than $(1 - 1/e)$ by rounding the new LP.
- When the LP solution is near-integral, e.g. for half-integral solutions, the new LP is provably better than the old one.

Our results provide the first indications that LP-Hartke might lead to improvements over the standard LP in general instances. All of the above results exploit the new constraints to show that the solutions to the new LP are more structured than those of the standard LP. We use this additional structure to propose a new two-phase dependent rounding algorithm, which leads to the improved approximation results.

We believe that LP-Hartke has an integrality gap strictly better than $(1 - 1/e)$, but are unable to show it formally. However, we prove that even using LP-Hartke we cannot get a PTAS for Tree-FF, as the integrality gap of LP-Hartke is at most 5/6.

Organization The organization of this chapter is as follows: in Section 6.1, we present the problems and state our results formally; in Section 6.2, we present the standard LP, as well as some observations about the problem; in Section 6.3, we show that the integrality gap of the standard LP is arbitrarily close to $1 - 1/e$; finally, in Section 6.4 we show the power of LP-Hartke, by proving that it outperforms the standard LP in different types of instances.

6.1 Problem Definitions and Results

Definition 6.1 (Valid Strategy)**.**

A *strategy* is a collection $\{S_i\}_{i\in[n]}$ where $S_i \subseteq V$, is the set of vertices protected at time step i. We denote by $V(S) = \bigcup_{i\in[n]} S_i$ the set of all vertices protected by S. We say S is *applied* if the vertices in S_i are protected at time step i, for every $i \in [n]$.

A *valid strategy* satisfies the constraint that no vertex is protected when burning, that is, $d'(s, S_i) \leq i$, where d' refers to the distance in $G \setminus \bigcup_{j=1}^{i-1} S_j$.

A strategy has *budget* B if $|S_i| \leq B$ for all $i \in [n]$.

A vertex $v \in V$ is *saved* by a valid strategy $\{S_i\}_{i\in[n]}$ if $v \in S_i$ for some $i \in [n]$ or s is disconnected from v in $G \setminus V(S)$.

Problem 6.2: Firefighter Problem (Max-FF).

- INSTANCE: (G, s, w, B), where
  - $G = (V, E)$ is a graph with vertex weights $w : V \to \mathbb{R}$ (by default, $w = \mathbf{1}$);
  - $s \in V$ is the *source* of a fire;
  - $B \in \mathbb{Z}_{\geq 1}$ is the *budget*, that is, the number of vertices that can be protected per time step (by default, $B = 1$).
- SOLUTION: a valid strategy $\{S_i\}_{i\in[n]}$ with budget B ($|S_i| \leq B$).
- GOAL: maximize the weight of the set of vertices R that is saved, $w(R) = \sum_{v\in R} w_u$

Problem 6.3: Firefighter Problem on Trees (Tree-FF).

- INSTANCE: (G, s, w, B), where
  - (G, s, w, B) is an instance of Max-FF;
  - G is a tree.
- SOLUTION: a valid strategy $\{S_i\}_{i\in[n]}$ with budget B ($|S_i| \leq B$).
- GOAL: maximize the weight of the set of vertices R that is saved, $w(R) = \sum_{v\in R} w_u$

Problem 6.4: Minimization Firefighter Problem (Min-FF).

- INSTANCE: $(G, s, \mathcal{X})$, where
  - $G = (V, E)$ is a graph;
  - $s \in V$ is the *source* of a fire;
  - $\mathcal{X} \subseteq V$ is a set of *terminals*.
- SOLUTION: a valid strategy $\{S_i\}_{i\in[n]}$ that saves all the terminals.
- GOAL: minimize the budget $B = \max\{|S_i| : i \in [n]\}$

Theorem 6.5. *Let $\varepsilon > 0$, $B \geq 1$. There is an instance $(G, s, \mathbf{1}, B)$ of Tree-FF of size bounded by some function $f(\varepsilon, B)$, such that the integrality gap of the standard LP is*

- *at most $(1 - 1/e + \varepsilon)$;*
- *at most $\big(1 - (1 - 1/k)^k + \varepsilon\big)$, when compared to $1/k$-integral solutions, for any $k \leq 2/\varepsilon$.*

Theorem 6.6. *Let $B \geq 1$ and $d \geq 2$ be integers such that $B < d/5$. There is an instance (G, s, w, B) of Tree-FF of size bounded by some function $f(d)$, such that the integrality gap of the standard LP is at most $(1 - 1/e + O(\sqrt{B/d}))$.*

Theorem 6.7. *Let $(T, s, w, 1)$, $w > 0$ be an instance of Tree-FF as described by Finbow et al. [FM09], and let x be an extreme point solution for* LP-HARTKE. *Then x is integral.*

Furthermore, there is an instance $(T, s, w, 1)$, for which the standard LP has an non-integral extreme point solution.

Theorem 6.8. *Let $(T, s, w, 1)$ be an instance of Tree-FF, and let x be a half-integral extreme point solution for* LP-HARTKE.

Then there is a valid strategy $\{S_i\}_{i\in[n]}$ that saves vertices with a weight of at least $5/6\, w^T x$.

Theorem 6.9. *Let $(T, s, w, 1)$ be an instance of Tree-FF, and let x be a* separable *(see Definition 6.29) extreme point solution for* LP-Hartke.

Then there is a valid strategy $\{S_i\}_{i\in[n]}$ that saves vertices with a weight of at least $\alpha\, w^T x$, where $\alpha > 1 - 1/e$.

Theorem 6.10. *Let $\varepsilon > 0$. There is an instance $(T, s, \mathbf{1}, 1)$ of Tree-FF of size bounded by some function $f(\varepsilon)$, such that the integrality gap of* LP-Hartke *is at most $5/6 + \varepsilon$.*

6.2 Standard Linear Program and Preliminaries

We will start by presenting the standard LP, as well as some assumptions that simplify our presentation. Let (T, s, w, B) be an instance of Tree-FF. We assume that T is rooted at s and then partition the vertices into layers, according to to their distance from the root. Formally, we have layers $L_0, L_1, \ldots, L_\lambda$, where, for each $i \in [\lambda]$, L_i contains all the vertices at depth i.

For any vertex v, let P_v denote the (unique) path from s to v. A natural LP relaxation is as follows: for each $v \in V$, we have variables x_v indicating whether v is protected by the solution, and y_v indicating whether v is saved.

Definition 6.11 (Standard LP for Tree-FF – LP-FF)**.**

$$\begin{aligned}
\max \quad & \sum_{v\in V} w_v y_v & \\
\text{s.t.} \quad & \sum_{v\in L_j} x_v \le B & \forall j \in [\lambda] \\
& y_v \le \sum_{u\in P_v} x_u & \forall v \in V \\
& x_v, y_v \in [0,1] & \forall v \in V
\end{aligned}$$

Notice that, in this LP relaxation, we assume that the vertices protected in each time step are contained in the layer corresponding to that time step. We say a strategy $\{S_i\}_{i\in[n]}$ is *layered* if $S_i \subseteq L_i$ for all $i \in [n]$. From this point onwards, and unless otherwise specified, we assume that all the strategies are layered. Proposition 6.12 proves that these assumptions are valid for Tree-FF (but not Max-FF in general).

Proposition 6.12. *Let (T, s, w, B) be an instance of Tree-FF, with layers $L_0, L_1, \ldots, L_\lambda$, and $S = \{S_i\}_{i\in[n]}$ be a valid strategy.*

Then there is a valid strategy $S' = \{S'_i\}_{i\in[\lambda]}$ that is layered and at least as good as S, that is:

(1) S' saves all the vertices that S saves;

(2) S' protects at most as many vertices as S at each time step, that is $|S'_i| \le |S_i|$ for all $i \in [n]$;

(3) S' is layered, that is, $S'_i \subseteq L_i$ for $i \in [\lambda]$.

Proof. For every $i \in [\lambda]$, and every $v \in S_i$ that is not contained in L_i, replace v by the ancestor v' of v that is in L_i. This is still a valid strategy, since vertices in L_i cannot burn before time step i. Furthermore, every vertex saved by v must also be saved by v', since v disconnects T_v from s, and v' disconnects $T_{v'} \supset T_v$. This proves the statement. □

Lemma 6.13. *Let $\varepsilon > 0$ and let (T, s, w, B) be an instance of the Max-FF problem, with integrality gap γ for* LP-FF.

There is an instance $(T', s, \mathbf{1}, B)$, with $T \subseteq T'$ with integrality gap of $\gamma+\varepsilon$ for LP-FF.

Proof. Let $w_{\max} = \max_{v \in V} w_v$ and $M = 3B(n+1)/\varepsilon w_{\max}$, where $n = |V(T)|$. Starting from T, we construct an instance $(T', s, \mathbf{1}, B)$ by adding $\lceil M w_v \rceil$ children to each vertex $v \in V$. Let $\mathcal{X} = V(T') \setminus V(T)$ denote the set of newly added vertices. Since $w_v \leq w_{\max}$ for all $v \in V$, we add at most $n \cdot 3B(n+1)/\varepsilon$ vertices, that is, $|V(T')| = O(Bn^2/\varepsilon)$.

Let (x, y) be the optimum LP solution for (T, s, w, B), and let $\text{opt} = w^T y$. We can construct a solution (x, y') for $(T', s, \mathbf{1}, B)$ by setting $y'_v = y_v$, $v \in V(T)$, and $y'_v = y_{p(v)}$ for $v \in \mathcal{X}$. By construction,

$$\text{opt}' = \mathbf{1}^T y' \geq \sum_{v \in V(T)} y_v \lceil M w_v \rceil \geq \sum_{v \in V(T)} y_v M w_v = M w^T y = M \,\text{opt}$$

Any integral solution can protect B vertices in $\mathcal{X}$ per layer, for a total of at most $(n+1)B$ vertices. Let R be the set of saved vertices by some integral solution for T'. Then the total weight saved is at most

$$\begin{aligned}
|R| &\leq \sum_{v \in V(T) \cap R} \left(1 + \lceil w_v M \rceil\right) + (n+1)B \\
&\leq \sum_{v \in V(T) \cap R} w_v M + 2n + (n+1)B \\
&\leq M\gamma \,\text{opt} + 2n + (n+1)B \\
&\leq M \,\text{opt} \left(\gamma + \frac{3(n+1)B}{M \,\text{opt}}\right) \\
&= \text{opt}' \left(\gamma + \frac{\varepsilon w_{\max}}{\text{opt}}\right) \\
&\leq \text{opt}' \left(\gamma + \varepsilon\right)
\end{aligned}$$

The last inequality follows from $\text{opt} \geq w_{\max}$, since the optimum solution can at least save the vertex of maximum weight.

We conclude that $(T', s, \mathbf{1}, B)$ has integrality gap $\gamma + \varepsilon$. □

6.3 Integrality Gap Instances for the Standard LP

In this section, we consider the goal of maximizing the number of *leaves* saved, as this simplifies the analysis dramatically. This corresponds to setting the weight on every leaf to 1 and the weight of every other vertex to 0. By Lemma 6.13, we can transform any instance into an instance with uniform weights, $w = \mathbf{1}$, in such a way that the integrality gap is only affected by a factor of $1 + \varepsilon$. This implies that all of the results below also apply to the uniform weight setting.

We construct the integrality gap instance in sets of contiguous layers, which we call *phases*, such that, in each phase, the LP can fractionally protect a vertex in *every* top to bottom path. By having k phases, and in each of them having the LP solution save each leaf by $1/k$, we get that all of the leaves are completely saved by the LP solution. On the other hand, we prove that any integral solution saves a fraction of at most $(1 - 1/e + O(1/k))$ of the leaves, meaning that, by increasing k, we can make the integrality gap arbitrarily close to $(1 - 1/e)$.

From now on, we will focus on proving Theorem 6.5. Let $\varepsilon > 0$. We will start by defining a construction that we will use as a building block to obtain the final instance.

Definition 6.14 (Good Gadget)**.** An (M, k', δ)-*good gadget* is a collection of trees $\mathcal{T} = \{T_1, \ldots, T_M\}$, with roots $r_1, \ldots, r_M$ (r_i is the root of T_i), as well as a subset $\mathcal{S} \subseteq \bigcup_{i \in [M]} V(T_i)$, satisfying the following properties:

- *Uniform depth:* We think of the gadget as having layers $L_0, L_1, \ldots, L_h$, where L_j is the union over all trees of all vertices at depth j and $L_0 = \{r_1, \ldots, r_m\}$. All leaves are in the same layer L_h.
- *LP-friendly:* $\mathcal{S}$ contains k' vertices per layer, that is, for $j \in [h]$, $|\mathcal{S} \cap L_j| \leq k'$ (and $|\mathcal{S} \cap L_0| = 0$). Moreover, for any tree T_i and a leaf $v \in V(T_i)$, the unique path from r_i to v must contain exactly one vertex in $\mathcal{S}$.
- *Integrally adversarial:* Let $1 \leq B \leq k'$, $\mathcal{B} \subseteq \{r_1, \ldots, r_M\}$ be a subset of roots and $\mathcal{U} = \{U_j\}_{j=1}^{h}$ be a valid layered strategy with budget B. A vertex $v \in T_i$ is $(\mathcal{U}, \mathcal{B})$-*risky* if $r_i \in \mathcal{B}$ and the unique (r_i, v)-path does not contain any vertex from $\mathcal{U}$.

  The number of $(\mathcal{U}, \mathcal{B})$-risky vertices in L_h is at least

  $$\left(1 - \frac{B}{k'}(1 + \delta)\right) \frac{|\mathcal{B}|}{M} |L_h|,$$

  for any $1 \leq B \leq k'$, $\mathcal{B} \subseteq L_0$, valid strategy $\mathcal{U}$ with budget B.

We say that vertices in $\mathcal{S}$ are *special* and all other vertices are *regular*. We say a vertex is $(\mathcal{U}, \mathcal{B})$-*safe* if it is not $(\mathcal{U}, \mathcal{B})$-risky.

The following two results combined are sufficient to prove the theorem.

Lemma 6.15. *Let $k' \geq 2$, $M \geq 1$ be integers, and $\delta > 0$ be a real number.*

Then, an (M, k', δ)-good gadget exists. Moreover, it contains at most $O(k'/\delta^2)^M$ vertices.

Lemma 6.16. *Let $k \geq 2$, $B \geq 1$ be integers and let $\delta > 0$. Assume that (M, kB, δ)-good gadgets exist for any $M > 0$.*

Then we can construct an instance (G, s, w, B) of size bounded by some function $f(B, k, \delta)$, such that the integrality gap of the standard LP is at most $\big(1 - (1 - 1/k)^k + \delta\big)$ (even against $1/k$-integral solutions).

Furthermore, if for some integer $d \geq 2$, d-ary (M, kB, δ)-good gadgets exist for any $M > 0$, then the construction above is also d-ary.

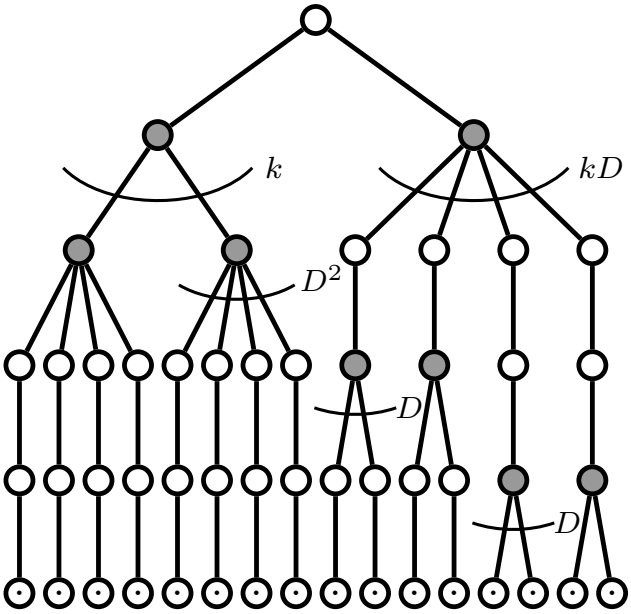

Figure 6.1: Simplified example of the instance used to achieve integrality gap of $1-1/e$ for LP-FF. The values used are $k = 2$ and $D = 2$. The *labels* in the figure indicate, in general, the number of edges in that location, in terms of k and D. Special vertices are colored *gray*.

Setting $k = \lceil 2e^{-1}/\varepsilon \rceil$, $\delta = \varepsilon/2$, we get an integrality gap of at most:

$$\begin{aligned} 1 - (1-1/k)^k + \frac{\varepsilon}{2} &\leq 1 - e^{-1}\left(1 - \frac{1}{k}\right) + \frac{\varepsilon}{2} \\ &= 1 - e^{-1} + \varepsilon \end{aligned}$$

Hence, we obtain an instance for which the integrality gap is $1 - e^{-1} + \varepsilon$, which finishes the proof of Theorem 6.5. The integrality gap of the LP against $1/k$-integral solutions follows directly from setting $\delta = \varepsilon$.

We first show how to derive the final construction, assuming the existence of a good gadget (Lemma 6.16). The proof of Lemma 6.15 follows later. Figure 6.1 exemplifies the construction for $k = 2$.

6.3.1 Constructing the Instance (Lemma 6.16)

Our construction proceeds in k phases, where k is a parameter of the construction. We show how to construct the instance inductively. For $B > 1$, the parameters of the good gadget must be scaled up by a factor of B, so that the construction works out. For this purpose, we use $k' = kB$. Let α be the height of the instance.

The first phase of the construction is simply a $(1, kB, \delta)$-good gadget, where the single root r_1 is the root of the instance. We now describe how to construct phase $q+1$, assuming that we have constructed the instance up to phase q.

Let $l_1, \ldots, l_{M_q} \in L_{\alpha_q}$ be the M_q leaves after the construction of phase q, with all of the leaves in layer α_q. For phase $q+1$, we take the (M_q, kB, δ)-good gadget

$(\mathcal{T}_q, \{r_j\}_{j\in[M_q]}, \mathcal{S}_q)$; recall that such a gadget consists of M_q trees. For each $i = 1, \ldots, M_q$, we unify each root r_i with the leaf l_i. This completes the description of the construction.

Denote by $\tilde{\mathcal{S}}_q = \bigcup_{q'\leq q} \mathcal{S}_{q'}$ the set of all special vertices in the first q phases. After phase q, we argue that our construction satisfies the following properties:

Lemma 6.17. *Let $1 \leq q \leq k$.*

(1) All of the leaves are in the same layer α_q.

(2) For every layer L_j, $j \leq \alpha_q$, it holds that $|L_j \cap \tilde{\mathcal{S}}_q| \leq kB$. For every path P from the root to $v \in L_{\alpha_q}$, $|P \cap \tilde{\mathcal{S}}_q| = q$.

(3) For any valid strategy $\mathcal{U}$ with budget B, the number of vertices in L_{α_q} that end up burning is at least

$$\left|L_{\alpha_q}\right| \left(\left(1 - \frac{1}{k}\right)^q - \delta\frac{q}{k}\right)$$

Proof. By construction, all of the leaves after phase q are in the same layer, since, in each of the phases, the distance from a leaf to its root is always the same.

As to the second property, when constructing the instance, each gadget occupies disjoint layers. Therefore, the gadget properties ensure that there are at most $k' = kB$ special vertices per layer. Moreover, consider any path P from the root to some vertex $v \in L_{\alpha_q}$. We will prove that $|P \cap \tilde{\mathcal{S}}_q| = q$. For $q = 1$, the instance is a single gadget, and, therefore, the statement holds by the gadget properties. For any $q \geq 2$, assume that the statement holds for the instance up to phase $q-1$. We can split this path into two parts $P = P' \cup P''$ where P' starts from the root and ends at some vertex $v' \in L_{\alpha_{q-1}}$, and P'' starts at v' and ends at v. By the induction hypothesis, $|P' \cap \tilde{S}_{q-1}| = q - 1$ and the second property of the gadget guarantees that $|P'' \cap \mathcal{S}_q| = 1$, which proves the statement.

To prove the final property, consider any valid strategy $\mathcal{U} = \{U_j\}_{j=1}^{\alpha_q}$ with budget B. By Proposition 6.12, we can assume that $U_j \subseteq L_j$. If $q = 1$, then the statement follows by the gadget properties, using $\mathcal{B} = \{r\}$, $M = 1$ and $k' = kB$, which implies that at least $(1 - 1/k - \delta/k)|L_h|$ of the vertices always burn, independently of the strategy.

For $q \geq 2$, assume that the statement is true for the first $q-1$ phases. We can partition $\mathcal{U}$ into $\mathcal{U}' \cup \mathcal{U}''$, where $\mathcal{U}' = \{U_j\}_{j=1}^{\alpha_{q-1}}$ corresponds to a valid strategy for the first $q-1$ phases, and $\mathcal{U}'' = \{U_j\}_{j=1+\alpha_{q-1}}^{\alpha_q}$ corresponds to a valid strategy for the last gadget. By the induction hypothesis, we have that at least $\big((1-1/k)^{q-1} - (q-1)\delta/k\big)|L_{\alpha_{q-1}}|$ vertices in $L_{\alpha_{q-1}}$ burn; denote these burning vertices by $\mathcal{B}$.

We now prove that every $(\mathcal{U}'', \mathcal{B})$-risky vertex ends up burning, and that the number of these vertices is large enough. Let $v \in L_{\alpha_q}$ be a $(\mathcal{U}'', \mathcal{B})$-risky vertex. There is no vertex of P_v protected by $\mathcal{U}'$ in the first $q-1$ phases, since the ancestor of v in layer α_q is in $\mathcal{B}$, and hence burning. Furthermore, by the definition of risky, there is also no ancestor of v protected by $\mathcal{U}''$. Hence, v ends up burning. By the properties of the gadget, the number of burning vertices in layer L_{α_q} is:

$$\begin{aligned}(1 - 1/k - \delta/k)\,\frac{|\mathcal{B}|}{|L_{\alpha_{q-1}}|}\left|L_{\alpha_q}\right| &\geq (1 - 1/k - \delta/k)\big((1-1/k)^{q-1} - (q-1)\delta/k\big)\left|L_{\alpha_q}\right| \\ &\geq \left((1-1/k)^q - q\delta/k\right)\left|L_{\alpha_q}\right|\end{aligned}$$

This concludes the proof of the lemma. □

After the construction is finished, we can set the weights as mentioned before: $w_v = 1$ for every leaf $v \in L_\alpha$ and $w_v = 0$ otherwise. By Lemma 6.15, we know that the size of the gadget is at most $O(k'/\delta^2)^M$. By simple induction, we get that the size of the instance is a tower function of $O(k'/\delta^2)$, which we can represent as $O(k'/\delta^2) \uparrow\uparrow k$ [1].

In order to finish the proof of Lemma 6.16, it is now sufficient to use Lemma 6.17 to define an LP solution. We set $x_v = 1/k$ for all the vertices in $\mathcal{S}$, and $x_v = 0$ otherwise. Since in each gadget, every root to leaf path goes through one special vertex, then every root to leaf path in the complete instance goes through k special vertices, and therefore we can set $y_u = 1$ for every leaf $u \in L_\alpha$. Lemma 6.17 directly implies the integrality gap of the instance (against $1/k$-integral solutions). We remark that if all the gadgets used are d-ary, so is the instance, as the only operation used in our construction is to unify leaves (which have no children) with other vertices.

6.3.2 Existence of Good Gadgets (Lemma 6.15)

We now show that an (M, k', δ)-good gadget exists for any value of $M \in \mathbb{Z}_{\geq 1}$, $k' \in \mathbb{Z}_{\geq 2}$ and $\delta > 0$. We first describe the construction and then show that it has the desired properties.

Construction Throughout the construction, we use a structure which we call *spider*. A spider is a tree in which every vertex except the root has at most one child. If a vertex has no children (i.e. a leaf), we call it a *foot* of the spider. We call the paths from the root to each foot the *legs* of the spider.

Let $D = 1 + \lceil 2/\delta \rceil$. Our construction is composed of M trees $T_1, \ldots, T_M$, where each tree T_i has a spider at the root, and the feet of these spiders form set $\mathcal{S}$. We then need to balance these trees, so that they all have the same height and the same number of leaves. To achieve this, we add a spider to each foot of the main spiders.

Formally, for each $i = 1, \ldots, M$, the tree T_i is constructed as follows: We have a spider rooted at r_i that contains $k'D^{i-1}$ legs. Its feet are in D^{i-1} consecutive layers, starting at layer $\alpha_i = 1 + \sum_{j=1}^{i-1} D^{j-1}$; each such layer has k' feet. Denote by $\mathcal{S}^{(i)}$ the feet of these spiders. Next, for each vertex $v \in \mathcal{S}^{(i)}$, we have a spider rooted at v with D^{M-i+1} feet, all of which belong to layer $\alpha = 1 + \sum_{j=1}^{M} D^{j-1}$. The set $\mathcal{S}$ is defined as $\mathcal{S} = \bigcup_{i=1}^{M} \mathcal{S}^{(i)}$.

We use the following observation:

Observation 6.18. *For every $i \in [M]$, the number of leaves of T_i is $k'D^M$.*

Furthermore, the gadget has $\alpha \leq D^M$ layers. Therefore, the size of each T_i is at most $k'D^{2M}$ and the size of the gadget is bounded by $Mk'D^{2M} \leq (k'D^2)^M$.

Analysis We now prove that the above gadget is (M, k', δ)-good. The construction ensures that all leaves are in the same layer L_α.

The second property also follows from the construction: for any two trees T_i, T_j, $i, j \in [M]$, $i \neq j$, the set of feet of the spider rooted at r_i and r_j are in disjoint sets of layers, and therefore, $\mathcal{S}^{(i)} \cap \mathcal{S}^{(j)} = \emptyset$. Since each $\mathcal{S}^{(i)}$ has at most k' vertices per layer,

[1] $a \uparrow\uparrow b := a^{a \uparrow\uparrow (b-1)}$, $a \uparrow\uparrow 1 = a$

so does $\mathcal{S}$. Moreover, any path from r_i to a leaf of T_i must go through a vertex in $\mathcal{S}^{(i)}$, since it must go through one of the legs of the spider to reach the leaf from the root.

The third and final property is established by the following claim.

Claim 6.19. *Let $B \geq 1$, $\mathcal{B}$ be a subset of roots and $\mathcal{U} = \{U_j\}_{j=1}^{h}$ be a valid strategy with budget B.*

For any $i \in [M]$, such that $r_i \in \mathcal{B}$, the fraction of leaves of T_i that are $(\mathcal{U}, \mathcal{B})$-safe is at most $B/k'(1 - 2/(D-1))$.

Proof. We recall that a vertex u is $(\mathcal{U}, \mathcal{B})$-risky if u is in a tree T_i rooted at some $r_i \in \mathcal{B}$, and $\mathcal{U}$ does not cut u from r_i; otherwise, u is $(\mathcal{U}, \mathcal{B})$-safe.

All of the vertices in $\mathcal{S}^{(i)}$ are contained in the first $\alpha_{i+1} - 1$ layers. If a vertex $v \in \mathcal{S}^{(i)}$ or one of its ancestors is in $\mathcal{U}$, then all of the D^{M-i+1} descendants of v in L_h will be $(\mathcal{U}, \mathcal{B})$-safe. Any other vertex is part of a spider rooted at some $v \in \mathcal{S}^{(i)}$, and therefore adding it to $\mathcal{U}$ only makes 1 leaf $(\mathcal{U}, \mathcal{B})$-safe.

We conclude that the number of leaves that are $(\mathcal{U}, \mathcal{B})$-safe is at most D^{M-i+1} for any vertex in $\mathcal{U}$ up to layer $\alpha_{i+1} - 1$, and at most 1 for any other vertex in $\mathcal{U}$ up to the layer α. Since there are at most B vertices in $\mathcal{U}$ for each layer, the number of such leaves is

$$\begin{aligned}
&B(\alpha_{i+1} - 1)D^{M-i+1} + B(\alpha - \alpha_{i+1} + 1) \\
&\qquad \leq B\left(\sum_{j=1}^{i} D^{j-1}\right) D^{M-i+1} + B\left(\sum_{j=1}^{M} D^{j-1} - \sum_{j=1}^{i} D^{j-1} + 1\right) \\
&\qquad \leq B\frac{D^i}{D-1}D^{M-i+1} + B\frac{D^M}{D-1} \\
&\qquad \leq BD^M\left(\frac{D+1}{D-1}\right)
\end{aligned}$$

The total number of leaves in each tree is $k'D^M$, and therefore the fraction of leaves that are $(\mathcal{U}, \mathcal{B})$-safe is:

$$\frac{BD^M\left(\frac{D+1}{D-1}\right)}{k'D^M} = \frac{B}{k'}\left(1 + \frac{2}{D-1}\right) \qquad \square$$

Since $D \geq 1 + 2/\delta$, $2/(D-1) \leq \delta$, and therefore, the total number of $(\mathcal{U}, \mathcal{B})$-risky vertices is at least $(1 - B/k'(1+\delta))k'D^M$ for each tree T_i with $r_i \in \mathcal{B}$, for a total of:

$$\left(1 - \frac{B}{k'}(1+\delta)\right)\frac{|\mathcal{B}|}{M}|L_h|.$$

6.3.3 Bounded-Degree Integrality Gap Instances

Iwaikawa et al. [IKM11] showed a $(1 - 1/e + \Omega(1/d))$-approximation algorithm for d-ary trees (with budget $B = 1$). We show an instance whose integrality gap is almost tight to this approximation factor. The rest of this section will be dedicated to to proving Theorem 6.6, which we recall below.

Theorem 6.6 (page 88). *Let $B \geq 1$ and $d \geq 2$ be integers such that $B < d/5$. There is an instance (G, s, w, B) of* Tree-FF *of size bounded by some function $f(d)$, such that the integrality gap of the standard* LP *is at most $(1 - 1/e + O(\sqrt{B/d}))$.*

To prove this theorem, we construct a bounded-degree analogue of our good gadgets. Specifically, we are going to use a d-ary $(M, k', (k'+2)/(d-1))$-good gadget for our construction, where $k' < d$.

Lemma 6.20. *Let $k' \geq 2$, $d \geq 2$ be integers such that $k' < d$.*
A d-ary $(M, k', (k'+2)/(d-1))$-good gadget exists for any $M > 0$.

Proof. The construction is similar to Lemma 6.15; we will build upon it by pointing out the differences. First, we set $D = d$. Besides this parameter change, the only difference is that we replace a vertex with $\ell > d$ children by a suitable d-ary tree with ℓ leaves.

Let $i \in [M]$. More specifically, instead of each r_i having $k'D^{i-1}$ children, we root a d-ary tree at r_i with $k'D^{i-1}$ leaves, in which each vertex in layers 0 through $i-2$ has exactly D children, and the vertices at layer $i-1$ have k' children. For each of these leaves, there is a path of the appropriate length so that we have k' special vertices in each layer from α_i to $\alpha_{i+1} - 1$. We remark that this tree has height $i \leq \alpha_i - 1$, so the entire tree fits in the layers before α_i, and hence the construction can be analogous to the construction of Lemma 6.15.

Similarly, we root a complete D-ary tree with D^{M-i+1} leaves at each special vertex, with a path starting at each of those leaves and ending at layer α. For the construction to be sound, the leaves of the tree have to be in layer at most α. Since the last special vertices are in layer $\alpha_{i+1} - 1$, there are at least $\alpha - (\alpha_{i+1} - 1)$ layers available. It is now sufficient to prove that the height of the complete D-ary tree, $M - i + 1$, does not exceed the layers available. Indeed,

$$\begin{aligned} \alpha - (\alpha_{i+1} - 1) &= 1 + \sum_{j=i+1}^{M} D^{j-1} \\ &\geq 1 + M - i, \end{aligned}$$

so we can conclude that the construction is well-defined.

It follows from the proof of Lemma 6.15 that this gadget has uniform depth and is LP-friendly. We will now show that it is also integrally adversarial. We will show the last property by examining how many vertices can be made $(\mathcal{U}, \mathcal{B})$-safe by adding a vertex to $\mathcal{U}$ in each layer:

- In layer $j \in [i-1]$, each vertex has exactly $k'D^{M-j}$ descendants in L_h;
- In layers i to $\alpha_{i+1} - 1$, each vertex has at most D^{M-i+1} descendants in L_h;
- In layer $\alpha_{i+1} - 1 + j$, $j \in [M-i+1]$, each vertex has at most $D^{M-i+1-j}$ descendants in L_h;
- In the remaining layers up to α (at most $\alpha - 1$ layers), each vertex has 1 descendant in L_h.

Since $\mathcal{U}$ has at most B vertices per layer, we can now bound the number of $(\mathcal{U}, \mathcal{B})$-safe vertices for each $r_i \in \mathcal{B}$ by:

$$\begin{aligned}
&B\left(\sum_{j=1}^{i-1} k'D^{M-j} + (\alpha_{i+1}-1)D^{M-i+1} + \sum_{j=1}^{M-i+1} D^{M-i+1-j} + \alpha\right) \\
&\leq B\left(\frac{k'D^M - k'D^{M-i+1}}{D-1} + \frac{D^i}{D-1}D^{M-i+1} + \frac{D^{M-i+1}}{D-1} + \frac{D^M}{D-1}\right) \\
&\leq \frac{B}{D-1}\left(k'D^M - k'D^{M-i+1} + D^{M+1} + D^{M-i+1} + D^M\right) \\
&\leq \frac{BD^M}{D-1}\left(k' + D + 1\right)
\end{aligned}$$

Since each tree has $k'D^M$ leaves, the fraction of $(\mathcal{U}, \mathcal{B})$-risky leaves for $r_i \in \mathcal{B}$ is at least

$$1 - \frac{\frac{BD^M}{D-1}(k'+D+1)}{k'D^M} = 1 - \frac{B}{k'} \cdot \frac{k'+D+1}{D-1} = 1 - \frac{B}{k'}\left(1 + \frac{k'+2}{D-1}\right)$$

Setting $D = d$, and summing over all trees in $\mathcal{B}$, the number of $(\mathcal{U}, \mathcal{B})$-risky vertices is at least

$$\left(1 - \frac{B}{k'}\left(1 + \frac{k'+2}{d-1}\right)\right)\frac{|\mathcal{B}|}{M}|L_h|. \qquad \square$$

By Lemma 6.16, the existence of d-ary $(M, k', (k'+2)/(d-1))$-good gadgets implies that there is an instance (G, s, w, B) with integrality gap

$$1 - \left(1 - \frac{1}{k}\right)^k + \frac{kB+2}{d-1} \leq 1 - e^{-1}\left(1 - \frac{1}{k}\right) + \frac{kB+2}{d-1}$$

The additive error factor is minimized when $k = \sqrt{(d-1)/B}$, which leads to an integrality gap of

$$1 - e^{-1} + O\left(\sqrt{\frac{B}{d}}\right)$$

This completes the proof of the theorem.

6.4 Improving the Standard LP with Hartke's Constraints

We have seen in Section 6.3 that the standard LP has an integrality gap of $1 - 1/e$. However, there is a PTAS for Tree-FF [ABZ17] that indirectly uses the standard LP. An interesting open question is whether we can obtain a PTAS, or at least a better approximation, by using a stronger LP relaxation.

In this section, we focus on the case of $B = 1$, that is, when only one vertex is protected per time step. Let $(T, s, w, 1)$ be an instance of Tree-FF. We assume that T is rooted at s and the vertices are partitioned into layers $L_0, L_1, \ldots, L_\lambda$, where, for each $i \in [\lambda]$, L_i contains all the vertices at depth i. For any vertex v, let P_v denote the (unique) path from s to v.

Hartke [Har04] suggested adding the following constraints to narrow down the integrality gap of the LP.

$$\sum_{u \in P_v \cup (T_v \cap L_j)} x_u \leq 1 \quad \forall v \in V, j \in [\lambda]$$

Intuitively, the constraints state that in an optimal solution, for any vertex $v \in V$, if an ancestor of v is protected, then no descendant of v should be protected. Conversely, if some descendant of v is protected, only one descendant should be protected per layer, and then no ancestor of v should be protected. We write this as an LP constraint by saying that the sum of the values of the variables corresponding to ancestors of v, plus, descendants of v in any given layer L_j, must be at most 1.

The new LP with these constraints is written down below:

Definition 6.21 (Hartke's LP for Tree-FF – LP-Hartke)**.**

$$\begin{aligned}
\max \quad & \sum_{v \in V} w_v y_v & & \\
\text{s.t.} \quad & \sum_{u \in P_v \cup (T_v \cap L_j)} x_u \leq 1 & \forall v \in V, j \in [\lambda] & \quad (1) \\
& y_v \leq \sum_{u \in P_v} x_u & \forall v \in V & \\
& x_v, y_v \in [0,1] & \forall v \in V &
\end{aligned}$$

We start by stating a property of the solution, which allows us to assign the y variables according to the x variables.

Observation 6.22. *Given a solution $x \in [0,1]^{V(T)}$ satisfying Constraints (1), the solution (x, y) defined by $y_v = \sum_{u \in P_v} x_u$ is feasible for* LP-Hartke *and at least as good as any other feasible solution (x, y').*

Proof. Since $y_v \leq \sum_{u \in P_v} x_u$, and $\sum_{u \in P_v} x_u \leq 1$ by Constraint (1), the observation follows. □

6.4.1 New Properties of Extreme Points

In their work, Finbow et al. [FM09] introduce some tractable instances of Tree-FF, in which the maximum degree in the tree is at most 3, and the fire starts at a vertex of degree 2. We denote these instances as *2-branching* (the important property in these instances is that, at any point, we have to choose between two vertices, i.e. one vertex is protected and the other one burns).

Let $(T, s, w, 1)$ be a 2-branching instance of Tree-FF, with $w > \mathbf{0}$. Even though we know how to solve these instances optimally in linear time, we show that the standard LP for Tree-FF is not integral on these instances. In contrast, LP-Hartke is integral in these instances, proving that it is a stronger relaxation. This section will be dedicated to proving these two points, which imply Theorem 6.7.

In order to prove that LP-Hartke is integral on these instances, we first present the following structural lemma.

Lemma 6.23. *Let (x, y) be an optimal extreme point for* LP-Hartke *on instance $(T, s, w, 1)$, $w > \mathbf{0}$. Suppose s has two children, denoted by a and b. Then $x_a, x_b \in \{0, 1\}$ and $x_a + x_b = 1$.*

Proof. Suppose that $x_a, x_b \in (0, 1)$. We will define two solutions (x', y') and (x'', y'') and derive that (x, y) can be written as a convex combination of (x', y') and (x'', y''), which is a contradiction with (x, y) being an extreme point.

First, we define (x', y') by setting $x'_b = 1, x'_a = 0$. For each vertex $v \in T_b$, we set $x'_v = 0$. For each vertex $v \in T_a$, we define $x'_v = x_v/(1 - x_a)$. We can show that x' is feasible for LP-Hartke: for each $v \in T_a$ and any layer L_j below v, we have that

$$\begin{aligned}\sum_{u \in P_v} x'_u + \sum_{u \in T_v \cap L_j} x'_u &= \frac{\left(\sum_{u \in P_v} x_u\right) - x_a}{(1 - x_a)} + \frac{\sum_{u \in T_v \cap L_j} x_u}{(1 - x_a)} \\ &\leq \frac{\left(\sum_{u \in P_v \cup (T_v \cap L_j)} x_u\right) - x_a}{(1 - x_a)} \\ &\leq 1,\end{aligned}$$

where the last inequality is due to the fact that x is feasible.

The constraint is trivially satisfied for every $v \in T_b$, because $x_v = 0$ for $v \in T_b$, so the constraint reduces to $x_b = \sum_{u \in P_v} x_u \leq 1$. For the root vertex $v = s$, we have that

$$\sum_{u \in L_j} x'_u = \frac{\left(\sum_{u \ \in (L_j \cap T_a)} x_u\right) - x_a}{(1 - x_a)} \leq 1.$$

We define (x'', y'') analogously: $x''_b = 0, x''_a = 1$. For each vertex $v \in T_a$, we set $x''_v = 0$, and for each $v \in T_b$, we define $x''_v = x_v/(1 - x_b)$. By symmetry, (x'', y'') is also feasible.

Observe that, for each $v \in T_b$, $y'_v = 1$ and for each $v \in T_a$, $y'_v = \frac{y_v - x_a}{1 - x_a}$. The value of x' is

$$\begin{aligned}w(T_b) + \sum_{v \in T_a} w_v y'_v &= w(T_b) + \frac{1}{1 - x_a} \sum_{v \in T_a} w_v (y_v - x_a) \\ &= w(T_b) + \frac{\sum_{v \in T_a} w_v y_v}{(1 - x_a)} - \frac{x_a}{(1 - x_a)} w(T_a)\end{aligned}$$

Similarly, the objective value of solution x'' is

$$w(T_a) + \frac{\sum_{v \in T_b} w_v y_v}{(1 - x_b)} - \frac{x_b}{(1 - x_b)} w(T_b)$$

Now, consider the convex combination

$$z = \frac{1 - x_a}{(2 - x_a - x_b)} x' + \frac{1 - x_b}{(2 - x_a - x_b)} x''$$

As a convex combination of x' and x'', z must be feasible. Its objective value is

$$\frac{1}{(2 - x_a - x_b)} \cdot \left((1 - x_a - x_b)(w(T_a) + w(T_b)) + \sum_{v \in V(T)} w_v y_v \right)$$

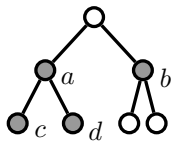

Figure 6.2: Instance with a non-integral extreme point for LP-FF. *Gray vertices*: $x_v = 1/2$; otherwise: $x_v = 0$.

Since $w(T_a) + w(T_b) = w(T)$, we have that $w(T_a) + w(T_b) > w^T y$ (since the solution cannot save both a and b). Combining this with the above, we get that the objective value of z is at least $w^T y$, with equality only if $x_a + x_b = 1$. Since y is optimal, we must have $x_a + x_b = 1$, and we therefore $z = (1 - x_a)x' + x_a x''$. It can be verified easily that $z_v = x_v$ for every $v \in V(T)$. □

We can now prove that extreme point solutions to LP-Hartke are integral for these instances.

Theorem 6.24. *Let (x, y) be an optimal extreme point for* LP-Hartke *on a 2-branching instance $(T, s, w, 1)$, $w > \mathbf{0}$.*

Then (x, y) is an integral solution.

Proof. We prove the theorem by induction on the height of the tree. If the tree has height 1, then Lemma 6.23 immediately shows that the solution is integral, since it is integral for the children of the root.

Now, for a tree T of height h, assuming the induction hypothesis for trees of height at most $h - 1$, let a, b be the two children of s. We apply Lemma 6.23 to conclude that $x_a, x_b \in \{0, 1\}$ and $x_a + x_b = 1$. W.l.o.g. let $x_a = 1$ and $x_b = 0$. By the constraints of LP-Hartke, $x_a = 1$ implies that $x_v = 0$ for all $v \in T_a$.

Now, let us focus on x restricted to T_b: it must be an extreme point solution for the instance restricted to T_b. Otherwise, we could either replace it by a better solution, contradicting optimality of x, or write it as a convex combination of solutions, which would imply the same for x and contradict the assumption that x is an extreme point.

Since x restricted to T_b is an extreme point solution to T_b, we can apply the induction hypothesis and conclude that x is integral for T_b, and hence for T.

This concludes the proof of the theorem. □

We present a simple 2-branching instance in Figure 6.2, as well as a solution for LP-FF on this instance that is optimal and an extreme point, but not integral. Furthermore, standard rounding on this instance has an expected value of 2.5, if vertices are protected with probabilities given by x.

Claim 6.25. *The solution (x, y) represented in Figure 6.2, with y defined according to Observation 6.22, is an extreme point solution of* LP-FF *on this instance.*

Proof. Suppose (for contradiction) that (x, y) is not an extreme point. Then, there are distinct solutions (x', y'), (x'', y'') and $\alpha \in (0, 1)$ such that $(x, y) = \alpha(x', y') + (1 - \alpha)(x'', y'')$. Since $y_c = 1$ and $y'_c, y''_c \leq 1$, then $y'_c = y''_c = 1$, and likewise, $y'_d = y''_d = 1$.

Combining that $x'_a + x'_c = y'_c = 1$ with $x'_a + x'_d = y'_d = 1$ and $x'_c + x'_d \leq 1$, we conclude that $x'_a \geq 1/2$. Similarly, we get that $x''_a \geq 1/2$, which implies that $x'_a = x''_a = 1/2$.

Similar reasoning using that $x'_a + x'_b \leq 1$ allows us to conclude that $x'_b = x''_b = 1/2$, and thus, $(x', y') = (x'', y'') = (x, y)$, which contradicts our assumption. □

6.4.2 Rounding 1/2-Integral Solutions

In this section, we consider 1/2-integral LP solutions. By Theorem 6.5, LP-FF is not strong enough to obtain a $(3/4 + \varepsilon)$-approximation algorithm, for any $\varepsilon > 0$. Here, we show a 5/6-approximation algorithm based on rounding LP-Hartke. The result is formalized in Theorem 6.8, which we recall below.

Theorem 6.8 (page 88). *Let $(T, s, w, 1)$ be an instance of Tree-FF, and let x be a half-integral extreme point solution for* LP-Hartke.

Then there is a valid strategy $\{S_i\}_{i\in[n]}$ that saves vertices with a weight of at least $5/6\, w^T x$.

We believe that the extreme points may be 1/2-integral in some interesting special cases. The rest of this section is dedicated to proving the theorem above.

Description of the algorithm Our algorithm considers the tree in top-down order, that is, it considers layers $L_1, \ldots, L_n$ in this order. We start with an empty strategy S. For each layer L_j, the algorithm chooses a vertex according to x, while giving preference to vertices whose ancestors are not in the support of x. Let $A_j = L_j \cap \text{supp}(x)$ be the set of vertices in L_j and the support of x. We partition A_j into (A'_j, A''_j), where $v \in A'_j$ if there is no ancestor of v in $\text{supp}(x)$, and $v \in A''_j$ if there is (exactly) one ancestor u of v in $\text{supp}(x)$. We choose S_j based on the following rules:

- If there is only one $v \in A_j$ such that is not saved by S so far, protect $S_j = \{v\}$. If there is no such vertex, protect nothing ($S_j = \emptyset$);
- If $A_j = A'_j$ or $A_j = A''_j$, pick a vertex u at random from A_j with uniform probability. Protect $S_j = \{u\}$;
- Otherwise, we have that $|A'_j| = |A''_j| = 1$. In this case, we protect $S_j = A'_j$ with probability 1/3; $S_j = A''_j$ otherwise.

Analysis Below, we argue that each vertex $v \in \text{supp}(x)$ is saved with probability at least $5/6\, y_v$. This immediately implies the theorem: consider a vertex $v' \in V$ such that $x_{v'} = 0$. If $y_{v'} = 0$, we are immediately done. Otherwise, consider the lowest ancestor v of v' in $\text{supp}(x)$. The probability that v' is saved is at least that of v, which is at least $5/6\, y_v = 5/6\, y_{v'}$.

Lemma 6.26. *Let $v \in \text{supp}(x)$. Then v is saved with probability at least $5/6\, y_v$.*

Proof. Let us first consider some simple cases. Consider a layer L_j such that $|A_j| = 1$; such a vertex $v \in A_j$ is saved with probability $1 \geq 5/6\, y_v$. Next, consider a layer L_j such that $|A'_j| = 2$; each vertex $v \in A'_j$ is saved with probability 1/2 and $y_v = 1/2$, so the probability of saving v is $y_v \geq (5/6)y_v$. The following claims cover the remaining two cases.

Claim 6.27. *Let L_j be a layer such that $|A'_j| = |A''_j| = 1$. Then the vertex $u \in A'_j$ is protected with probability $2/3 \geq 5/6\, y_u$ and vertex $v \in A''_j$ is saved with probability $5/6$.*

Proof. Let $v' \in A_{j'}$ be the ancestor of v in the support of x.

We prove the lemma by induction on j, the depth of the layer. In particular, it is sufficient to show, under the assumption that v' is protected with probability at least $1/2$, that u is protected with probability at least $2/3$ and v is saved with probability $5/6$.

Let us start by showing that the assumption follows directly from the induction hypothesis. For the base case, let L_j be the highest layer such that $|A'_j| = |A''_j| = 1$. This means that $|A'_{j'}| = 2$ (or $|A_{j'}| = 1$), and therefore the probability of v' being protected is at least $1/2$. For the induction step, induction hypothesis and the statement above also show that v' is protected with probability at least $1/2$ (specifically $1/2$ if $|A'_{j'}| = 2$ and at least $2/3$ if $|A'_{j'}| = 1$).

Notice that if v' is protected, then v is saved as a consequence, and therefore, u is the only vertex left in A_j to protect. Therefore, the event that u is not saved occurs only if v' is not protected and u itself is not protected. The probability of this happening is

$$\begin{aligned}\Pr[u \text{ not saved}] &= \Pr[v' \notin S_{j'}] \cdot \Pr[u \notin S_j \mid v' \notin S_{j'}] \\ &\leq \frac{1}{2} \cdot \frac{2}{3} = \frac{1}{3}\end{aligned}$$

We conclude that u is protected with probability $2/3$.

Consider now vertex v. For v not to be saved, then v' must not be protected and u must be protected in layer j. The probability that this happens is

$$\begin{aligned}\Pr[v \text{ not saved}] &= \Pr[v' \notin S_{j'}] \cdot \Pr[u \in S_j \mid v' \notin S_{j'}] \\ &\leq \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6}\end{aligned}$$

□

Claim 6.28. *Let L_j be a layer such that $A''_j = \{u, v\}$ (containing two vertices). Then each such vertex is saved with probability at least $5/6$.*

Proof. Let $u' \in A'_i$ and $v' \in A'_k$ be the ancestors of u and v in the support of x. If at least one of u' or v' is protected, then both u and v are saved, one of them as consequence of the ancestor being protected, and the other protected by the algorithm as the only choice left for L_j.

Let us first show that $\Pr[u' \notin S_i \cap v' \notin S_k] \leq 1/3$, and then show how it follows that u is saved with probability at least $5/6$. We consider several cases: If u' and v' are in the same layer ($i = k$), then the probability above is 0, since one of them is protected. If at least one of u', v' is in a layer that Claim 6.27 applies to ($|A'_i| = |A''_i| = 1$ or $|A'_k| = |A''_k| = 1$), then

$$\Pr[u' \notin S_i \cap v' \notin S_k] \leq \min\big(\Pr[u' \notin S_i], \Pr[v' \notin S_k]\big) \leq \frac{1}{3}$$

Finally, in the remaining case, u' and v' are independent from each other, and therefore,

$$\Pr[u' \notin S_i \cap v' \notin S_k] = \frac{1}{4}$$

Using the fact that $\Pr[u' \notin S_i \cap v' \notin S_k] \leq 1/3$, we have that

$$\begin{aligned}\Pr[u \text{ not saved}] &= \Pr[u \notin S_j \cap u' \notin S_i \cap v' \notin S_k] \\ &= \Pr[u' \notin S_i \cap v' \notin S_k] \Pr[u \notin S_j \mid u' \notin S_i \cap v' \notin S_k] \\ &\leq \frac{1}{3} \cdot \frac{1}{2} = \frac{1}{6}\end{aligned}$$

We conclude that u is saved with probability at least $5/6$ and, by symmetry, the same is true for v as well. □

By the arguments above, any vertex v is saved with probability at least $5/6\, y_v$, as desired. □

6.4.3 Ruling Out the Gap Instances in Section 6.3

In this section, we will show that the instances presented in Section 6.3 admit an approximation with ratio better than $(1 - 1/e)$ using LP-Hartke. To this end, we introduce the concept of *η-separable* LP solutions and show an improved rounding algorithm for solutions in this class (Theorem 6.9). The intuition for this result is that, in separable solutions, the rounding process creates dependencies in the lower layers, as many vertices are saved by protecting their ancestors. If many vertices in a layer are already saved, we may exclude them from the selection process, and thus scale up the LP value for unprotected vertices. This causes the probability that each vertex is protected to increase, and, under the right circumstances, increases the quality of the solution.

Definition 6.29. Let $\eta \in (0, 1)$, and (x, y) be a solution for LP-FF or LP-Hartke.

We say that a vertex v is *η-light* (for x) if $\sum_{u \in P_v \setminus \{v\}} x_u < \eta$, and that v is *η-heavy* otherwise. A layer L_j is η-light (resp. η-heavy) if all vertices in L_j are η-light (resp. η-heavy).

A solution (x, y) is said to be *η-separable* if all layers are η-light or η-heavy, that is, for every $j \in [n]$, either the vertices in L_j are all η-light, or all η-heavy.

Observation 6.30. *The LP solution presented in Section 6.3 is η-separable for all values of $\eta \in \{1/k, 2/k, \ldots, 1\}$.*

Observation 6.31. *If v is η-heavy, then so is any descendant of v. Similarly, if a layer L_j is η-heavy, so are all layers $L_{j'}$, $j' > j$.*

Theorem 6.32. *Let $(T, s, w, 1)$ be an instance of Tree-FF, and let x be an η-separable extreme point solution for* LP-Hartke*, for some $\eta \in (0, 1)$.*

Then there is an efficient algorithm that produces an integral solution of value

$$\big(1 - 1/e + f(\eta)\big) w^T y$$

where $f(\eta)$ is some function depending only on η.

The theorem above is a more specific restatement of Theorem 6.9, and we will spend the rest of the section proving it.

Algorithm By Observation 6.31, there is some i such that layers $L_1, \ldots, L_i$ are η-light, and layers $L_{i+1}, \ldots$ are η-heavy. Our algorithm proceeds in two phases, corresponding to η-light and η-heavy layers.

In the first phase, the algorithm protects, for $1 \leq j \leq i$, one vertex in L_j at random, such that the probability that v is chosen is x_v (as done by Cai et al. [CVY08]). Denote by $S' = \{S_j\}_{j \in [i]}$ the strategy obtained in this phase.

In the second phase, our algorithm performs randomized rounding conditioned on the solutions in the first phase. Specifically, when we process each η-heavy layer L_j, let $\tilde{L}_j \subseteq L_j$ be the collection of vertices that was not saved by S'. We sample one vertex $v \in \tilde{L}_j$ such that

$$\Pr\left[S_j = \{v\}\right] = \frac{x_v}{x\left(\tilde{L}_j\right)}.$$

Let $S'' = \{S_j\}_{j>i}$ be the strategy obtained in the second phase. The algorithm outputs $S = S' \cup S''$ as its strategy. This completes the description of the algorithm. We remark that running the algorithm in two phases simplifies the analysis, but is not strictly required. Sampling in every layer only from vertices that have not been saved cannot decrease the probability of a vertex being saved, compared to above. Therefore, the same performance guarantees apply.

In order to simplify the presentation, we will focus on the case when $\eta = 1/2$, while at the same making it clear where the analysis would change for different values. We will argue that each vertex $v \in T$ is saved with probability at least $(1 - 1/e + \delta)y_v$ for some universal constant $\delta > 0$ depending only on η. We denote by $P(v)$ the s-v-path in T. We will need the following simple observation that follows directly by standard probabilistic analysis (see Cai et al. [CVY08]).

Proposition 6.33. *For each vertex $v \in V(T)$, the probability that v is not saved is*

$$\Pr[v \text{ not saved}] \leq \prod_{u \in P(v)} (1 - x_u) \leq e^{-y_v}.$$

Taking advantage of the difference between $(1 - x)$ and e^{-x}, as well as the difference between $1 - e^{-x}$ and $(1 - e^{-1})x$, we can immediately get an improvement in the probability of saving certain vertices. The following lemma formalizes this.

Lemma 6.34. *Let $v \in T$, and $\delta \in [0, e^{-1})$.*

There are constants β_1, β_2 such that, if $y_v \leq \beta_1$ or there is some ancestor $u \in P(v)$ such that $x_u \geq \beta_2$, then

$$\Pr[v \text{ saved}] \geq (1 - 1/e + \delta)y_v.$$

For $\delta = 0.01$, we have $\beta_1 = 0.96$ and $\beta_2 = 0.22$

Proof. Since v is saved with probability at least $1 - e^{-y_v}$ (see Proposition 6.33), we are interested in finding values of y_v such that

$$1 - e^{-y_v} \geq (1 - 1/e + \delta)y_v$$
$$\frac{1 - e^{-y_v}}{y_v} \geq 1 - 1/e + \delta$$

Since $(1 - e^{-y_v})/y_v$ is continuous and decreasing on $(0, 1]$, from arbitrarily close to 1 (when y_v is close to 0) to $1 - 1/e$ (when $y_v = 1$), there must be a value of $\beta_1 \in (0, 1]$ such that for all $y_v \leq \beta_1$, $1 - e^{-y_v} \geq (1 - 1/e + \delta)y_v$.

Now, let us consider the case of $x_u > \beta_2$, for some ancestor $u \in P(v)$. The bound used in the standard analysis is only tight when the values are small, and can be improved when one of the values is relatively big.

$$\begin{aligned}
\Pr[v \text{ is saved}] &\geq 1 - \prod_{u' \in P(v)} (1 - x'_u) \\
&\geq 1 - (1 - x_u)e^{-(y_v - x_u)} \\
&\geq 1 - e^{x_u}(1 - x_u)e^{-y_v} \\
&\geq 1 - e^{x_u}(1 - x_u)e^{-1}y_v \\
&\geq \big(1 - e^{x_u}(1 - x_u)e^{-1}\big)y_v
\end{aligned}$$

$e^{x_u}(1 - x_u)$ is continuous and decreasing in $[0, 1]$, from 1 ($x_u = 0$) to 0 ($x_u = 1$). For some value of β_2 such that

$$e^{\beta_2}(1 - \beta_2)e^{-1} = e^{-1} - \delta,$$

we have that

$$\begin{aligned}
\Pr[v \text{ is saved}] &\geq \big(1 - e^{x_u}(1 - x_u)e^{-1}\big)y_v \\
&\geq \big(1 - e^{-1} + \delta\big)y_v
\end{aligned}$$

For $\delta = 0.01$, we can numerically determine the values $\beta_1 = 0.96$ and $\beta_2 = 0.22$ □

From now on, we only consider vertices $v \in T$ such that $y_v > \beta_1$ and $x_u < \beta_2$ for every $u \in P(v)$, for values of β_1, β_2 such that $\beta_1 - \eta - \beta_2 > 0$ is a positive constant.

Let $\mathcal{X}_1 \subseteq V$, $\mathcal{X}_2 \subseteq V \setminus \mathcal{X}_1$ be random variables representing the sets of vertices considered above that are saved by the strategy in the first and second phases, that is, by S' and S'', respectively. Unless specified, we always consider probabilities over the choice of S', S''. We can write the probability of v not being saved as

$$\Pr[v \notin \mathcal{X}_1 \cup \mathcal{X}_2] = \Pr[v \notin \mathcal{X}_1] \Pr[v \notin \mathcal{X}_2 \mid v \notin \mathcal{X}_1]$$

For any terminal v, let $P'(v)$ and $P''(v)$ be the sets of ancestors of v that are η-light and η-heavy respectively, i.e. ancestors in $P'(v)$ and $P''(v)$ considered by the algorithm in Phase 1 and 2 respectively. By Proposition 6.33, we can upper bound $\Pr[v \notin \mathcal{X}_1]$ by $e^{-x(P'(v))}$. In the rest of this section, we show that the second term above is upper bounded by $e^{-x(P''(v))}c$ for some $c < 1$, and therefore

$$\Pr[v \notin \mathcal{X}_1 \cup \mathcal{X}_2] \leq ce^{-x(P'(v)) - x(P''(v))} \leq ce^{-y_v}.$$

If the value of c satisfies $c = 1 - \delta e$, we get by simple algebraic manipulation that

$$\Pr[v \notin \mathcal{X}_1 \cup \mathcal{X}_2] \geq 1 - ce^{-y_v} \geq (1 - 1/e + \delta)y_v$$

The following lemma is the main technical tool we need in the analysis. We remark that this lemma encapsulates the additional power of LP-Hartke, and we do not distinguish between LP-Hartke and LP-FF in the remainder of the analysis.

Lemma 6.35. *Let x be an η-separable solution to* LP-Hartke, *$v \in V$ and L_j be a layer containing an η-heavy ancestor of v.*

Then

$$\mathrm{E}_{S'}\left[x\big(\tilde{L}_j\big) \mid v \notin \mathcal{X}_1\right] \leq \alpha$$

for $\alpha = 1 - \eta + (1 - e^{-\eta})$.

For $\eta = 1/2$, $\alpha \leq 0.9$.

Intuitively, this lemma says that any vertex that is not saved by the result of the first phase will have ancestors on "sparse" η-heavy layers (in expectation). We defer the proof of this lemma in order to show how to complete the analysis. Since the term that the lemma above bounds appears in the denominator of the probabilities used in the algorithm, we cannot simply lower bound the improvement. The following definition establishes some notation to refer to vertices where the lemma gives an improvement. We later show that this happens for each vertex with constant probability, and therefore, we can conclude that the probability that each vertex is selected increases.

For each vertex v, denote by $L(v)$ the layer to which vertex v belongs ($\tilde{L}(v)$ is defined analogously to $\tilde{L}_j$). Let $\lambda \in (0, 1]$ be the solution to $\lambda^{-3}\alpha = 1$.

Definition 6.36. Let $v \in V$ and $u \in P''(v)$ be an η-heavy ancestor of v,

For a fixed S', we say that vertex v is *partially protected* by S' if

$$\sum_{u \in P''(v)} x_u x\big(\tilde{L}(u)\big) \leq \lambda^{-1}\alpha x(P''(v))$$

Furthermore, we say that u is *good* for v and S' if

$$x\big(\tilde{L}(u)\big) \leq \lambda^{-2}\alpha$$

We denote as $\mathcal{X}' \subseteq V \setminus \mathcal{X}_1$ the subset of vertices that are partially protected by S', and as $\mathcal{S}_v^{\text{good}} \subseteq P''(v)$ the set of good ancestors of v.

Intuitively, we say $v \in V$ is partially protected if the average of $x\big(\tilde{L}(u)\big)$ over all η-heavy ancestors u, weighted by x_u is only slightly worse than α. We say that u is good for v if $x\big(\tilde{L}(u)\big)$ is not too worse than α (only by a factor λ^{-2}).

The next two claims show that each vertex that is not saved by S' is partially protected with probability at least $1 - \lambda$, and that, for each such vertex, the sum of x_u for all the good ancestors makes up a fraction of at least $1 - \lambda$ of the total. We exploit this to argue that, in this case, it is more likely that an ancestor of v is protected, and therefore, that v is saved.

Claim 6.37. *For any $v \in V$,*

$$\Pr_{S'}\left[v \in \mathcal{X}' \mid v \notin \mathcal{X}_1\right] \geq 1 - \lambda$$

Proof. By linearity of expectation and Lemma 6.35,

$$\mathrm{E}_{S'}\left[\sum_{u \in P''(v)} x_u x\big(\tilde{L}(u)\big) \;\middle|\; v \notin \mathcal{X}_1\right] = \sum_{u \in P''(v)} x_u \,\mathrm{E}_{S'}\left[x\big(\tilde{L}(u)\big) \;\middle|\; v \notin \mathcal{X}_1\right] \leq \alpha x(P''(v))$$

Using Markov's inequality,

$$\begin{aligned}
&\Pr_{S'}\left[\sum_{u\in P''(v)} x_u x\big(\tilde{L}(u)\big) \le \lambda^{-1}\alpha x(P''(v)) \;\middle|\; v \notin \mathcal{X}_1\right] \\
&\qquad = 1 - \Pr\left[\sum_{u\in P''(v)} x_u x\big(\tilde{L}(u)\big) > \lambda^{-1}\alpha x(P''(v)) \;\middle|\; v \notin \mathcal{X}_1\right] \\
&\qquad \ge 1 - \frac{\alpha x(P''(v))}{\lambda^{-1}\alpha x(P''(v))} \\
&\qquad = 1 - \lambda
\end{aligned}$$

□

Claim 6.38. *For any strategy S' and vertex $v \in \mathcal{X}'$,*

$$x\big(\mathcal{S}_v^{\text{good}}\big) \ge (1-\lambda)x(P''(v)).$$

Proof. Suppose (for contradiction) that the weighted fraction of good ancestors was less than $1-\lambda$, that is, $x\big(P''(v) \setminus \mathcal{S}_v^{\text{good}}\big) \ge \lambda$.

For each such $u \in P''(v) \setminus \mathcal{S}_v^{\text{good}}$, we have $x\big(\tilde{L}(u)\big) > \lambda^{-2}\alpha$. Then,

$$\sum_{u\in P''(v)} x_u x\big(\tilde{L}(u)\big) > \lambda^{-2}\alpha \sum_{u\in P''(v)\setminus\mathcal{S}_v^{\text{good}}} x_u \ge \lambda^{-2}\alpha\cdot\lambda = \lambda^{-1}\alpha.$$

But then v is not partially protected, which is a contradiction. □

Using the results so far, we can now provide a better bound for $\Pr[v \notin \mathcal{X}_2 \mid v \notin \mathcal{X}_1]$.

Lemma 6.39. *Let $\beta = \beta_1 - \eta - \beta_2$.*

$$\Pr[v \notin \mathcal{X}_2 \mid v \in \mathcal{X}_1] \le e^{-x(P''(v))}\delta',$$

where $\delta' = \lambda + (1-\lambda)e^{-(1-\lambda)^2\beta/\lambda} < 1$.

Proof. The proof follows mostly algebraically, and with only simple probabilistic techniques. We can rewrite the probability of a vertex $v \in V$ not being saved after the second phase.

$$\begin{aligned}
&\Pr[v \notin \mathcal{X}_2 \mid v \notin \mathcal{X}_1] \\
&= \Pr\left[v \in \mathcal{X}' \mid v \notin \mathcal{X}_1\right]\Pr\left[v \notin \mathcal{X}_2 \mid v \in \mathcal{X}'\right] + \Pr\left[v \notin \mathcal{X}' \mid v \notin \mathcal{X}_1\right]\Pr\left[v \notin \mathcal{X}_2 \mid v \notin \mathcal{X}'\right] \\
&\le (1-\lambda)\Pr\left[v \notin \mathcal{X}_2 \mid v \in \mathcal{X}'\right] + \lambda\cdot e^{-x(P''(v))}
\end{aligned}$$

The last inequality holds because $\Pr[v \notin \mathcal{X}_2 \mid v \notin \mathcal{X}']$ is at most $e^{-x(P''(v))}$ from Proposition 6.33.

It remains to provide a better upper bound for $\Pr[v \notin \mathcal{X}_2 \mid v \in \mathcal{X}']$. We express the term above as a sum over all the possible choices for S', such that that choice leads to $v \in \mathcal{X}'$. We remark that, for a good ancestor $u \in \mathcal{S}_v^{\text{good}}$, $x\big(\tilde{L}(u)\big) \le \lambda^{-2}\alpha = \lambda$.

$$
\begin{aligned}
&\Pr\left[v \notin \mathcal{X}_2 \mid v \in \mathcal{X}'\right] \\
&\quad= \sum_{Q' : v \in \mathcal{X}'} \Pr_{S'}\left[S' = Q' \;\middle|\; v \in \mathcal{X}'\right] \Pr_{V_2}\left[v \notin \mathcal{X}_2 \mid S' = Q'\right] \\
&\quad\le \sum_{Q' : v \in \mathcal{X}'} \Pr_{S'}\left[S' = Q' \;\middle|\; v \in \mathcal{X}'\right] \prod_{\text{bad } u \in P''(v)} (1 - x_v) \prod_{\text{good } u \in P''(v)} \left(1 - \frac{x_v}{\lambda}\right) \\
&\quad\le \sum_{Q' : v \in \mathcal{X}'} \Pr_{S'}\left[S' = Q' \;\middle|\; v \in \mathcal{X}'\right] \prod_{\text{bad } u \in P''(v)} e^{-x_v} \prod_{\text{good } u \in P''(v)} e^{-x_v/\lambda} \\
&\quad\le \sum_{Q' : v \in \mathcal{X}'} \Pr_{S'}\left[S' = Q' \;\middle|\; v \in \mathcal{X}'\right] e^{-\lambda x(P''(v))}\, e^{-(1-\lambda)x(P''(v))/\lambda} \qquad \text{(a)} \\
&\quad\le e^{-\lambda x(P''(v))}\, e^{-(1-\lambda)x(P''(v))/\lambda} \\
&\quad\le e^{-x(P''(v))} e^{-(1-\lambda)x(P''(v))\left(\lambda^{-1}-1\right)} \\
&\quad\le e^{-x(P''(v))} e^{-(1-\lambda)^2 x(P''(v))/\lambda}
\end{aligned}
$$

Step (a) follows by Claim 6.38 and the fact $e^{-x_v} \le e^{-x_v/\lambda}$. Combining both the expressions, we get:

$$
\begin{aligned}
\Pr\left[v \notin \mathcal{X}_2 \mid v \notin \mathcal{X}_1\right] &\le \left((1-\lambda)\Pr\left[v \notin \mathcal{X}_2 \mid v \in \mathcal{X}'\right] + \lambda\, e^{-x(P''(v))}\right) \\
&\le \left((1-\lambda)\, e^{-x(P''(v))} e^{-(1-\lambda)^2\beta/\lambda} + \lambda\, e^{-x(P''(v))}\right) \\
&\le e^{-x(P''(v))}\left(\lambda + (1-\lambda) e^{-(1-\lambda)^2\beta/\lambda}\right) \\
&\le e^{-x(P''(v))}\delta'
\end{aligned}
$$

By Lemma 6.34, we can assume that $x(P''(v)) \ge \beta > 0$. Since $\beta > 0$ and $\lambda < 1$, we conclude that $\delta' < 1$. □

Overall, the approximation factor we get is $(1 - e^{-1}\delta')$ for some universal constant $\delta' \in (0, 1)$.

Proof of Lemma 6.35

Let $v \in V$ and L_j be a layer containing an η-heavy ancestor of v. Recall that we want to prove that

$$\mathrm{E}_{S'}\left[x\left(\tilde{L}_j\right) \mid v \notin \mathcal{X}_1\right] \le \alpha$$

The proof will proceed by considering the probability that each $u \in L_j$ is in $\tilde{L}_j$, and then further group these vertices according to the least common ancestor with v. Using the constraints in LP-Hartke, we can then bound the value of x_u, for all descendants u of an ancestor of v, which will then allows to bound the sum.

We introduce the following two claims that will help us prove the lemma. As a reminder, we denote by $\mathcal{X}_1 \subseteq V$ the random variable representing the sets of vertices that are saved by S'. We will prove the lemma by considering the probability of a vertex $u \in L_j$ being in $\mathcal{X}_1$.

Claim 6.40. *Let $u \in L_j$ such that $u' := \mathrm{lca}(u, v)$ is η-light, and let $\varepsilon = y(u')$. Then,*

$$\Pr\left[u \notin \mathcal{X}_1 \mid v \notin \mathcal{X}_1\right] \leq e^{-(\eta-\varepsilon)}$$

Proof. Let $u' = \mathrm{lca}(u, v)$. It is equivalent to say $u \notin \mathcal{X}_1$ or $V(S') \cap P(u) = \emptyset$, and similarly, $v \notin \mathcal{X}_1$ if and only if $V(S') \cap P(v) = \emptyset$. Therefore, we can rewrite our probability as

$$\begin{aligned}\Pr\left[u \notin \mathcal{X}_1 \mid v \notin \mathcal{X}_1\right] &= \Pr\left[V(S') \cap P(u) = \emptyset \;\middle|\; V(S') \cap P(v) = \emptyset\right] \\ &= \Pr\left[V(S') \cap \big(P(u) \setminus P(v)\big) = \emptyset \;\middle|\; V(S') \cap P(v) = \emptyset\right]\end{aligned}$$

Using Proposition 6.33, and that $y(u) \geq \eta$, $y(u') = \varepsilon$, and $P(u) \setminus P(v) = P(u) \setminus P(u')$, we have that:

$$\begin{aligned}\Pr\left[u \notin \mathcal{X}_1 \mid v \notin \mathcal{X}_1\right] &= \Pr\left[V(S') \cap \big(P(u) \setminus P(v)\big) = \emptyset \;\middle|\; V(S') \cap P(v) = \emptyset\right] \\ &\leq \prod_{w \in P'(u) \setminus P'(u)} (1 - x_w) \\ &\leq e^{-(x(P'(u)) - x(P'(u')))} \\ &\leq e^{-(\eta-\varepsilon)}\end{aligned}$$

□

If the lowest common ancestor of u and v is not η-light, we simply bound the probability by

$$\Pr\left[u \notin \mathcal{X}_1 \mid v \notin \mathcal{X}_1\right] \leq 1$$

Claim 6.41. *Let $\varepsilon > 0$ and $L' \subseteq L_j$ be the set of vertices u such that $y(\mathrm{lca}(u, v)) \geq \varepsilon$. Then $x(L') \leq 1 - \varepsilon$.*

Proof. This claim follows from the constraints in LP-Hartke. Let w be the topmost ancestor of v such that $y_w \geq \varepsilon$. Then all vertices in L' must be descendants of w, so

$$\begin{aligned} & x(P(w)) + x(L') \leq 1 \\ \implies \quad & x(L') \leq 1 - x(P(w)) \leq 1 - \varepsilon \end{aligned}$$

□

With the above two tools, we are ready to proceed. First we break the expectation term into

$$\mathrm{E}_{S'}\left[x\big(\tilde{L}_j\big) \mid v \notin \mathcal{X}_1\right] = \sum_{u \in L_j} x_u \Pr\left[u \notin \mathcal{X}_1 \mid v \notin \mathcal{X}_1\right]$$

Let $v' \in L_j$ be the ancestor of v in layer L_j. We break down the sum further based on the lowest common ancestor of u and v. Using Claim 6.40, we get

$$\begin{aligned}\mathrm{E}_{S'}\left[x\big(\tilde{L}_j\big) \;\middle|\; v \notin \mathcal{X}_1\right] &= \sum_{\varepsilon \in [0,1)} \sum_{u \in L_j : y(\mathrm{lca}(u,v)) = \varepsilon} x_u \Pr\left[u \notin \mathcal{X}_1 \mid v \notin \mathcal{X}_1\right] \\ &\leq \sum_{\varepsilon \in [0,\eta)} e^{-(\eta-\varepsilon)} \sum_{u \in L_j : y(\mathrm{lca}(u,v)) = \varepsilon} x_u + \cdot \sum_{u \in L_j : y(\mathrm{lca}(u,v)) \geq \eta} x_u\end{aligned}$$

Notice that there is only a finite (actually, linear) number of values that $y(w)$ can take, for an ancestor w of v. We slightly abuse notation and write $\varepsilon \in [0, \eta)$ to represent iterating over all these possible values.

We now define $z_\varepsilon := \sum_{u \in L_j : y(\mathrm{lca}(u,v)) = \varepsilon} x_u$ in order to simplify the expression above. Similarly, we define $z_\eta := \sum_{u \in L_j : y(\mathrm{lca}(u,v)) \geq \eta} x_u$ to include the remaining cases. We can now write the above as

$$\sum_{\varepsilon \in [0,\eta)} e^{-(\eta-\varepsilon)} z_\varepsilon + z_\eta.$$

By Claim 6.41, we know that

$$\sum_{\varepsilon' \in [\varepsilon,\eta]} z_{\varepsilon'} \leq 1 - \varepsilon$$

We can express the sum above as the sum of the areas of rectangles of width z_ε and height $e^{-(\eta-\varepsilon)}$. Equivalently, we can write it as an integral over a piece-wise constant function $f(t)$, which takes the value $e^{-(\eta-\varepsilon)}$ for a length of z_ε. This function can further be made monotonically non-decreasing by a suitable ordering of these intervals. Claim 6.41 now implies that the interval in which the function is at least $e^{-(\eta-\varepsilon)}$ has length at most $1 - \varepsilon$.

Using all of this, we conclude that the integral is maximized when $f(t) = e^{-(\eta-t)}$ for $t \in [0, \eta)$ and $f(t) = 1$ for $t \in [\eta, 1]$, and therefore we have that:

$$\begin{aligned}
\mathrm{E}_{S'}\left[x\big(\tilde{L}_j\big) \mid v \notin \mathcal{X}_1\right] &\leq \int_0^1 f(t)dt \\
&\leq \int_0^\eta e^{-(\eta-t)}dt + \int_\eta^1 1\,dt \\
&= e^{-(\eta-t)}\Big|_0^\eta + t|_\eta^1 \\
&= (1-\eta) + (1-e^{-\eta})
\end{aligned}$$

6.4.4 Integrality Gap for LP-Hartke

In this section, we present, for any $\varepsilon > 0$, an instance where LP-Hartke has an integrality gap of $5/6 + \varepsilon$. Interestingly, these instances admit an optimal half-integral LP solution. We recall the formal result below.

Theorem 6.10 (page 89). *Let $\varepsilon > 0$. There is an instance $(T, s, \mathbf{1}, 1)$ of Tree-FF of size bounded by some function $f(\varepsilon)$, such that the integrality gap of* LP-Hartke *is at most $5/6 + \varepsilon$.*

Gadget The motivation of our construction is a simple gadget represented in Figure 6.3. In this instance, vertices are either *special* (colored gray) or *regular*. This gadget has three properties of interest:

- If we assign an LP-value of $x_v = 1/2$ to every special vertex, then this is a feasible LP solution that ensures $y_u = 1$ for every leaf u.
- For any strategy S that does not pick any vertex in the first layer of this gadget, at most 2 out of 3 leaves of the gadget are saved.
- Any pair of special vertices in the same layer do not have a common ancestor inside this gadget.

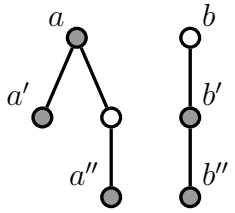

Figure 6.3: Gadget used to achieve an integrality gap of 5/6 for LP-Hartke. Special vertices are colored *gray*.

The basic idea is to make two interleaved copies of the gadget. In order to help with visualization, we will always label each vertex according to their "function" in the gadget of Figure 6.3. At first try, we could try using the following instance:

- **Layer 1:** two special vertices $a(1)$, $a(2)$ and two regular vertices $b(1)$, $b(2)$;
- **Layer 2:** two special vertices $a'(1)$ and $b'(1)$, descendants of $a(1)$ and $b(1)$ respectively;
- **Layer 3:** two special vertices $a'(2)$ and $b'(2)$, descendants of $a(2)$ and $b(2)$ respectively;
- **Layer 4:** two special vertices $a''(1)$ and $b''(1)$, descendants of $a(1)$ and $b'(1)$ respectively;
- **Layer 5:** two special vertices $a''(2)$ and $b''(2)$, descendants of $a(2)$ and $b'(2)$ respectively;

Notice that $a'(i)$ and $a''(i)$ descend from $a(i)$, but $b''(i)$ descends from $b'(i)$, which descends from $b(i)$ (as in the gadget of Figure 6.3). We skip in this description any regular vertices that are only used for layering, e.g. since $a'(2)$ is a descendant of $a(2)$ in layer 3, there must be a vertex in layer 2 which is a child of $a(2)$ and the parent of $a'(2)$. We will present a formal construction later. The construction described above is depicted in Figure 6.4 ($\alpha = 1$).

Unfortunately, this construction does not have the desired integrality gap, since there is a strategy that protects all the leaves: We can save all of the leaves from the first gadget (descendants of $a(1)$ and $b(1)$) by protecting only 2 vertices (say in layers 1 and 2), leaving us free to protect both $a''(2)$ and $b''(2)$ in layers 4 and 5.

To overcome this problem, we make copies of the vertices in the description above, with the number of copies increasing exponentially with depth. Specifically, we make α^{i-1} copies of the vertices in layer i, $i \in [5]$. At the same time, we add dummy layers (in which each vertex has a single child), which has the effect of increasing the budget comparably to the number of copies. This allows a strategy to be repeated for all copies, while preventing the use of "leftover" vertices from previous layers.

Consider the previous example: we can protect $a(1)$, $b'(1)$, $a'(2)$, the parent of $a''(2)$, and $b''(2)$, which saves all the leaves. Notice that we can save both $a''(2)$ and $b''(2)$ only because, in the previous layer, both $a''(1)$ and $b''(1)$ were already saved, and therefore we could save an additional vertex in the second copy of the gadget. However, by adding the copies as described above, there are now α^3 copies of (and layers corresponding to)

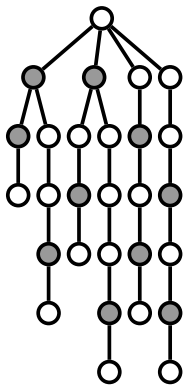

Figure 6.4: Example of the integrality gap instance for LP-Hartke, when $\alpha = 1$. Special vertices ($x_v = 1/2$) are colored *gray*.

$a''(1)$ and $b''(1)$, whereas there are α^4 copies of $a''(2)$ and $b''(2)$. Therefore, we can only save $\alpha^3 + \alpha^4$ of the total $2\alpha^4$ copies of $a''(2)$ and $b''(2)$, and hence the strategy above no longer saves all of the leaves.

Finally, we need to balance the instance, since there are a different number of copies of each of the original vertices. To do this, we add children to the leaves in the current construction in such a way that each of $a'(1)$, $a'(2)$, $a''(1)$, $b''(1)$, $a''(2)$, $b''(2)$ corresponds to exactly α^5 leaves. For instance, there will be α copies of $a'(1)$, so we give each of those copies α^4 descendants, such that, in total, we have α^5 leaves corresponding to $a'(1)$.

A simplified depiction of the construction for any value of α is presented in Figure 6.5.

We will now describe the construction formally and show that the integrality gap of LP-Hartke for this instance converges to 5/6 as the value of α increases.

Construction We denote the instance as T, and take s to be its root.

The first layer of this instance, L_1, contains 4 vertices: two special vertices, which we name $a(1)$ and $a(2)$, and two regular vertices, which we name $b(1)$ and $b(2)$. Recall the definition of spider from Section 6.3.2.

Let $\alpha = \lceil 1/\varepsilon \rceil$. The vertices $b(1)$ and $b(2)$ are the roots of two spiders. Specifically, the spider Z_1 rooted at $b(1)$ has α feet, with one foot per layer, in consecutive layers $L_2, \ldots, L_{\alpha+1}$. For each $j \in [\alpha]$, denote by $b'(1,j)$, the j^{th} foot of spider Z_1. The spider Z_2, rooted at $b(2)$, has α^2 feet, with one foot per layer, in layers $L_{\alpha+2}, \ldots, L_{\alpha^2+\alpha+1}$. For each $j \in [\alpha^2]$, denote by $b'(2,j)$, the j^{th} foot of spider Z_2. All the feet of spiders Z_1 and Z_2 are special vertices.

For each $j \in [\alpha]$, the vertex $b'(1,j)$ is also the root of spider $Z'_{1,j}$, with α^2 feet, lying in the α^2 consecutive layers $L_{2+\alpha+j\alpha^2}, \ldots, L_{1+\alpha+(j+1)\alpha^2}$ (one foot per layer). For $j' \in [\alpha^2]$, let $b''(1,j,j')$ denote the j'-th foot of spider $Z'_{1,j}$ that lies in layer $L_{1+\alpha+j\alpha^2+j'}$. Notice that we have α^3 such feet of these spiders $\left\{Z'_{1,j}\right\}_{j=1}^{\alpha}$ lying in layers $L_{2+\alpha+\alpha^2}, \ldots, L_{1+\alpha+\alpha^2+\alpha^3}$. Similarly, for each $j \in [\alpha^2]$, the vertex $b'(2,j)$ is the root of spider $Z'_{2,j}$ with α^2 feet, lying in consecutive layers $L_{2+\alpha+\alpha^3+j\alpha^2}, \ldots, L_{1+\alpha+\alpha^3+(j+1)\alpha^2}$. We denote by $b''(2,j,j')$ the j'-th foot of this spider.

The special vertex $a(1)$ is also the root of spider W_1 which has $\alpha + \alpha^3$ feet: The first α feet, denoted by $a'(1,j)$ for $j \in [\alpha]$, are aligned with the vertices $b'(1,j)$, i.e. for each

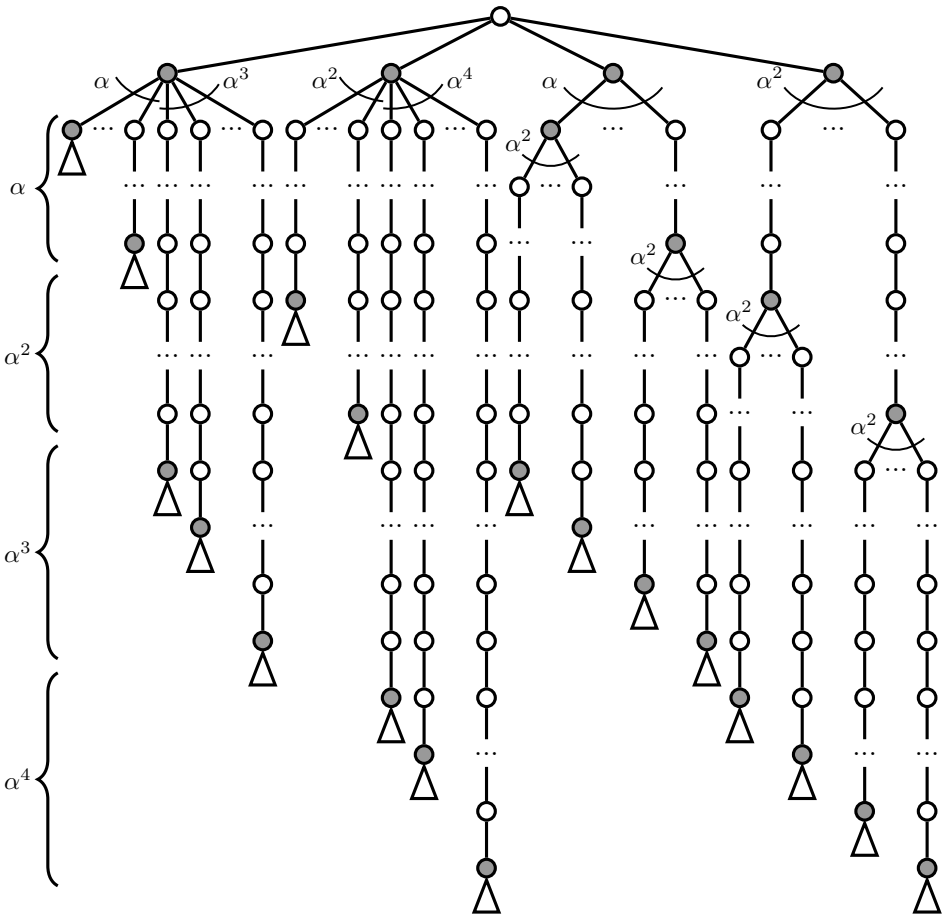

Figure 6.5: Simplified instance used to achieve integrality gap of 5/6 for LP-Hartke. The *labels* in the figure indicate the number of edges in that location, in terms of α. Special vertices ($x_v = 1/2$) are colored *gray*.

$j \in [\alpha]$, the foot $a'(1,j)$ of spider W_1 is in the same layer as the foot $b'(1,j)$ of Z_1. For each $j \in [\alpha], j' \in [\alpha^2]$, we also have a foot $a''(1,j,j')$ which is placed in the same layer as $b''(1,j,j')$. Similarly, the special vertex $a(2)$ is the root of spider W_2 with $\alpha^2 + \alpha^4$ feet. For $j \in [\alpha^2]$, spider W_2 has a foot $a'(2,j)$ placed in the same layer as $b'(2,j)$. For $j \in [\alpha^2], j' \in [\alpha^2]$, W_2 also has a foot $a''(2,j,j')$ in the layer of $b''(2,j,j')$. All the feet of both W_1 and W_2 are special vertices.

Finally, for $i \in \{1,2\}$, and $j \in [\alpha^i]$, each vertex $a'(i,j)$ has α^{5-i} children, which are leaves of the instance. For $j \in [\alpha], j' \in [\alpha^2]$, the vertices $b''(i,j,j')$, $a''(i,j,j')$ have α^{3-i} children each which are also leaves of the instance. We set the weight of all the leaves to 1, and of all other vertices to 0. Let $\mathcal{X}$ be the set of all leaves.

Proposition 6.42. *The instance satisfies $|\mathcal{X}| = 6\alpha^5$. Moreover, (i) the number of leaves in subtrees $T_{a(1)} \cup T_{b(1)}$ is $3\alpha^5$, and (ii) the number of leaves in subtrees $T_{a(2)} \cup T_{b(2)}$ is $3\alpha^5$.*

Proof. Each vertex $a'(1,j)$ has α^4 children, and there are α such vertices. Similarly, each vertex $a'(2,j)$ has α^3 children. There are α^2 such vertices. This accounts for $2\alpha^5$

terminals.

For $i \in \{1, 2\}$, each vertex $a''(i, j, j')$ has α^{3-i} children. There are α^{i+2} such vertices. This accounts for another $2\alpha^5$ terminals. Finally, there are α^{3-i} children of each $b''(i, j, j')$, and there are α^{2+i} such vertices. □

Fractional solution We define the solution x, with $x_v = 1/2$ for every special vertex v, and $x_v = 0$ otherwise. We will now show that this solution is feasible and saves every leaf. Notice that, in each layer, there are at most 2 special vertices, and therefore, $x(L_j) \leq 1$. The following lemma implies that the constraints in LP-Hartke are satisfied.

Lemma 6.43. *For each special vertex v, for each layer L_j below v, the set $L_j \cap T_v$ contains at most one special vertex.*

Proof. Each layer contains two special vertices, either $a'(i, j)$ and $b'(i', j')$ or $a''(i, j)$ and $b''(i', j')$. In any case, the least common ancestor of such two special vertices in the same layer is always the root s (since one vertex is in $T_{a(i)}$, while the other is in $T_{b(i)}$). This implies that, for any non-root vertex v, the set $L_j \cap T_v$ can contain at most one special vertex. □

Consider a vertex $v \in V$ and a layer L_j below v: if $x(P_v) = 0$, the constraint is satisfied, since $x(L_j) \leq 1$; if $x(P_v) = 1$, there are no special descendants of v by construction; if $x(P_v) = 1/2$, then applying Lemma 6.43 to the special ancestor of v implies that $L_j \cap T_v$ contains at most one special vertex, that is, $x(L_j \cap T_v) \leq 1/2$, and the constraint is satisfied.

Let us now consider the saved vertices: our construction guarantees that any path from root to leaf contains 2 special vertices: either $a(i)$ and $a'(i, j)$; $a(i)$ and $a''(i, j, j')$; or $b'(i, j)$ and $b''(i, j, j')$.

Integral solution We argue that no strategy can save more than $(1 + 1/\alpha)5\alpha^5$ terminals. Let $\mathcal{X}_1 = \mathcal{X} \cap (T_{a(1)} \cup T_{b(1)})$ and $\mathcal{X}_2 = \mathcal{X} \cap (T_{a(2)} \cup T_{b(2)})$, that is, the leaves that are descendants of $a(1)$, $b(1)$ or $a(2)$, $b(2)$, respectively.

The following lemma is the key to our analysis.

Lemma 6.44. *Any strategy S such that $S_1 \cap \{a(1), b(1)\} = \emptyset$ saves at most $\alpha^5(2 + 3/\alpha)$ vertices of $\mathcal{X}_1$.*

Proof. The lemma follows by counting the number of descendants of each vertex that are in $\mathcal{X}_1$, and then bounding the total number of such vertices that can be saved.

Let $Q' = \{a'(1, j)\}_{j \in [\alpha]} \cup \{b'(1, j)\}_{j \in [\alpha]}$ be the set of all special vertices of the form $a'(1, j)$, $b'(1, j)$, and $Q'' = \{a''(1, j, j')\}_{j \in [\alpha], j' \in [\alpha^2]} \cup \{b''(1, j, j')\}_{j \in [\alpha], j' \in [\alpha^2]}$ be the set of all special vertices of the form $a''(1, j, j')$, $b''(1, j, j')$. By construction, the vertices in Q' are contained in the first $1 + \alpha$ layers of T and have α^4 descendants in $\mathcal{X}$ each. Similarly, the vertices in Q'' are contained in the first $1+\alpha+\alpha^2+\alpha^3$ layers and have α^2 descendants in $\mathcal{X}$. All vertices that are not special (with the exception of $b(1)$) have degree exactly 1, and therefore the number of descendants of a vertex that are in $\mathcal{X}$ is determined by its descendants in Q' or Q''.

Using this information, we can upper bound the number of vertices in $\mathcal{X}_1$ that can be saved: in the first $1 + \alpha$ layers, the maximum number of vertices in $\mathcal{X}_1$ we can save

is α^4 per vertex protected, by protecting a vertex in Q' or one of its ancestors, since we do not protect $a(1)$ or $b(1)$ by assumption; in the following $\alpha^2 + \alpha^3$ layers, we can save at most α^2 vertices of $\mathcal{X}_1$ per layer, by protecting a vertex in Q'' or one of its ancestors; and then we can protect at most 1 leaf in the next layer; no consequent layer contains vertices in $(T_{a(1)} \cup T_{b(1)})$.

In total, the number of vertices that we can save, if we avoid $a(1)$ and $b(1)$, is at most:

$$(1+\alpha)\alpha^4 + (\alpha^2+\alpha^3)\alpha^2 + 1 = 2\alpha^5 + 2\alpha^4 + 1 \leq 2\alpha^5 + 3\alpha^4 \qquad \square$$

Lemma 6.45. *Any strategy S such that $S_1 \cap \{a(2), b(2)\} = \emptyset$ saves at most $\alpha^5(2 + 4/\alpha)$ vertices of $\mathcal{X}_2$.*

Proof. The proof of this lemma is similar to that of Lemma 6.44: in the first $1 + \alpha + \alpha^2$ layers, the maximum number of vertices in $\mathcal{X}_2$ we can save is α^3 per layer; in the following $\alpha^3 + \alpha^4$ layers, we can save at most α vertices in $\mathcal{X}_2$ per layer; in the final layer, we can only protect one vertex in $\mathcal{X}_2$.

In total, the number of vertices that we can save, if we avoid $a(2)$ and $b(2)$, is at most:

$$(1+\alpha+\alpha^2)\alpha^3 + (\alpha^3+\alpha^4)\alpha + 1 \leq 2\alpha^5 + 4\alpha^4 \qquad \square$$

Since the vertices $a(1)$, $a(2)$, $b(1)$, $b(2)$ are all in the first layer, it is only possible to protect one of them. Therefore, either Lemma 6.44 or Lemma 6.45 applies, and therefore the total number of vertices that can be saved by an integral solution is at most $5\alpha^5 + 4\alpha^4 \leq 5\alpha^5(1 + 1/\alpha)$.

We conclude that the integrality gap of this instance is $5\alpha^5(1+1/\alpha)/(6\alpha^5) \leq 5/6 + \varepsilon$.

CHAPTER 7

Firefighter Problem beyond Trees

The firefighter problem is interesting from the viewpoint of approximation: while it is easy to approximate on trees (there is a PTAS [ABZ17]), it is $n^{1-\varepsilon}$-inapproximable on general graphs (assuming P $\neq$ NP), where even a single mistake can cause almost all of the vertices to burn [ACH$^+$12].

On trees, Max-FF has been extensively studied [ACH$^+$12, CVY08, CC10, Har95, HL00, IKM11, KM10], with constant-factor approximation algorithms known [CVY08, HL00], and more recently, even a PTAS [ABZ17]. However, trees have a very simple structure: because they have no cycles, there is always a single (simple) path between the fire source and the any vertex. Furthermore, there needs to be no consideration about strategically "delaying" the fire, that is, protecting vertices along short paths, in order to gain more time to save a larger portion of the graph.

In this chapter, we study the approximability of Max-FF on bounded-treewidth graphs. This is a natural choice for this problem: on the one hand, even treewidth-2 graphs introduce the challenges of multiple paths and delays; on the other hand, it allows us to consider the intermediate case between the existence of a PTAS for tree instances, and $n^{1-\varepsilon}$-inapproximability on general graphs. Although results on trees are usually generalizable to bounded-treewidth graphs, the dynamic nature of the firefighter problem makes it hard to generalize known algorithms.

Related work Little is known about the problem in the bounded-treewidth setting. Cai et al. [CCV$^+$10] study the surviving rate of bounded-treewidth graphs. The surviving rate is defined as the fraction of vertices that can be saved, if the fire breaks out at a (uniformly) random vertex in the graph. They show that, if the budget is less than $w = \mathrm{tw}(G)$, there are graphs (such as the complete bipartite graph $K_{w,n-w}$) where only $O(w)$ vertices can be saved, independently of the starting point of the fire. On the other hand, if the budget is at least w, there is a strategy that ensures that the surviving rate is $1 - O(w^2 \log n/n)$.

For a more thorough description of the problem and related work for Max-FF on trees, we refer the reader to Chapter 6 and Section 6.1.

Our results Our goal is to describe the influence of treewidth on the approximation ratio. We present both lower and upper bounds for the best approximation ratio for different values of treewidth.

We start by presenting an $O(1)$-approximation algorithm for Max-FF on outerplanar graphs (see Section 2.1 for a definition). We show that outerplanar graphs have a nice structure, which allows us to use a modified greedy algorithm to obtain our result. Our techniques work for all graphs that satisfy a structural property which we denote k-completability, and we think our results may generalize to other classes of graphs, such

as treewidth-2 graphs.

Our second result confirms that even on bounded-treewidth graphs, Max-FF is a hard problem: we show that, on graphs of treewidth 5, it is NP-hard to $n^{1-\varepsilon}$-approximate. More generally, for any budget $B \geq 1$, there is no polynomial-time $n^{1-\varepsilon}$-approximation algorithm for Max-FF on graphs of treewidth $2B + 3$, unless P = NP. This implies that on graphs of treewidth w, a subpolynomial approximation is possible only if the budget is at least $w/2 - 1$. We find this to be an interesting result, as it provides a partial parallel to the result of Cai et al. [CCV+10], which shows that the surviving rate is sub-constant when $B < w$, and close to 1 otherwise (for constant w). To obtain our result, we start from the NP-hardness construction of Chlebíková and Chopin [CC14], and modify it to obtain a graph with low treewidth, where Max-FF cannot be approximated.

Organization The organization of this chapter is as follows: in Section 7.2, we present a constant-approximation algorithms for Max-FF on outerplanar graphs (and other k-completable graphs); in Section 7.3, we show that it is hard to $n^{1-\varepsilon}$-approximate Max-FF on graphs with treewidth 5.

7.1 Problem Definitions and Results

Problem 7.1: Firefighter Problem on Outerplanar Graphs (Outer-FF).

- Instance: (G, s, w, B), where
  - (G, s, w, B) is an instance of Max-FF;
  - G is outerplanar.
- Solution: a *valid strategy* $\{S_i\}_{i\in[n]}$ with budget B ($|S_i| \leq B$).
- Goal: maximize the weight of the set of vertices R that is *saved*, $w(R) = \sum_{v\in R} w_u$

Problem 7.2: *Cubic Monotone 1-in-3-SAT* (CM-1-in-3-SAT).

- Instance: $\phi = (\phi_1, \phi_2, \ldots, \phi_n)$ on variables $v_1, \ldots, v_n$, where
  - $v_1, v_2, \ldots, v_n$ are Boolean *variables*;
  - $\phi_i = (v_{i_1}, v_{i_2}, v_{i_3})$ is a *clause* on 3 positive literals, $i_1, i_2, i_3 \in [n]$, $i \in [n]$;
  - Each variable v_j appears in exactly 3 clauses, $j \in [n]$.
- Solution: an *assignment* $a \in \{0, 1\}^n$, such that, for each clause ϕ_i, $i \in [n]$, exactly one of the literals is assigned the value 1.
- Goal: Decide whether a feasible assignment exists
- *Remark:* CM-1-in-3-SAT is known to be NP-hard [MR01, PSS10].

Theorem 7.3. *There is an algorithm that computes an $O(1)$-approximation to the Outer-FF problem in polynomial time.*

Theorem 7.4. *Let $\varepsilon > 0$, $B \geq 1$ and ϕ be an instance of CM-1-in-3-SAT.*

There is a polynomial-time reduction from ϕ to an instance $\mathcal{I} = (G, s, \mathbf{1}, B)$ of Max-FF, where G has N vertices and treewidth $2B + 3$, such that:

- *If ϕ is CM-1-in-3-SAT-satisfiable, there is a valid strategy for $\mathcal{I}$ with budget B that saves at least $N^{1-\varepsilon}$ vertices.*

- *If ϕ is not CM-1-in-3-SAT-satisfiable, there is no valid strategy for $\mathcal{I}$ with budget B that saves more than N^{ε} vertices.*

Corollary 7.5. *It is NP-hard to approximate Max-FF on graphs of treewidth at least 5 (treewidth $2B+3$ for budget $B \geq 1$) to a factor better than $n^{1-\varepsilon}$, for any $\varepsilon > 0$.*

Corollary 7.6. *Unless $P = NP$, Max-FF on graphs of treewidth w can only be approximated (with factor below $n^{1-\varepsilon}$, $\varepsilon > 0$) if the budget is at least $B \geq w/2 - 1$.*

7.2 The Firefighter Problem on Outerplanar Graphs

In this section, we study Max-FF on outerplanar graphs, which remain simple in structure, but break all of the important properties of trees. Our results show that in graphs with a certain structure, which we name k-completable, we can think of the solution in terms of small vertex cuts (of size at most k). We show that, in that case, we can save a constant fraction of the optimum by repeatedly protecting the k vertices that separate the most vertices from the fire source. As a consequence, we obtain an $O(1)$-approximation algorithm for outerplanar graphs, which are 3-completable. We believe that k-completable graphs are useful generalization of trees for Max-FF, but we focus on outerplanar graphs in this thesis.

We will start by formally introducing some definitions that we will use throught this section, then introduce k-completable graphs and show that outerplanar graphs are 3-completable, and finally present an algorithm to approximate the problem in k-completable graphs. The following theorems summarize our results (see Section 7.2.2 for definitions).

Theorem 7.7. *The class of outerplanar graphs is 3-completable.*

Theorem 7.8. *Let (G, s, w, B) be an instance of the problem and $k \in \mathbb{Z}^+$.*

Then there is an algorithm that runs in time $O(n^{k+1}\, m)$ and finds a solution that is:

(1) a $(4k+2)$-approximation if G is k-weak-completable and $B \geq k$;

(2) a $(4k^2+4k+2)$-approximation if G is k-completable (and $B < k$);

Combining these results implies Theorem 7.3.

7.2.1 Delays and Influence

In this section, we will introduce some notation that is heavily used in the explanation of the algorithm. We will start by defining delayed vertices, and then move on to define the influence relationship. We will finally prove some properties and connections between the two terms.

We denote by $d(u, v)$ the shortest-path distance between vertices u and v, and denote $d(v) := d(s, v)$ for convenience.

Definition 7.9. Let $u, v \in V$.

We say v is a *delay vertex* (or simply a delay) if, for some $u \in V \setminus \{v\}$, v is contained in an s-u-shortest-path. For such vertices $u \in V \setminus \{v\}$, we say u is *delayed* by v or that v is a delay (vertex) for u.

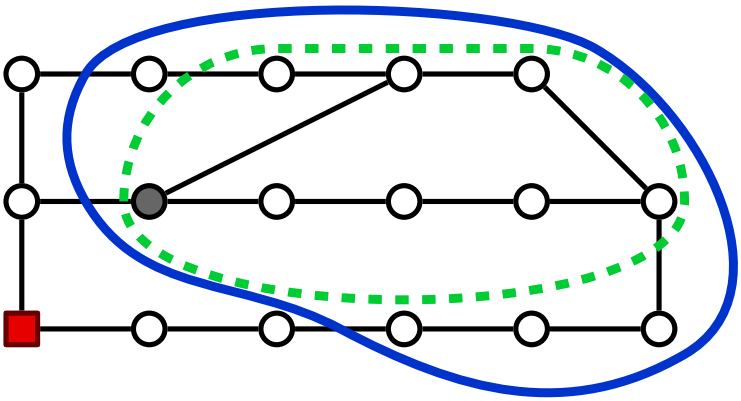

Figure 7.1: Example of delay and influence. The fire source is marked as a filled square (red), a vertex v is marked as a filled circle. The vertices delayed and influenced by v are the vertices inside the dashed (green) and full (blue) curve, respectively.

We now introduce the notion of influence.

Definition 7.10. Let $u, v \in V$.

We say u is *influenced* by v if $u = v$ or if $d(u) > d(v)$ and there is a path P from v to u such that for all $u' \in V(P) \setminus v$, $d(u') > d(v)$.

In other words, a vertex u is influenced by v if the fire can spread to u, even if we protect all other vertices at the distance $d(v)$ from the source. We will now show that if a vertex u is delayed by a vertex v, it is also influenced by v. This allows us to consider only the notion of influence in the analysis of our algorithm. See Figure 7.1 for an example of the concepts of delay and influence.

Lemma 7.11. *Let $u, v \in V$. If u is delayed by v, then u is influenced by v.*

Proof. Since v is a delay for u, there must be a shortest path from s to u containing v. Let $P = (v_0, v_1, \ldots, v_j = v, \ldots, v_k = u)$ be such a shortest path.

Since P is a shortest path, we have that $d(v_i) = i$ for all $i \in [k]$. Indeed, if $d(v_i) < i$ we could replace $(v_0, v_1, \ldots, v_i)$ by a shorter path between s and v_i, a contradiction; and since $(v_0, v_1, \ldots, v_i)$ has length i, $d(v_i) \leq i$, which together imply that $d(v_i) = i$.

Now, the subpath $(v = v_j, \ldots, v_k = u)$ satisfies the constraints in Definition 7.10: it is a v-u-path, and $d(v_\ell) = \ell > j$ for $\ell > j$, which finishes the proof. □

Finally, we will prove that both the delay relation and the influence relation are partial orders, and thus transitive.

Lemma 7.12. *Let $u, v \in V$ and denote by $u \succcurlyeq_D v$ the relation "u is delayed by v" and by $u \succcurlyeq_I v$ the relation "u is influenced by v".*

Then $\succcurlyeq_D$ and $\succcurlyeq_I$ are partial orders, that is, they are reflexive, antisymmetric and transitive.

Proof. We start by proving the statement for the delay relation, $\succcurlyeq_D$. It is reflexive, since v is trivially in every (shortest) path from s to v; it is antisymmetric, since $u \succcurlyeq_D v$

implies that $u = v$ or $d(v) < d(u)$, as v is in a shortest path from s to u. This means that $u \succcurlyeq_D v$, $v \succcurlyeq_D u$ either implies $u = v$ or leads to $d(u) > d(v)$ and $d(v) > d(u)$, which is a contradiction. It is transitive because if $w \succcurlyeq_D u$ and $u \succcurlyeq_D v$, then we can obtain a shortest path from s to w by appending any shortest path from s to u to any shortest path from u to w. Since $u \succcurlyeq_D v$, there is a shortest path from s to u containing v, which means that we can construct a shortest path from s to w containing v too.

As to the influence relation, it is reflexive by definition, and antisymmetric, since for $u \neq v$, either $d(u) > d(v)$ or $d(v) > d(u)$. To prove that it is transitive, take $v, u, w \in V$ such that $w \succcurlyeq_I u$, $u \succcurlyeq_I v$. By definition, $d(w) > d(u) > d(v)$, and there are paths P' from u to w, P'' from v to u such that for all $u' \in V(P') \setminus u$, $d(u') > d(u) > d(v)$, and for all $u' \in V(P'') \setminus v$, $d(u') > d(v)$. Concatenating P' and P'' results in a path from v to w satisfying the conditions, thus finishing the proof. □

7.2.2 k-Completable Graphs

In this section, we will define k-completable graphs and show that outerplanar graphs are 3-completable. We start with a weaker (but more general) definition, of k-weak-completable graphs, and then move on to the stronger definition.

Throughout this section, we will refer to small sets of vertices or vertex cuts as *squads*. Formally, a *squad* is simply a set of vertices $C \subseteq V$, to be interpreted as a vertex cut separating s from some vertices in the graph. We also say that a squad C *saves* a vertex v if s and v are in different connected components of $G - C$.

Definition 7.13. A graph G is *k-weak-completable* if, for any fire source $s \in V$, and any vertex $v \in V$, there is a squad $C_v \subseteq V$, such that:

(1) $v \in C_v$,

(2) $|C_v| \leq k$,

(3) $d(u) \geq d(v)$ for all $u \in C_v$, and

(4) every vertex delayed by v is saved by C_v, that is, it is separated from s in $G - C_v$.

A class of graphs $\mathcal{G}$ is k-weak-completable if for all $G \in \mathcal{G}$, G is k-weak-completable.

Intuitively, a graph G is k-weak-completable if for any vertex $v \in V$, there is a small vertex cut C_v separating a fire source from the vertices delayed by v. This allows us to consider the solution as the union of small cuts, while losing a factor of $O(k)$ in the approximation. For this purpose, we impose that the vertices in a set C_v must be at least as far from the fire source as v itself.

However, if the budget is small, this is not sufficient. In that case, we need extra constraints on vertices v close to the fire source, stating that it must be possible to protect all the vertices in C_v using a valid strategy.

This motivates the following definition:

Definition 7.14. Let G be a graph and $C \subseteq V(G)$ be a set of its vertices.

We say that C is *dispersed* at distance a if, for some ordering $v_1, v_2, \ldots, v_{|C|}$ of the vertices of C, we have $d(v_i) \geq a + i$ for every $i \in [|C|]$.

If G is k-weak-completable and all the squads C_v are dispersed at distance $d(v)$ when v is close to the fire source, we call the graph *k-completable*.

Definition 7.15. A graph G is *k-completable* if it is k-weak-completable and, for any fire source $s \in V$, satisfies the following constraints: for any $v \in V$ such that $d(v) \leq k$, there are $2k$ squads $C_1, C_2, \ldots, C_{2k} \subseteq V$ such that

(1) $|C_i| \leq k$ and C_i is dispersed at distance $d(v)$, for $i \in [2k]$;

(2) Every vertex delayed by v is saved by one of the squads C_i, $i \in [2k]$, that is, for every vertex u delayed by v, there is $i \in [2k]$ such that C_i separates s from u.

A class of graphs $\mathcal{G}$ is k-completable if for all $G \in \mathcal{G}$, G is k-completable.

We will now prove that the class of outerplanar graphs is 3-completable. We start by proving that outerplanar graphs are 3-weak-completable, and then prove Theorem 7.7, which we recall below. Although outerplanar graphs are k-completable for constant k and also have bounded treewidth, it is not clear whether these two concepts are related. We defer the discussion of this question to Chapter 8.

Lemma 7.16. *The class of outerplanar graphs is 3-weak-completable.*

Theorem 7.7 (page 119). *The class of outerplanar graphs is 3-completable.*

Before we continue with the proofs, we need to introduce the definition of a tour, as well as prove some results about tours that will be useful later.

Definition 7.17. A *tour* $T \subseteq G$ of an outerplanar graph G is a minimum length closed walk that goes through every vertex, goes through every edge of the outer face (at most twice), and uses no chords.

A *subtour* is a closed subwalk of a tour, that is, a closed walk that only goes through vertices and edges of the outer face.

A *slice* $A \subseteq T$ of a tour is a (continuous) subwalk of the tour, and can be thought as either the sequence or set of vertices of A, depending on context.

When considering a tour T as a sequence of vertices, we refer to each time that v appears in the tour as an *appearance* of v in T.

When using the concept of tour, we generally specify a special vertex s (the fire source). In that case, we consider (one of the appearances of) s to be the first vertex of a tour. If a special vertex s is specified, we assume that a slice does not contain s.

Lemma 7.18. *Let G be an outerplanar graph. Then G has a tour.*

Proof. The proof follows from basic principles presented in the book by Diestel [Die12, Chapter 4], but is not explicitly stated. Therefore, we present a short proof of the lemma.

Let f be the outer face (for an arbitrary embedding), and let $H = G[f]$ be the subgraph of G induced by the face f. By [Die12, Lemma 4.2.2], we can partition the edges of H into edges that belong to cycles and bridge edges, which are not on any cycle. Since all edges of H must be in the frontier of f, we can further partition the edges into cycles according to the other face whose frontier they belong to. This implies that H is the union of edge-disjoint cycles and bridge edges.

Doubling the bridge edges, we obtain a graph where the degree of every vertex is even (each cycle and bridge edge now contributes degree 2), and therefore there is an Eulerian circuit, which is a tour, since it visits every vertex and edge at most twice. □

Lemma 7.19. *Let G be an outerplanar graph, $s \in V(G)$ be a special vertex, T be a tour of G, and $u, v \in T \setminus \{s\}$.*

(1) Let A_u be the smallest slice containing all the appearances of u. Then $\{u\}$ is a vertex cut separating $T \setminus A_u$ from A_u.

(2) Let $u, v \in V$ be such that uv is either a chord of G, or can be added as a chord of G without changing the embedding. Let A_{uv} be the smallest slice containing all the appearances of u and v. Then $\{u, v\}$ is a vertex cut separating $T \setminus A_{uv}$ from A_{uv}.

Proof. We start by proving Point (1), and then show how it implies Point (2) by a simple transformation.

Let T' be the subtour obtained by connecting s to the first appearance of u along T, and then connecting the last appearance of u to s along T. In other words, T' is obtained by closing $T \setminus A_u$ by unifying two appearances of u. Notice that A_u is also a subtour, as it starts and ends in u.

Now, we need to prove that T' and A_u are almost disjoint. In fact, we will prove that the only vertex in T' and A_u is u. Assume for contradiction that there exists a vertex w in T' and A_u. Take w that is minimally close to u through T' and A_u, that is, such that there is no other vertex closer to u simultaneously through T' and A_u. Now let H be a subgraph of G formed by the union of two simple shortest paths connecting w to u in T' and in A_u. H must be a cycle, since, by minimality, the two paths cannot share any vertices.

Since all of the edges of H are in a cycle of T, they must also be edges in cycles of A_u or T'. However, this implies that each edge of H should be part of three faces: the outer face, the face defined by H, and a face in A_u (or T'), which is a contradiction.

Now, since the outer face of G does not contain any chords, all the edges in $E(G)$ are either in T or in the inner faces of $G[T]$. These faces are identified by cycles, so the faces of $G[T']$ and $G[A_u]$ must be disjoint, since T' and A_u only share one single vertex. Therefore, no edge can cross from a vertex in T' to a vertex in A_u, as it would not be contained in T or any of the inner faces of T, and thus $\{u\}$ is a vertex cut.

Assuming that the first point is true, we can easily prove the second point. Notice that, since outerplanar graphs have a forbidden minor characterization [Die12], then edge-contraction must preserve outerplanarity (a minor of the contracted graph is also a minor of the original graph). Therefore, we can contract the chord $e = uv$ in $G \cup \{uv\}$, obtaining $G' = G/e$. Let v_e be the vertex obtained by contracting e. The statement is immediately implied by the above, since cutting $\{u, v\}$ in G is equivalent to cutting v_e in G'. □

We are now ready to finish the proof that outerplanar graphs are 3-completable.

Proof of Lemma 7.16. Let T be a tour of G, and let A be the set of all vertices influenced by v (including v). We will now prove that A is a slice of T. Notice that A must be connected, since all vertices in A are connected to v via paths in A. Now, if A is not a slice, since it is connected, it must be composed of multiple slices connected by chords. Let $c = u_1u_2$ be one of the chords connecting disjoint slices of A, and A_c be the smallest slice of T containing both endpoints of c. By choice of c, $A_c \not\subseteq A$. However, by Lemma 7.19, $\{u_1, u_2\}$ is a cut separating $s \notin A_c$ from A_c. For every $w \in A_c$, any

path going from s to w must go through either u_1 or u_2, and thus we must have that $d(w) > \min(d(u_1), d(u_2)) > d(v)$, which implies that w is influenced by v. Since all the vertices in A_c are influenced by v, A_c must be a subset of A, which is a contradiction. Therefore, A must be a slice.

Let u_1, u_2 be the vertices immediately before and after $A \setminus \{v\}$ in T. We remark that it is possible that $u_1 = u_2$, or even $u_1 = v$ or $u_2 = v$, since we exclude v before taking the neighbors of A. Given the above, we define the squad for v as $C_v = \{u_1, u_2, v\}$.

We will now prove that any edge $uw \in E$ with $u \notin A$ to $w \in A \setminus v$, satisfies $u \in C_v$ and $d(u) = d(v)$. In other words, all of the edges into A (excluding v) come from C_v, and therefore C_v saves all vertices in A. The statement also holds (trivially) when $u = v$.

First, we show that, for such an edge, $d(u) = d(v)$ and $d(w) = d(v) + 1$. In fact, $uw \in E$ implies that $d(w) \leq d(u) + 1$, and $d(u) \leq d(v)$ (otherwise $u \in A$). Therefore, $d(w) \leq d(v)+1$, and since $w \in A \setminus \{v\}$, $d(w) \geq d(v)+1$, which means that $d(w) = d(v)+1$ and $d(u) = d(v)$.

It remains to prove that $u \in C_v$. If $uw \in T$, that follows directly, since A is a slice of T and therefore has at most two neighbors, both of which are in C_v. If uw is a chord, then it defines a slice A_{uw}. By Lemma 7.19, $\{u, w\}$ separates s from A_{uw}, and therefore, every path from s to A_{uw} must go through u or w. Since $d(w) > d(v)$, $d(u) = d(v)$, this further implies that all of the vertices $a \in A_{uw} \setminus u$ satisfy $d(a) > d(v)$, and thus, $(A_{uw} \setminus \{u\}) \subseteq A$. This makes u a neighbor of A, which implies that $u \in C_v$.

The proof of the lemma follows easily from the above. First, there are edges $u_1w \in T$, $u_2w \in T$, which implies by the above that $d(u_1) = d(u_2) = d(v)$. On the other hand, since every edge into $A \setminus \{v\}$ has an endpoint in C_v, then C_v is a cut, concluding the proof. □

Proof of Theorem 7.7. We know already from Lemma 7.16 that outerplanar graphs are 3-weak-completable. All that remains is to prove the additional conditions of Definition 7.15. Let $v \in V$ be a vertex satisfying $d(v) \leq k$, and take $C_v = \{u_1, u_2, v\}$ and A as defined in the proof of Lemma 7.16.

Now, assuming that A is the slice starting at the neighbor of u_1 along T and ending at the neighbor of u_2 along T, take $u_1' \in A$ to be the farthest neighbor of u_1 along A (before v) satisfying $d(u_1') = d(v) + 1$. Similarly, take $u_2' \in A$ to be the farthest neighbor of u_2 along A (after v) satisfying $d(u_2') = d(v) + 1$.

We can now define the squads C_i that satisfy the definition. In general, we have $C_1 = \{u_1, u_1'\}$, $C_2 = \{u_2, u_2'\}$, $C_3 = \{v, u_1'\}$, $C_4 = \{v, u_2'\}$. If $u_1 = v$ (resp. $u_2 = v$), we remove the squad C_1 (resp. C_2), and remove u_1' (resp. u_2') from C_3. If $C_3 = C_4 = \{v\}$, keep only one set.

To complete the proof, we just need to check that the squads satisfy the definition. Trivially, $|C_i| \leq 3$ for $i \in [3]$. Regarding every vertex delayed by v being saved by one of these squads, we claim that the subslices of A between u_1 and u_1'; u_1' and v; v and u_2'; u_2' and u_2 are covered by C_1, C_3, C_4, C_2, respectively. This can be shown by applying Lemma 7.19 to the respective pairs of vertices, since u_1u_1', u_2u_2' are edges; similarly, edges vu_1', vu_2' could be added, since C_1, C_2 are vertex cuts, which means that no edge can cross these. Finally, we know that $d(u_1) = d(v)$, $d(u_1') = d(v) + 1$, and therefore C_1, C_3 can be protected in the first 2 time steps. The same reasoning applies for C_2 and C_4. □

7.2.3 A Greedy Algorithm for k-Completable Graphs

In this section, we present an algorithm that outputs an $O(k^2)$-approximation for Max-FF on k-completable graphs, with any budget. We also present an $O(k)$-approximation algorithm for k-weak-completable graphs, when the budget is at least k.

We show that, for k-completable or k-weak-completable graphs, we can cover all of the vertices saved by an optimum solution just by protecting squads of size k. Furthermore, we show that the number of squads we need is relatively low: there is a set of squads that saves the same vertices as the optimum solution, and which needs only c squads for every k time steps, where c is a constant depending only on k. By using an adaptation of the greedy algorithm for trees, this is sufficient to show a $O(c)$-approximation, since for every k time steps, we can protect at least one squad out of c, implying that we save a fraction of at least $1/O(c)$ of the vertices saved by the optimum in the same time steps.

In order to present these results, we need two technical ideas. First, we need to reduce the optimal solution to a strategy in which vertices are not delay vertices to others in the strategy. This is not possible in general, as delayed vertices may be needed in order to save all the vertices saved by the optimum solution. Fortunately, the definitions of completability imply that there are small squads that save all of the vertices influenced by the optimum solution. Since influence is transitive, there is a subset of vertices of the optimum solution that are not delayed by others, and influence all of the vertices saved by the solution, which is sufficient for our analysis.

Afterwards, we show an analogue of the greedy algorithm on trees, and analyze it with respect to a collection of groups of at most c squads. If we know that protecting all of the vertices in all of the groups is at least as good as an optimum solution, but we are allowed to protect only one squad per group, we show that the greedy algorithm obtains an $O(c)$ approximation. Finally, we apply these results to obtain approximation algorithms for k-completable graphs ($c = O(k^2)$) and k-weak-completable graphs, when $B \geq k$ ($c = O(k)$).

Let (G, s, w, B) be an instance of the problem. In this section, we analyze a simple greedy algorithm, which starts with an empty strategy, and greedily finds the best k vertices to add to the strategy. In order to formally describe the algorithm, we need the following definition:

Definition 7.20. Let (G, s, w, B) be an instance of the problem and let $U \subseteq V(G)$ be a set of vertices.

We say U is *schedulable* if the vertices in U can be protected with budget B, that is, U is schedulable if for all $i \in [n]$,

$$\left|\{v \in U : d(v) \leq i\}\right| \leq iB$$

If U is schedulable, we can find a valid strategy S with budget B that protects all of the vertices in U, by protecting each vertex $v \in U$ on the latest time step that is at most $d(v)$ and has budget available.

Using this definition, the algorithm can be described as follows: start with an empty vertex set U; until the fire stops spreading, repeatedly find a squad Q of at most k vertices such that $U \cup Q$ is schedulable, maximizing the total weight of the vertices saved by Q;

add these vertices to U. We denote this algorithm as GreedyFF; see Algorithm B.1 for a formal description.

Lemma 7.21. *Let S be a strategy and let $\hat{S}$ be the substrategy of S containing only the minimal elements of S with respect to the delay relation, that is, only those vertices of S that are not delayed by any other $v \in V(S)$.*

Then all vertices delayed by a vertex in S (including the vertices saved by S) are also delayed by a vertex in $\hat{S}$.

Proof. Let $u \in V$ be any vertex delayed by $v \in V(S)$. If $v \in V(\hat{S})$, the claim follows. Otherwise, since the delay relation is a partial order (Lemma 7.12), there must be a minimal element v' of $V(S)$ such that v is delayed by v'. The set of minimal elements is precisely $V(\hat{S})$, so v is delayed by a vertex in $\hat{S}$, and by transitivity, so is u. □

Lemma 7.22. *Let $c, k \in \mathbb{Z}^+$, and let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_\ell$ be sets of at most c squads, each of size at most k, and let T be defined as the set of vertices saved by any of the squads in $\bigcup_{i\in[\ell]} \mathcal{C}_i$. Additionally, assume that, for any $C \in \mathcal{C}_i$ and any schedulable set of vertices U of size $|U| \leq (i-1)k$, $C \cup U$ is schedulable.*

Then, GreedyFF *outputs a valid strategy with budget B that saves a total weight of at least $w(T)/(2c+2)$, and runs in time $O(n^{k+1}m)$.*

Proof. We start by ordering all the squads in $\bigcup_{i\in[\ell]} \mathcal{C}_i$ into $C_{1,1}, C_{1,2}, \ldots, C_{2,1}, \ldots, C_{\ell,1}, \ldots$, where $C_{i,j} \in \mathcal{C}_i$ for all $j \leq c$. We then define $T(C_{i,j})$, as the set of vertices of T first saved by $C_{i,j}$, that is, all vertices in T saved by $C_{i,j}$ not saved by any previous squad in the ordering above. We also define $T(\mathcal{C}_i) = \bigcup_{C\in\mathcal{C}_i} T(C)$.

We will show that, after the algorithm has chosen ℓ squads, the strategy S that protects those squads saves a total weight of at least $w(T)/(2c+2)$. Let Q_i be the squad chosen by the algorithm on iteration i. For any squad, we define as $\text{saved}(Q)$ the set of vertices saved by Q, and as $\text{saved}'_i(Q)$ the set of vertices saved by Q after the first $i-1$ iteration, that is,

$$\text{saved}'_i(Q) = \text{saved}(Q) \setminus \bigcup_{j=1}^{i-1} \text{saved}(Q_j)$$

Now, we say that iteration i was a *good* iteration if

$$w(\text{saved}'(Q_i)) \geq \frac{w(T(\mathcal{C}_i))}{c+1}$$

Otherwise, we say iteration i was *bad*. In this case, since at iteration i any squad in $\mathcal{C}_i$ can be protected, then for all $C \in \mathcal{C}_i$, $w(\text{saved}'_i(C)) < \frac{w(T(\mathcal{C}_i))}{c+1}$. This implies that a total weight of at least $w(T(C)) - \frac{w(T(\mathcal{C}_i))}{c+1}$ of $T(C)$ must already have been saved in previous iterations. Taking the union over all squads in $\mathcal{C}_i$, we conclude that, out of the vertices in $T(\mathcal{C}_i)$, S must save a total weight of at least

$$\begin{aligned} w\big(\text{saved}(S) \cap T(\mathcal{C}_i)\big) &\geq \sum_{C\in\mathcal{C}_i} \left(w(T(C)) - \frac{w(T(\mathcal{C}_i))}{c+1} \right) \\ &\geq w(T(\mathcal{C}_i)) - c\frac{w(T(\mathcal{C}_i))}{c+1} \\ &= \frac{1}{c+1} w(T(\mathcal{C}_i)) \end{aligned}$$

This means that, in every iteration, the strategy chosen by the algorithm either saves a total weight of at least $w(T(\mathcal{C}_i))/(c+1)$, or we can conclude that a weight of at least $w(T(\mathcal{C}_i))/(c+1)$ of $T(\mathcal{C}_i)$ has already been saved. Let $T^g = \bigcup_{\text{good } i} T(\mathcal{C}_i)$, $T^b = T \setminus T^g$ be the subset of vertices in T assigned to good and bad iterations, respectively. We know that our solution saves a weight of at least $w(T(\mathcal{C}_i))/(c+1)$ in every good iteration, and hence $w(\text{saved}(S)) \geq w(T^g)/(c+1)$. As to the bad iterations, we know that while we may perform badly, S saves a fraction of $1/(c+1)$ of the total weight in $T(\mathcal{C}_i)$, and hence saves a fraction of $1/(c+1)$ of the total weight in T^b. We conclude that, since $w(T^g) + w(T^b) = w(T)$,

$$w(\text{saved}(S)) \geq \frac{1}{c+1} \max\left(w(T^g), w(T^b)\right) \geq \frac{1}{2(c+1)} w(T) \qquad \square$$

Having established the needed technical tools, we can now prove our results on k-completable graphs. This, together with the fact outerplanar graphs are 3-completable (Theorem 7.7), proves Theorem 7.3. The rest of this section is dedicated to proving Theorem 7.8, which we recall for convenience.

Theorem 7.8 (page 119). *Let (G, s, w, B) be an instance of the problem and $k \in \mathbb{Z}^+$. Then there is an algorithm that runs in time $O(n^{k+1} m)$ and finds a solution that is:*

(1) a $(4k+2)$-approximation if G is k-weak-completable and $B \geq k$;

(2) a $(4k^2+4k+2)$-approximation if G is k-completable (and $B < k$);

Proof. Let S be an optimal solution, and $\hat{S}$ be defined as in Lemma 7.21. We consider an ordering $v_1, v_2, \ldots, v_\ell$ of the vertices of $\hat{S}$, so that $d(v_j)$ is non-decreasing.

Case 1 (k-weak-completable, $B \geq k$): For this case, we consider the following squads: $C_1, C_2, \ldots, C_\ell$, where C_j is the cut corresponding to v_j in Definition 7.13. We then partition the squads into sets $\mathcal{C}_i$ of $2k$ squads, that is,

$$\mathcal{C}_i = \{C_j : j \in [\ell], 2k(i-1) < j \leq 2ki\}$$

We will now apply Lemma 7.22 with $c = 2k$. To do this, we need to show that any set in $\mathcal{C}_i$ can be protected along with any other $(i-1)k$ vertices. Let $t = d(v_{2k(i-1)+1})$; since $d(v_j)$ is non-decreasing, we know that $d(v_j) \geq t$ for $j > 2k(i-1)$. To prove that we can protect any set in $\mathcal{C}_i$ along with any other $(i-1)k$ vertices, it is now sufficient to prove that $ik \leq Bt$, which means that we are able to protect all of the ik vertices in t time steps, by protecting at most B vertices per time step. We know that $i \leq \lceil Bt/2k \rceil$, since all of the vertices with $d(v_j) \leq t$ are contained in the first $\lceil Bt/2k \rceil$ sets. This implies our statement: we want to be able to protect at most ik vertices until time step t, to satisfy the conditions of the lemma, and indeed,

$$ik \leq \left\lceil \frac{Bt}{2k} \right\rceil k \leq \frac{Bt}{k} k = Bt$$

The last inequality follows because $\lceil a/2b \rceil \leq a/b$ for $a \geq b$.

Applying Lemma 7.22 with $c = 2k$ shows that the greedy algorithm outputs a $2(c+1) = (4k+2)$-approximation in this case.

Case 2 (k-completable, $B < k$): For this case, we do something slightly different: we consider an ordering $v_1, v_2, \ldots, v_\ell$ of the vertices of $\hat{S}$ as before, but we consider the squads and their partition only for the vertices satisfying $d(v_j) > k$. The remaining vertices, which are closest to the root, are handled separately.

Let j' be the maximum index such that $d(v_{j'}) \leq k$. Notice that $j' \leq Bk$, since $\hat{S}$ can protect at most Bk vertices at most k units away from the source. We now obtain the squads for v_j, $j > j'$, and partition them as before into sets $\mathcal{C}_1, \mathcal{C}_2, \ldots$.

Similarly to the previous case, we show that Lemma 7.22 is applicable on these sets. Since $t \geq k$ (and hence $Bt \geq k$), we can use the previous argument to conclude that for any set of $(i-1)k$ vertices, there is a strategy protecting them, plus any set $C_i \in \mathcal{C}_i$.

It remains to handle the vertices v_j, $j \leq j'$. For these, we invoke the properties of k-completable graphs (Definition 7.15), and take the sets $C_{j,1}, C_{j,2}, \ldots, C_{j,2k}$ in the definition. In each $\mathcal{C}_i$, $i \in [B]$, we insert sets $C_{j,\ell}$ for $(i-1)k < j \leq ik$. For each set $\mathcal{C}_i$, $i \in [B]$, and corresponding j, we know that $j \leq B\, d(v_j)$, and $j > k(i-1)/B$, which implies that $Bd(v_j) > (i-1)k$. This means that we can still protect at least one vertex at time step $d(v_j)$. By the properties of k-completable graphs, this is enough, since each squad can be protected on time steps $[d(v_j), d(v_j) + k - 1]$ with budget 1, and we proved that there is still at least 1 unit of budget free for each time step in that interval.

We conclude that all of the squads in the sets $\mathcal{C}_i$ satisfy the required properties, and therefore, we can apply Lemma 7.22 with $c = \max |\mathcal{C}_i| = 2k^2 + 2k$. Therefore, we show that the greedy algorithm outputs a $2(c+1) = (4k^2 + 4k + 2)$-approximation in this case. □

7.3 The Firefighter Problem on Bounded-Treewidth Graphs

In this section, we prove that it is NP-hard to approximate the Max-FF problem on graphs of treewidth $2B + 3$ to any factor smaller than $n^{1-\varepsilon}$, for any $\varepsilon > 0$, and any budget $B \geq 1$. To show this result, we present a reduction from the CM-1-in-3-SAT problem, that is, given an instance ϕ for this problem, we construct a tree instance T_ϕ for Max-FF. This tree has the property that, if ϕ is CM-1-in-3-SAT-satisfiable, there is a strategy with budget 1 that will stop the fire in T_ϕ and save all of its leaves. If, on the other hand, ϕ is not CM-1-in-3-SAT-satisfiable, then we will show that there is no strategy that saves all of the leaves of T_ϕ.

As a reminder, a formula ϕ for CM-1-in-3-SAT has no negative literals, every variable appears in exactly 3 clauses and each clause contains 3 literals (see also Section 7.1). The goal is to find an assignment such that every clause contains exactly 1 true literal. The NP-hardness of CM-1-in-3-SAT is implied by the results of Moore and Robson [MR01], as well as by Porschen et al. [PSS10], who prove the NP-hardness of a more general problem, XSAT for k-CNF$^{\ell}_{+}$ (CM-1-in-3-SAT corresponds to the case of $k = \ell = 3$).

Since CM-1-in-3-SAT is NP-hard, this implies that it is also NP-hard to determine, given a tree T, whether there is a strategy (with budget 1) that saves all of its leaves. Finbow et al. [FKM$^+$07] were the first to present such a construction, but we will use instead an adaptation of the result of Chlebíková and Chopin [CC14] for our purpose. By changing their construction, we can ensure that, until all leaves are saved, there is always one leaf at most 2 steps away from the fire. This property then allows us to prove that, for any budget $B \geq 1$, there is a graph $G_{\phi,B}$ with treewidth $2B + 3$ such that a

large group of vertices can be saved if ϕ is CM-1-in-3-SAT-satisfiable, and almost all of these vertices burn otherwise.

The construction of $G_{\phi,B}$ is simple: for $B = 1$, we start from T_ϕ, and add 4 *gate vertices*. These gate vertices are all connected by edges to every leaf of T_ϕ, as well as to $M = n^{O(1/\varepsilon)}$ new additional vertices, which we call *drones*. Notice that $G_{\phi,B}$ has feedback vertex number 4, as removing the gate vertices makes it a forest, which also implies that the treewidth of $G_{\phi,B}$ is 5. For $B > 1$, we add a subtree that forces any strategy to "waste" the additional budget, and we increase the number of gate vertices to $2B + 2$ to handle the higher budget.

If ϕ is satisfiable, there is a strategy that saves all of the leaves of T_ϕ, and hence in $G_{\phi,B}$ saves all of the gate vertices and M drones. Otherwise, we can show that the fire always reaches the gate vertices, and that they cannot all be protected by the time the fire reaches them. Therefore, at least one of the gate vertices burns, and the fire then spreads to the drones, very few of which can be protected.

The rest of this section is dedicated to proving Theorem 7.4 formally, which we recall for convenience.

Theorem 7.4 (page 118). *Let $\varepsilon > 0$, $B \geq 1$ and ϕ be an instance of CM-1-in-3-SAT.*

There is a polynomial-time reduction from ϕ to an instance $\mathcal{I} = (G, s, \mathbf{1}, B)$ of Max-FF, where G has N vertices and treewidth $2B + 3$, such that:

- *If ϕ is CM-1-in-3-SAT-satisfiable, there is a valid strategy for $\mathcal{I}$ with budget B that saves at least $N^{1-\varepsilon}$ vertices.*
- *If ϕ is not CM-1-in-3-SAT-satisfiable, there is no valid strategy for $\mathcal{I}$ with budget B that saves more than N^{ε} vertices.*

We start by showing the construction of T_ϕ and proving that it is possible to save all the leaves of T_ϕ if and only if ϕ is CM-1-in-3-SAT-satisfiable (with budget 1). As long as the condition above holds, and there is always one leaf at most 2 steps aways from the fire, we can construct an instance $G_{\phi,B}$ for a given $B \geq 1$, such that $G_{\phi,B}$ has treewidth $2B + 3$ and approximating Max-FF on $G_{\phi,B}$ is equivalent to deciding if ϕ is satisfiable. We give a simplified example of T_ϕ in Figure 7.2.

Tree instance T_ϕ Let ϕ be a formula of CM-1-in-3-SAT with n variables $x_1, \ldots, x_n$ and m clauses $C_j = (x_{i(j,1)} \vee x_{i(j,2)} \vee x_{i(j,3)})$. We will work with the clauses C_j and their negation $\overline{C}_j = (\bar{x}_{i(j,1)} \vee \bar{x}_{i(j,2)} \vee \bar{x}_{i(j,3)})$. This allows us to restate the problem as finding an assignment where each C_j contains at most one true literal, and $\overline{C}_j$ contains at most two true literals (where the respective variables are assigned to false).

We will now show how to construct T_ϕ. The construction follows the one by Chlebíková and Chopin [CC14], but there are both simplifications and changes to their construction. Start with a path $(s = u_1, u'_1, u_2, u'_2, \ldots, u_n)$, where s is the fire source. Each of the vertices u'_i has exactly one child (u_{i+1}), whereas the vertices u_i have children v_i, $\bar{v}_i$, and, unless $i = n$, u'_i. To each vertex v_i (resp. $\bar{v}_i$) add a leaf child, as well as a path with $2(n - i)$ edges. The end vertex of this path has 3 children, each corresponding to a clause containing x_i. We denote by ℓ^i_j (resp. $\bar{\ell}^i_j$) the vertex corresponding to clause

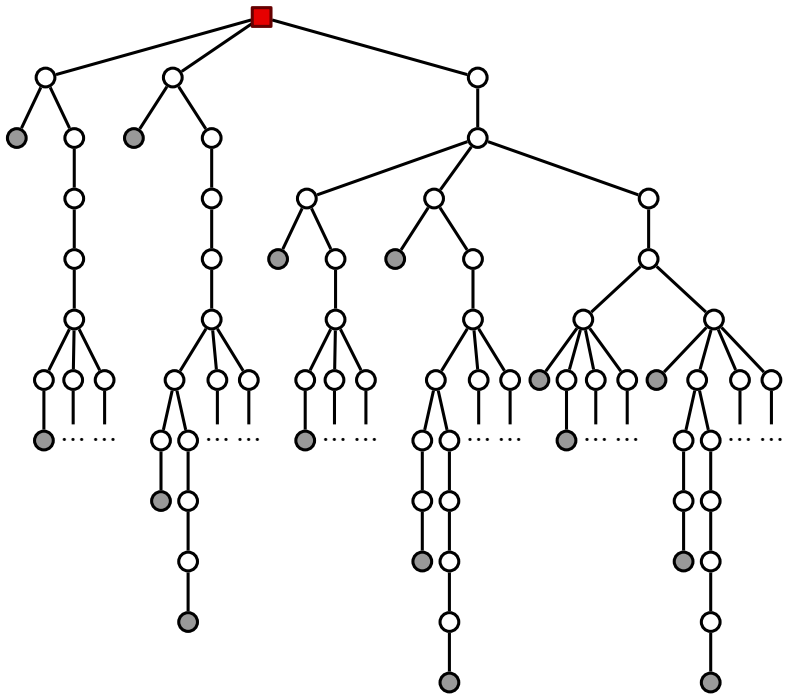

Figure 7.2: Construction of T_ϕ for 3 variables and one example clause, $x_1 \vee x_2 \vee x_3$. In $G_{\phi,B}$, all the leaves (shaded gray) are connected to the gate vertices.

C_j (resp. $\overline{C}_j$). This completes the first phase of the construction, corresponding to the variables. Notice that all of the vertices ℓ_j^i and $\bar{\ell}_j^i$ are on layer $2n$, since, for any i, it takes $2(i-1)$ steps from s to u_i, 1 from u_i to v_i (or $\bar{v}_i$), $2(n-i)$ from v_i (or $\bar{v}_i$) to the end of the path, and 1 step more to ℓ_j^i (or $\bar{\ell}_j^i$).

After the first $2n$ layers, we have 5 layers for each clause. Let C_j be a clause. For each variable x_i in C_j, add a path of length $5(j-1)$ to the corresponding vertices ℓ_j^i and $\bar{\ell}_j^i$ (so that we reach the layers for C_j). Let λ_j^i, $\bar{\lambda}_j^i$ be the vertices at the end of these paths. Next, add a single child to λ_j^i, which will be a leaf. To $\bar{\lambda}_j^i$, add two paths, of length 2 and 4 for one of the variables, and 3 and 5 for the two others.

This concludes the construction of the instance. We will now prove that ϕ is satisfiable if and only if there is a strategy which saves all the leaves. Additionally, we are going to prove that one of the leaves is always two steps away from the fire, until all of the leaves are saved.

Observation 7.23. *T_ϕ has at most $56n^2$ vertices and height $7n$.*

Proof. Since there are n variables and n clauses, the height of the T_ϕ is $2n$ in the first phase, and $5n$ in the second phase.

In the first $2n$ layers, there are at most $2n+3$ vertices per layer: 1 vertex u_i or u_i', 2 vertices which are leaf children of some v_i, $\bar{v}_i$, and then 2 vertices per variable, corresponding to the paths from v_i, $\bar{v}_i$. In the second phase, each layer has size at most $6n+3$: each clause has at most 6 vertices in paths from ℓ_j^i to λ_j^i (or $\bar{\ell}_j^i$ to $\bar{\lambda}_j^i$), and the clause corresponding to the current layer may have 3 additional paths, from the forking

at $\bar{\lambda}^i_j$). Layer $2n+1$ has $6n+2$ vertices: all the vertices ℓ^i_j, $\bar{\ell}^i_j$, plus the leaf children of v_n, $\bar{v}_n$.

We conclude that, since each layer has at most $6n+2 \leq 8n$ vertices, the whole instance has size at most $56n^2$. □

Lemma 7.24. *Let ϕ be a formula for CM-1-in-3-SAT and T_ϕ be the corresponding tree constructed as above.*

Then ϕ is CM-1-in-3-SAT-satisfiable if and only if there is a strategy for T_ϕ that saves all the leaves. Additionally, until the fire stops spreading, it can spread to a leaf in 2 time steps.

Proof. We will divide the proof in two parts: first we show a strategy that saves all the leaves if ϕ is satisfiable, then we show that any solution must be well-behaved, which implies that there is no strategy that saves all leaves if ϕ is unsatisfiable, and that the fire can always spread to a leaf in 2 time steps.

Assume ϕ is CM-1-in-3-SAT-satisfiable, and let a be a satisfying assignment. For every variable x_i, if $a_i = 1$, we will protect $\bar{v}_i$ and the leaf child of v_i; if $a_i = 0$, we protect v_i and the leaf child of $\bar{v}_i$. This defines the strategy for the first $2n$ layers.

As detailed in the construction, each clause C_j corresponds to 5 layers in the tree (layers $2n+5(j-1)+1$ to $2n+5(j-1)+5$). Since a is a satisfying assignment, each clause C_j contains exactly one x_i set to true, and two variables $x_{i'}$, $x_{i''}$ set to false. This implies that after $2n+5(j-1)$ layers, λ^i_j, $\bar{\lambda}^{i'}_j$ and $\bar{\lambda}^{i''}_j$ are burning, and $\bar{\lambda}^i_j$, $\lambda^{i'}_j$, and $\lambda^{i''}_j$ are saved by protecting $\bar{v}_i$, $v_{i'}$, and $v_{i''}$ respectively. In the first layer for C_j, we protect the leaf child of λ^i_j, then in the second and third layers, protect vertices on the shorter paths leaving $\bar{\lambda}^{i'}_j$ and $\bar{\lambda}^{i''}_j$ (of sizes ≥ 2 and ≥ 3), and in the fourth and fifth layers, protect vertices on the longer paths leaving $\bar{\lambda}^{i'}_j$ and $\bar{\lambda}^{i''}_j$ (of sizes ≥ 4 and ≥ 5).

To conclude, we protect all the leaves that all children of v_i, $\bar{v}_i$ vertices, as well as all of the leaves in the layers of each clause, which means that the strategy described saves all the leaves.

We will now argue that if a strategy saves all the leaves, it must be well-behaved, meaning that for each variable, it has to protect either:

(1) v_i and the leaf child of $\bar{v}_i$;

(2) $\bar{v}_i$ and the leaf child of v_i; or

(3) both leaf children of v_i and $\bar{v}_i$.

Additionally, we will show that in the first $2n$ time steps, there is always one leaf that is at most 2 time steps away from the fire. We prove this claim by induction on n for a strategy that saves all the leaves.

Consider the first time step: there are two leaves, children of v_1 and $\bar{v}_1$ respectively, 2 time steps away from the source. Since the paths to these leaves are disjoint, the solution must protect one vertex in each path, and it cannot protect both v_1 and $\bar{v}_1$ simultaneously. The remaining options are the ones claimed above, the strategy must respect one of these options in order to save the leaf children of v_i, $\bar{v}_i$.

If $n = 1$, the argument above proves the claim. If $n > 1$, we know that u_2 burns, and therefore we can apply the induction hypothesis for $n-1$ variables $x_2, \ldots, x_n$, with

source u_2. Notice that as part of the induction, we also show that there are always leaves at most 2 steps away from the fire.

We will continue our argument and show a similar argument for the clause layers: if a strategy saves all leaves, then, for each clause, there must be exactly one vertex λ_j^i and two vertices $\bar{\lambda}_j^i$ burning, and there is always one leaf at most two steps from the fire, until the fire stops spreading. Furthermore, the strategy protects only vertices descending from these burning vertices in the 5 layers corresponding to each clause.

Consider the first clause, C_1, and consider a strategy that saves all the leaves. We know that this strategy must be well-behaved in the first $2n$ layers, which means, that, for each variable x_i, λ_1^i or $\bar{\lambda}_1^i$ (or both) must be burning.

If more than one λ_1^i burns, it is not possible to save all the leaves. Since each of these has a leaf child, a strategy can only protect one of the leaf children, and therefore any additional children will burn. Therefore, we conclude that either one λ_1^i or none burn. If none burn, we know that all of the vertices $\bar{\lambda}_1^i$ must burn. However, if all of the $\bar{\lambda}_1^i$ burn since each of these has 2 leaf descendants, there are 6 leaves to save, all in disjoint paths, in only 5 layers. This is clearly impossible, since the strategy would have to protect 6 vertices in 5 layers. We conclude that, if the strategy saves all the leaves, then there must be there must be exactly one vertex λ_1^i and two vertices. $\bar{\lambda}_1^i$ burning.

Now we simply need to prove that in this case, the solution is well behaved. Let λ_1^i, $\bar{\lambda}_1^{i'}$, $\bar{\lambda}_1^{i''}$ be the burning vertices. Since λ_1^i has one leaf child, the strategy must protect in the first time step, since it will otherwise burn. Now, there are 4 leaves left (2 descendants of $\bar{\lambda}_1^{i'}$, $\bar{\lambda}_1^{i''}$ each) left to save, and there are also 4 layers left. At this point, the fire is at distance at most 2 from two of the leaves (those on the shorter paths). Therefore, the strategy must protect one vertex in each of these shorter paths. After that, the fire reaches the third layer of the clause, and again the remaining two leaves are at distance at most 2 from the fire, so the strategy must protect a vertex in each of the paths.

If the instance has only one clause, the proof is complete, and otherwise, the five layers corresponding to C_1 do not affect the strategy in the remaining layers, as only vertices corresponding to this clause were protected. Therefore, the instance is now similar to one with fewer clauses, and the induction hypothesis completes the proof.

The proof of the lemma follows easily: if ϕ is not satisfiable, then for each assignment there must be one clause that is not satisfied. Since a solution must be well-behaved to save all the leaves, this means that for any unsatisfied clause C_j, there will either be at least two vertices λ_j^i burning, or all $\bar{\lambda}_j^i$ burn, after $2n + 5(j-1)$ steps. As we have seen above, this implies that the solution cannot save all the leaves. □

Hardness instance $G_{\phi,B}$ Given a budget $B \geq 1$, we start by constructing a tree T'_ϕ that satisfies the guarantees of Lemma 7.24 for strategies with budget B. For $B = 1$, we have $T'_\phi = T_\phi$. For $B > 1$, we augment T_ϕ with some additional structure. Let r be the root of T_ϕ, and $h = \text{height}(T_\phi)$ be its height. For every $i \in [h]$, we add $B - 1$ paths of length i from the root, ending at $B - 1$ distinct leaves (in layer i). We call the resulting tree T'_ϕ. The idea is that any valid strategy for T'_ϕ must always protect $B - 1$ of the recently added leaves, or it will fail to save all the leaves. The remaining vertex can then be used to recreate a strategy in T_ϕ.

Observation 7.25. *T'_ϕ has at most $28n^2(B+1)$ vertices and height $7n$.*

Proof. The height of T'_ϕ is the same as T_ϕ, which by Observation 7.23 is $h = 7n$.

For each $i \in [h]$ we add $(B-1)i$ vertices to the tree, for a total of $h(h+1)(B-1)/2$ vertices. Since $h = 7n$, we add at most $28n^2(B-1)$ vertices. By Observation 7.23, T_ϕ has at most $56n^2$ vertices, so T'_ϕ has at most $56n^2 + 28n^2(B-1) = 28n^2(B+1)$ vertices. □

Let $\varepsilon > 0$, and let $N = (2|T'_\phi|)^{1/\varepsilon}$. We construct $G_{\phi,B}$ by adding, to T'_ϕ, $2B+2$ *gate vertices* $g_1, \ldots, g_{2B+2}$ and $M = N - |T'_\phi| - (2B+2)$ *drones* (so $|V(G_{\phi,B})| = N$). We also add edges connecting each g_i to all the leaves of T'_ϕ (including the newly added ones), as well as between each g_i and all the drones.

Observation 7.26. *$G_{\phi,B}$ has treewidth $2B+3$ and feedback vertex number $2B+2$.*

Proof. Removing the gate vertices $g_1, \ldots, g_{2B+2}$ turns $G_{\phi,B}$ into a forest, which implies the feedback vertex number bound. To obtain a tree decomposition of width $2B+3$ for $G_{\phi,B}$, start by constructing a tree decomposition of width 1 for $G_{\phi,B} \setminus \{g_i : i \in [2B+2]\}$, and add $g_1, \ldots, g_{2B+2}$ to all the bags. □

Lemma 7.27. *Let $\varepsilon > 0$, ϕ be a formula for CM-1-in-3-SAT and $G_{\phi,B}$ be the graph constructed as above.*

Then, if ϕ is CM-1-in-3-SAT-satisfiable there is a strategy for $G_{\phi,B}$ that saves at least M vertices. Otherwise, there is no strategy for $G_{\phi,B}$ that saves more than $2|T_\phi|$ vertices.

Proof. If ϕ is CM-1-in-3-SAT-satisfiable, we can construct a valid strategy that saves all the leaves of T'_ϕ: we use the strategy of Lemma 7.24 to protect one vertex of T_ϕ per layer, and protect the $B-1$ added leaves in layer i at time i, $i \in [h]$. By saving all the leaves, the fire cannot spread to the gate vertices or drones, which implies that at least M vertices (the drones) are saved.

Consider now the case that ϕ is not satisfiable, and consider a strategy S for $G_{\phi,B}$. We say that S is *respecting* at time i (or on time step i) if, on time step i, it protects the $B-1$ added leaves in layer i, at most one vertex in T_ϕ, and none of the gate vertices or drones.

By Lemma 7.24, we know that at least one leaf of T_ϕ burns if the strategy protects at most one vertex per time step in T_ϕ. In this case, the fire reaches the gate vertices. We will now show that the fire always reaches the gate vertices, and, by that time, S can only protect $2B+1$ of the gate vertices. Since there are $2B+2$ gate vertices, one of these burns, and the fire spreads on the next time step to all of the unprotected drones. This implies that at most $3B+1$ of the vertices outside T'_ϕ can be saved.

Let i be the earliest time step on which a leaf burns or S is not respecting ($i \leq h$ by Lemma 7.24). If S does not protect all the added leaves in layer i, then one of them burns; hence we can reduce this to the case that a leaf burns. If any leaf in T'_ϕ burns, then the fire reaches the gate vertices at time $i+1$. Since S is respecting up to time step $i-1$, it can only protect gate vertices on time steps i and $i+1$, and thus it can protect at most $2B$ gate vertices.

If no leaf burns on time step i, but S is not respecting, then S must protect a vertex outside T'_ϕ, and therefore does not protect a vertex in T_ϕ. In this case, S protects at most 1 gate vertex on this time step. By Lemma 7.24, there are always leaves in T_ϕ that are 2 time steps away from the fire; since no vertex in T_ϕ is protected at time i, the fire is now 1 step ahead, and hence there are always leaves that are 1 time step away from the fire.

Let $i' > i$ be the earliest time step after i on which a leaf burns or S is not respecting (again, $i' \leq h$ by Lemma 7.24). As above, if S does not protect all the added leaves in layer i, then a leaf burns. Furthermore, if no vertex in T_ϕ is protected, the leaves that are 1 time step away from the fire burn. Therefore, at least one leaf burns at time i', and the fire reaches the gate vertices at time $i' + 1$. S can protect $2B$ gate vertices on time steps i', $i' + 1$, plus 1 more at time i, for a total of $2B + 1$ gate vertices.

As we have shown above, at most $2B + 1$ gate vertices can be protected for any strategy S, which implies at most $3B + 1$ vertices outside of T'_ϕ can be saved. Since $3B + 1 < |T'_\phi|$, S saves at most $2|T'_\phi|$ vertices, as desired. □

We conclude that if ϕ is CM-1-in-3-SAT-satisfiable, there is a strategy saving at least $M = N - |T'_\phi| - 4 \geq N - N^\varepsilon \geq N^{1-\varepsilon}$, and otherwise, no strategy saves more than $2|T'_\phi| = N^\varepsilon$ vertices. This concludes the proof of Theorem 7.4.

Furthermore, Theorem 7.4 tells us that, unless $\mathrm{tw}(G) \leq 2B + 2$, Max-FF cannot be $n^{1-\varepsilon}$-approximated on G. In other words, if $B \geq (\mathrm{tw}(G) - 2)/2 = \mathrm{tw}(G)/2 - 1$, a better approximation may exist, as stated in Corollary 7.6.

CHAPTER 8

Conclusion and Open Problems

Our research on the firefighter problem has the goal of clarifying the aproximability of the problem in two settings: on trees, constant approximation algorithms are known, and we focused on the limitations of current techniques, as well as on overcoming them; on bounded-treewidth graphs, we sought to study the influence of treewidth on the approximability of the problem, and to determine when the problem becomes inapproximable, as it is known to be in general graphs [ACH$^+$12].

Until the recent PTAS was discovered by Adjiashvili et al. [ABZ17], the best approximation result on trees was a $(1-1/e)$-approximation algorithm by LP rounding [CVY08]. We show that this is essentially the best possible approximation ratio that can be obtained by comparing to the optimum value of the standard LP, since there are instances with integrality gap arbitrarily close to $1-1/e$.

We then present some evidence that a stronger LP, suggested by Hartke [Har04], might allow for better approximation results: we improve the approximation ratio when rounding half-integral solutions, as well as instances we call separable, which include the integrality gap instances for the standard LP. While Hartke's LP cannot be rounded to obtain an approximation factor better than 5/6, it is possible that a family of progressively stronger constraints can lead to better approximation algorithms, until a PTAS is achieved (matching the result of Adjiashvili et al. [ABZ17]).

Despite the existence of a PTAS for Max-FF on trees, the problem is known to be $n^{1-\varepsilon}$-inapproximable in general graphs. We show that this inapproximability carries over to much simpler graphs, with treewidth 5, or treewidth $2B+3$ for instances with budget B. This also implies that, on graphs of treewidth w, it is necessary that the budget is at least $B \geq w/2-1$, in order to obtain a subpolynomial approximation. On the algorithmic side, we show that outerplanar graphs, a family of simple planar graphs with treewidth 2, admit an $O(1)$-approximation algorithm.

In our opinion, these results help us understand the approximability of Max-FF, as well as the applicability of current techniques. Furthermore, they also raise other questions about the complexity of the problem, which have yet to be answered.

8.1 Open Problems

Regarding Max-FF on trees, the question of approximability has been answered by Adjiashvili et al. [ABZ17]. We would find it interesting to obtain an LP rounding algorithm that directly implies a PTAS, by using progressively stronger LP relaxations (see e.g. [Lau03]). It is possible that Hartke's LP is one step in this direction, but its integrality gap is only known to lie between $1-1/e$ and 5/6. Another question of interest regarding an LP for the problem would be to solve the LP in linear-time. This would imply a $(1-1/e)$-approximation, or even a PTAS, with a fast running time.

Open Problem 8.1. What is the integrality gap of Hartke's LP?

Open Problem 8.2. Is it possible to obtain a PTAS by rounding a family of progressively stronger LP relaxations?

Open Problem 8.3. Is it possible to solve the LP for Max-FF on trees with linear running time?

On the opposite end of the spectrum, one might ask if it is possible to obtain a PTAS or at least a good constant approximation with a fully combinatorial algorithm, especially with a linear running time. The best known result with these properties is the greedy 2-approximation algorithm by Hartnell and Li [HL00]. Anshelevich et al. [ACH$^+$12] show that Max-FF on trees can be formulated as maximizing a monotone submodular function subject to a partition matroid constraint. Interestingly, the same approximation factors, of $1 - 1/e$ for an LP rounding algorithm, and 2 for a greedy algorithm, are the best known for this more general problem. While this problem is NP-hard to approximate to a factor better than $1 - 1/e$ [FGM$^+$06], it would be interesting to obtain an improved combinatorial approximation algorithm, both for Max-FF on trees and the submodular function maximization problem above.

Open Problem 8.4. Is there a combinatorial algorithm for Tree-FF with an approximation factor better than 2?

Open Problem 8.5. Is there a combinatorial algorithm with an approximation factor better than 2 for the monotone submodular function maximization problem subject to a partition matroid constraint?

For the bounded-treewidth setting, the main open question is to determine the smallest value of treewidth such that Max-FF is inapproximable. If multiple firefighters per time step are allowed, we can also ask how large the budget B must be, as a function of treewidth, for a constant approximation to Max-FF to a exist. Our results show that $B \geq w/2-1$ necessarily, where w is the treewidth of the input graph; however, there is no known constant approximation algorithm, even for any B and treewidth 2. We conjecture that, similarly to the surviving rate [CCV$^+$10], a good approximation is possible if and only if $B \geq w$.

Open Problem 8.6. Is it NP-hard to $n^{1-\varepsilon}$-approximate Max-FF with budget B on graphs of treewidth $B + 1$, for any $\varepsilon > 0$, $B \geq 1$?

Open Problem 8.7. Is there an $O(1)$-approximation algorithm for Max-FF on graphs of treewidth w, with budget $B \geq w$?

While Max-FF may be hard to approximate even in bounded-treewidth graphs, it is possible that, on graphs with more structure such as planar graphs, better approximation algorithms do exist. Indeed, our result for outerplanar graphs uses planarity, as well as other properties, to obtain an $O(1)$-approximation, and our hardness instances are not planar. This could suggest that planarity might be exploited to obtain constant approximations to the problem. One good starting point could be grid graphs, which could benefit from previous work on fire containment in infinite grids (see [FM09]).

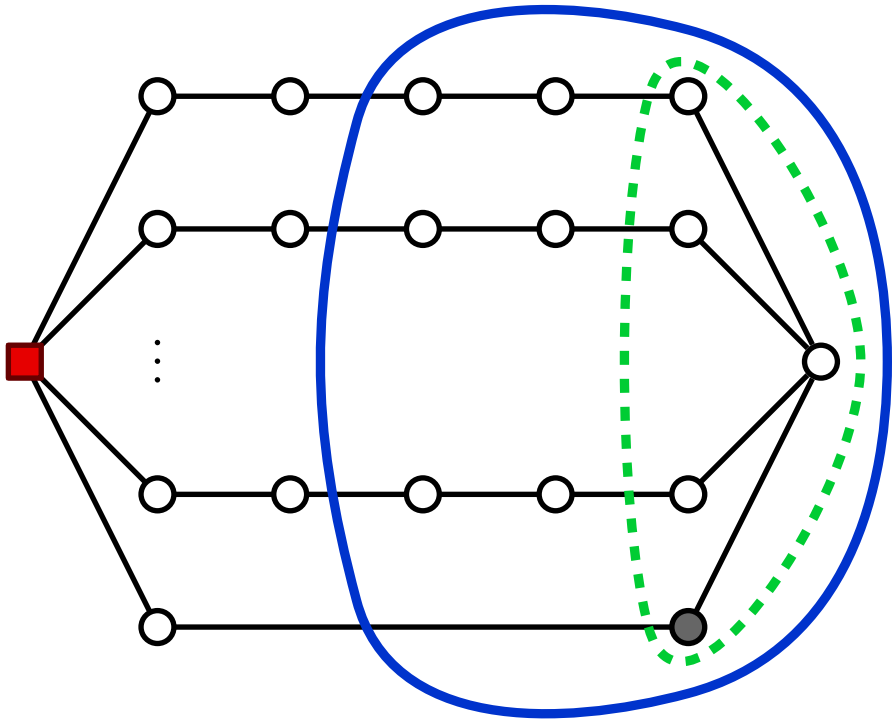

Figure 8.1: Example of $\Omega(n)$-completable instance with treewidth 2, consisting of $(n-4)/5$ parallel paths. The fire source is marked as a filled square (red), a vertex v is marked as a filled circle. The vertices delayed and influenced by v are the vertices inside the dashed (green) and full (blue) curve, respectively. In order to save all the vertices delayed by v, $(n-4)/5 > n/10$ vertices must be protected (one per path).

Open Problem 8.8. Is there a constant approximation for Max-FF on grid graphs?

Open Problem 8.9. Is there a constant approximation for Max-FF on planar graphs?

Our result for outerplanar graphs relies on the notion of k-completable graphs, by showing that, in such graphs, a simple greedy algorithm that considers k vertices at a time outputs a constant approximation to the problem. It might be possible to improve both the running time, and the dependence of the approximation ratio on the value of k. Moreover, there are simple examples of graphs with treewidth 2 that are not $(n/10)$-completable (see Figure 8.1). However, a similar idea of covering all of the vertices saved by the solution, using small vertex cuts, seems to work intuitively in these examples. This suggests that the notion of k-completable graphs might not be the most adequate, and raises the question of whether a better definition exists.

Open Problem 8.10. Can we improve the running time of the approximation algorithm for outerplanar graphs?

Open Problem 8.11. Can we improve the approximation ratio for k-completable graphs?

Open Problem 8.12. Is there a better notion of k-completable, such that treewidth-2 graphs are k-completable, for some constant k?

Finally, it is worthwhile to discuss how to extend the firefighter problem, so that it could be used to model real-world situations more accurately. In epidemiology, one of the best-known models is the SIR model (see e.g. [Het00]), which stands for *Susceptible*, *Infectious*, and *Removed*, the three states that an individual can be in the model. The Max-FF problem uses a restricted version of the SIR model in which susceptible individuals can become infectious by being infected by a neighbor, or become removed by being protected by the strategy. It would be interesting to generalize the Max-FF problem so that infectious individuals can recover and gain immunity. An example of such a problem could be to minimize the maximum number of simultaneous infectious individuals, given a budget for vaccination per time step. To the extent of our knowledge, no effort has been made to solve such problems or analyze their computational complexity.

APPENDIX A

Appendix for Part I

A.1 Omitted proofs of Chapter 3

A.1.1 Proof of Lemma 3.17

Lemma 3.17 (page 27). *Let $(\Omega, \mathcal{F}, P)$ be a probability space, $A_i \in \mathcal{F}$ be events with indicator variable X_i and $X = \sum_i X_i$.*

Then $\mathrm{E}[X \mid X \geq 1] \leq \max_i \mathrm{E}[X \mid A_i]$.

Proof. We will now consider a probability space conditioned on $X \geq 1$. Formally, we consider a probability space $(\Omega', \mathcal{F}', P')$, with $\Omega' = \bigcup_i A_i$, $\mathcal{F}' = \{A \subseteq \mathcal{F} \mid A \subseteq \Omega'\}$ and

$$P'(A) = \frac{P(A)}{P(\Omega')} = \frac{P(A)}{P(X \geq 1)}$$

We will write $\mathrm{E}[Z]$ for the expected value of Z with probability function P, and $\mathrm{E}'[Z]$ with probability function P'.

This formulation allows us to state some interesting facts. We start by showing that $\mathrm{E}'[X] = \mathrm{E}[X \mid X \geq 1]$. Indeed,

$$\mathrm{E}'[X] = \sum_i P'(A_i) = \sum_i \frac{P(A_i)}{P(X \geq 1)} = \frac{\mathrm{E}(X)}{P(X \geq 1)} = \mathrm{E}[X \mid X \geq 1]$$

Similarly, we can show that $\mathrm{E}'(X \mid A_i) = \mathrm{E}(X \mid A_i)$:

$$\begin{aligned} \mathrm{E}'[X \mid A_i] &= \sum_j \frac{P'(A_j \cap A_i)}{P'(A_i)} \\ &= \sum_j \frac{P(A_j \cap A_i)/P(X \geq 1)}{P(A_i)/P(X \geq 1)} \\ &= \mathrm{E}[X \mid A_i] \end{aligned}$$

We can now finish the proof using the above facts, together with Jensen's inequality,

$\mathrm{E}'[X]^2 \leq \mathrm{E}'[X^2]$:

$$\begin{aligned}
\mathrm{E}'[X]^2 &\leq \mathrm{E}'[X^2] \\
&= \sum_{i,j} P'(A'_i \cap A'_j) \\
&= \sum_i P'(A_i) \sum_j P'(A_j \mid A_i) \\
&= \sum_i P'(A_i)\, \mathrm{E}'[X \mid A_i] \\
&= \sum_i P'(A_i)\, \mathrm{E}[X \mid A_i] \\
&\leq \sum_i P'(A_i) \max_i \mathrm{E}[X \mid A_i] \\
&= \mathrm{E}'(X) \max_j \mathrm{E}[X \mid A_j]
\end{aligned}$$

Rearranging and replacing $\mathrm{E}'[X] = \mathrm{E}[X \mid X \geq 1]$, we get:

$$\mathrm{E}[X \mid X \geq 1] \leq \max_j \mathrm{E}[X \mid A_j] \qquad \square$$

A.2 Algorithms

A.2.1 Rounding Algorithm of Garg et al. [GKR00]

```
function ApproxGST(G, c, r, {S_i}_{i∈[h]})
    // First solve the LP
    x, f ← SolveLP(G, c, r, {S_i}_{i∈[h]})

    // Repeat rounding ℓ log n log h times for some large ℓ
    for i ∈ [ℓ log n log h] do
        F_i ← RoundGKR(G, r, x)
    end for
    F̂ ← ⋃_{i∈[ℓ log n log h]} F_i

    // If a group is not covered, connect it using a shortest path
    for i ∈ [h] do
        if F̂ ∩ S_i = ∅ then
            F̂ ← F̂ ∪ ShortestPath(r, S_i)
        end if
    end for

    return F
end function

function RoundGKR(G, r, x)
    F ← ∅                    // Our solution

    for e ∈ E(G) in BFS order from r do
        if p(e) ∈ F then
            Add e to F with probability x_e/x_{p(e)}
        end if
    end for

    return F̂
end function
```

Algorithm A.1: Complete algorithm to approximate GST [GKR00].

A.2.2 Modified Rounding Algorithm for STGST (Section 3.4.2)

function RoundMod($\widetilde{\mathcal{T}}$, $\widetilde{\mathcal{T}}_p$, x)
  $\mathcal{Y} \leftarrow \emptyset$ *// Our solution*

  for $e = (\lambda, \tilde{t}) \in E(\widetilde{\mathcal{T}})$ in BFS order **do**
    if $e \in \mathcal{Y}$ and $\tilde{t} \in \widetilde{\mathcal{T}} \setminus \widetilde{\mathcal{T}}_p$ **then**
      // If $\tilde{t}$ is a combination node, choose independently
      for $e' \in E(\widetilde{\mathcal{T}})$ children of e ($p(e') = e$) **do**
        Add e' to $\mathcal{Y}$ with probability $x_{e'}/x_e$
      end for
    else if $e \in \mathcal{Y}$ and $\tilde{t} \in \widetilde{\mathcal{T}}_p$ **then**
      // Otherwise, choose exactly one edge
      Pick exactly one child edge e' of e, with probability $x_{e'}/x_e$.
      Add e' to $\mathcal{Y}$
    end if
  end for

  return $\mathcal{Y}$
end function

Algorithm A.2: Modified algorithm for rounding an instance of Theorem 3.38.

A.2.3 Rounding Algorithm for GSF (Section 3.5)

function ApproxGSF(G, c, $\{(A_i, B_i)\}_{i\in[h]}$)
  $\hat{F} \leftarrow \emptyset$ // *Our solution*
  $P \leftarrow [h]$ // *Set of yet uncovered group pairs*

  while $|P| > 0$ **do**
    // *Compute solution for every choice for the root*
    for $r \in V$ **do**
      $F_r \leftarrow$ ApproxMDGSF$(G, c, r, \{(A_i, B_i)\}_{i\in P})$
    end for

    // *Choose best solution in this iteration*
    $F' \leftarrow \operatorname{argmin}\{d(F_r) : r \in V\}$
    $\hat{F} \leftarrow \hat{F} \cup F'$

    // *Update* P
    for $i \in [h]$ **do**
      Remove i from P if F' connects A_i to B_i
    end for
  end while

  return $\hat{F}$
end function

Algorithm A.3: Algorithm to approximately solve GSF.

A.2.4 Rounding Algorithm for MDGSF (Section 3.5)

function ApproxMDGSF(G, c, r, $\{(A_i, B_i)\}_{i\in[h]}$)
  // Build an instance of MDSTGSF for $\bigcup_{i\in[h]}\{A_i, B_i\}$
  $(\widetilde{\mathcal{T}}, \tilde{c}, \tilde{r}, \widetilde{S}, \widetilde{\mathcal{T}}_c, \widetilde{\mathcal{T}}_p) \leftarrow$ MDSTGSF$(G, c, r, \bigcup_{i\in[h]}\{A_i, B_i\})$

  // Solve the LP
  $x, y \leftarrow$ SolveLP-MDSTGSF$(\widetilde{\mathcal{T}}, \tilde{c}, \tilde{r}, \{(\widetilde{A}_i, \widetilde{B}_i) : i \in P\})$

  // Choose the best bucket
  Choose $j^* = \text{argmax}_{j\in[2\log h]} \sum_{i\in\mathcal{Y}_j} y_i$,
    where $\mathcal{Y}_j = \{i \in [h] : 2^{-j} < y_i \le 2^{-(j-1)}\}$

  // Repeat rounding $2^{j^*}\ell \log n$ *times for some large* ℓ
  for $i \in [2^{j^*}\ell \log n]$ **do**
    $F_i \leftarrow$ RoundMod$(\widetilde{\mathcal{T}}, \widetilde{\mathcal{T}}_p, x')$ *// From Algorithm A.2*
  end for

  return $\bigcup_{i\in[\ell\log n]} F_i$
end function

Algorithm A.4: Algorithm to approximately solve MDGSF.

A.2.5 Algorithm for Connectivity-K Mimicking Networks (Section 4.3)

function ConnKMimNet(G, X, K)
  // ViolatingCut *computes a cut that violates connectivity-K linkedness, i.e.* $|E(A,B)| < \min(|X \cap A|, |X \cap B|, K)$
  $(A, B) \leftarrow$ ViolatingCut(G, X, K)

  // *Return a star containing all terminals, centered at a steiner vertex v*
  if no such cut exists **then**
    return $(X \cup v, \{(v, u) : u \in X\})$
  end if

  Let $G_A \leftarrow G[A]$, $X_A \leftarrow T \cap A$, $G_B \leftarrow G[B]$, $X_B \leftarrow T \cap B$
  for every edge $(u, v) \in E(A, B)$ **do**
    Add a terminal t_{uv} to G_A, X_A and edge (u, t_{uv}) to G_A
    Add a terminal t_{vu} to G_B, X_B and edge (u, t_{vu}) to G_B
  end for

  $H_A \leftarrow$ ConnKMimNet(G_A, X_A, K), $H_B \leftarrow$ ConnKMimNet(G_B, X_B, K)
  $H \leftarrow H_A \cup H_B$
  for every edge $(u, v) \in E_G(A, B)$ **do**
    Remove t_{uv}, t_{vu} from H
    Add edge (u, v) to H
  end for

  return H
end function

Algorithm A.5: Algorithm for computing a minor connectivity-K mimicking network.

APPENDIX B

Appendix for Part II

B.1 Algorithms

```
function GreedyFF(G, s, w, B, k)
    U ← ∅                    // Our strategy
    B ← {s}                  // Set of burning vertices

    // While the fire is spreading
    while N(B) \ U ≠ ∅ do
        // Find best set of k vertices
        Find Q, |Q| ≤ k, such that
            U ∪ Q is schedulable,
            Q maximizes the weight of vertices saved in G \ U.
        U ← U ∪ Q, B ← B ∪ N(B) \ Q
    end while

    return U
end function
```

Algorithm B.1: Greedy algorithm for k-completable and k-weak-completable graphs.

B.2 Integrality instance for LP-Hartke ($\alpha = 2$)

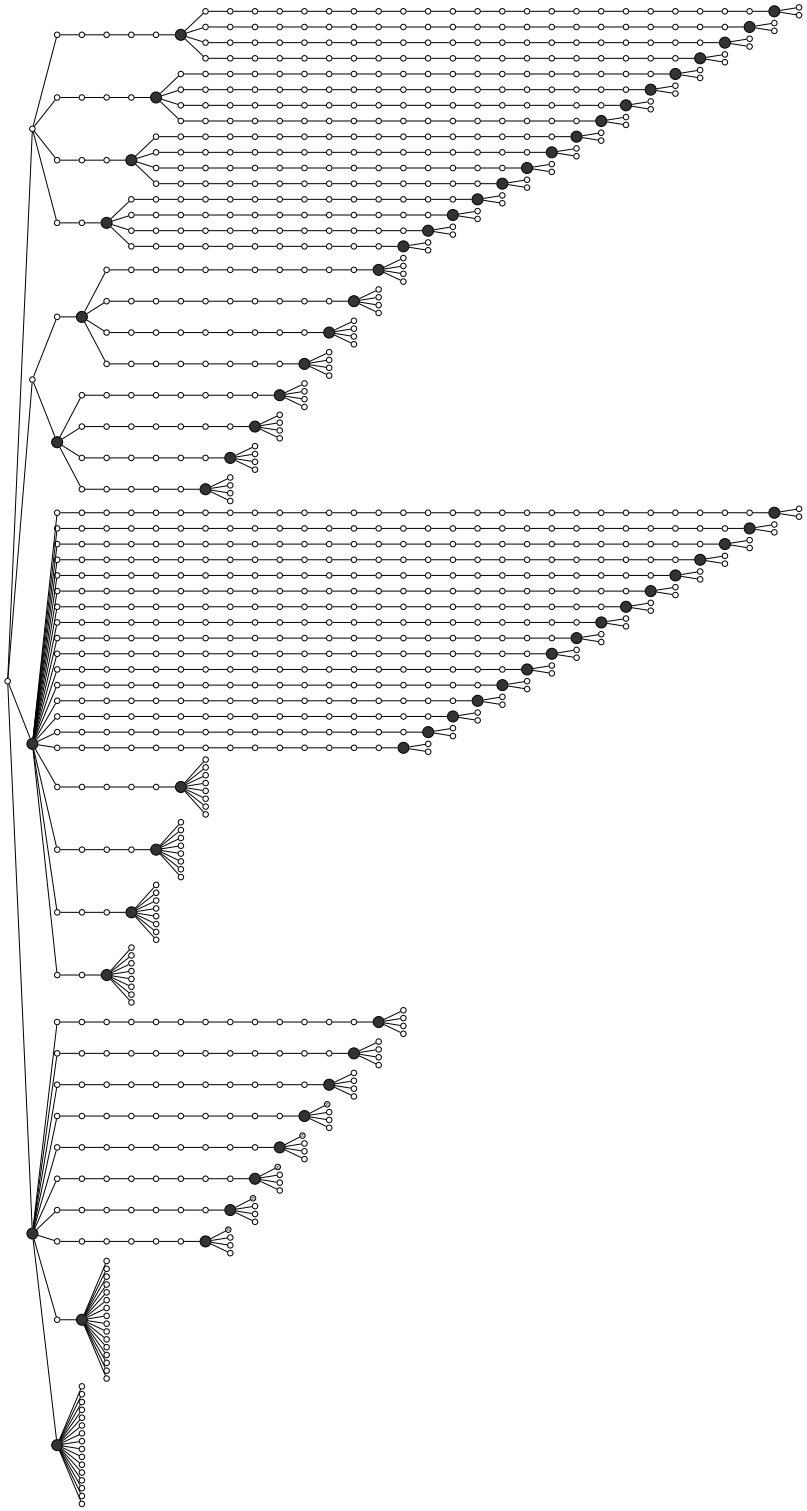

Figure B.1: Full example of the integrality gap instance for LP-Hartke, when $\alpha = 2$. Special vertices ($x_v = 1/2$) are colored *black*.

Bibliography

[AB18] Amir Abboud and Greg Bodwin. Reachability preservers: New extremal bounds and approximation algorithms. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1865–1883. SIAM, 2018.

[ABZ17] David Adjiashvili, Andrea Baggio, and Rico Zenklusen. Firefighting on trees beyond integrality gaps. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2364–2383. SIAM, 2017.

[ACH+12] Elliot Anshelevich, Deeparnab Chakrabarty, Ameya Hate, and Chaitanya Swamy. Approximability of the firefighter problem – computing cuts over time. *Algorithmica*, 62(1-2):520–536, 2012.

[AKP+95] Noga Alon, Richard M. Karp, David Peleg, and Douglas B. West. A graph-theoretic game and its application to the k-server problem. *SIAM J. Comput.*, 24(1):78–100, 1995.

[Aro98] Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

[Bü11] İSMET Esra Büyüktahtakin. *Dynamic Programming Via Linear Programming*. Wiley Online Library, 2011.

[Bak94] Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

[Bar96] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 184–193. IEEE Computer Society, 1996.

[Bar98] Yair Bartal. On approximating arbitrary metrices by tree metrics. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 161–168. ACM, 1998.

[BB73] Umberto Bertelè and Francesco Brioschi. On non-serial dynamic programming. *J. Comb. Theory, Ser. A*, 14(2):137–148, 1973.

[BBM+13] Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Approximation algorithms for spanner problems and directed Steiner forest. *Inf. Comput.*, 222:93–107, 2013.

[BCC+14] Cristina Bazgan, Morgan Chopin, Marek Cygan, Michael R. Fellows, Fedor V. Fomin, and Erik Jan van Leeuwen. Parameterized complexity of firefighting. *J. Comput. Syst. Sci.*, 80(7):1285–1297, 2014.

[BCK+15] Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015.

[BCR13] Cristina Bazgan, Morgan Chopin, and Bernard Ries. The firefighter problem with more than one firefighter on trees. *Discret. Appl. Math.*, 161(7-8):899–908, 2013.

[BDD+16] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A c^k n 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016.

[BDH+16] MohammadHossein Bateni, Erik D. Demaine, MohammadTaghi Hajiaghayi, and Dániel Marx. A PTAS for planar group Steiner tree via spanner bootstrapping and prize collecting. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 570–583. ACM, 2016.

[BDT14] Glencora Borradaile, Erik D. Demaine, and Siamak Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Algorithmica*, 68(2):287–311, 2014.

[BG07] André Berger and Michelangelo Grigni. Minimum weight 2-edge-connected spanning subgraphs in planar graphs. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2007.

[BGH+95] Hans L. Bodlaender, John R. Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, 1995.

[BHM11] MohammadHossein Bateni, Mohammad Taghi Hajiaghayi, and Dániel Marx. Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth. *J. ACM*, 58(5):21:1–21:37, 2011.

[BKM15] Glencora Borradaile, Philip N. Klein, and Claire Mathieu. A polynomial-time approximation scheme for Euclidean Steiner forest. *ACM Trans. Algorithms*, 11(3):19:1–19:20, 2015.

[Bod88] Hans L. Bodlaender. NC-algorithms for graphs with small treewidth. In Jan van Leeuwen, editor, *Graph-Theoretic Concepts in Computer Science, 14th International Workshop, WG '88, Amsterdam, The Netherlands, June*

15-17, 1988, Proceedings, volume 344 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 1988.

[BPT92] Richard B. Borie, R. Gary Parker, and Craig A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(5&6):555–581, 1992.

[BT97] Dimitris Bertsimas and John Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1st edition, 1997.

[BZ17] Glencora Borradaile and Baigong Zheng. A PTAS for three-edge-connected survivable network design in planar graphs. In Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh S. Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, volume 81 of *LIPIcs*, pages 3:1–3:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[CC10] Parinya Chalermsook and Julia Chuzhoy. Resource minimization for fire containment. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1334–1349. SIAM, 2010.

[CC14] Janka Chlebíková and Morgan Chopin. The firefighter problem: A structural analysis. In Marek Cygan and Pinar Heggernes, editors, *Parameterized and Exact Computation - 9th International Symposium, IPEC 2014, Wroclaw, Poland, September 10-12, 2014. Revised Selected Papers*, volume 8894 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 2014.

[CCC+99] Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed Steiner problems. *J. Algorithms*, 33(1):73–91, 1999.

[CCK08] Tanmoy Chakraborty, Julia Chuzhoy, and Sanjeev Khanna. Network design for vertex connectivity. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 167–176. ACM, 2008.

[CCP+11] Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.

[CCV+10] Leizhen Cai, Yongxi Cheng, Elad Verbin, and Yuan Zhou. Surviving rates of graphs with bounded treewidth for the firefighter problem. *SIAM J. Discret. Math.*, 24(4):1322–1335, 2010.

[CDD+13] Vítor Costa, Simone Dantas, Mitre Costa Dourado, Lucia Draque Penso, and Dieter Rautenbach. More fires and more fighters. *Discret. Appl. Math.*, 161(16-17):2410–2419, 2013.

[CDE+18] Parinya Chalermsook, Syamantak Das, Guy Even, Bundit Laekhanukit, and Daniel Vaz. Survivable network design for group connectivity in low-treewidth graphs. In Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, volume 116 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[CDK+17] Eden Chlamtác, Michael Dinitz, Guy Kortsarz, and Bundit Laekhanukit. Approximating spanners and directed Steiner forest: Upper and lower bounds. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 534–553. SIAM, 2017.

[CDL+17] Parinya Chalermsook, Syamantak Das, Bundit Laekhanukit, and Daniel Vaz. Beyond metric embedding: Approximating group Steiner trees on bounded treewidth graphs. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 737–751. SIAM, 2017.

[CEG+11] Chandra Chekuri, Guy Even, Anupam Gupta, and Danny Segev. Set connectivity problems in undirected graphs and the directed Steiner network problem. *ACM Trans. Algorithms*, 7(2):18:1–18:17, 2011.

[CEK06] Chandra Chekuri, Guy Even, and Guy Kortsarz. A greedy approximation algorithm for the group Steiner problem. *Discret. Appl. Math.*, 154(1):15–34, 2006.

[CFK+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[CG04] Douglas E. Carroll and Ashish Goel. Lower bounds for embedding into distributions over excluded minor graph families. In Susanne Albers and Tomasz Radzik, editors, *Algorithms - ESA 2004, 12th Annual European Symposium, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3221 of *Lecture Notes in Computer Science*, pages 146–156. Springer, 2004.

[CGL15] Parinya Chalermsook, Fabrizio Grandoni, and Bundit Laekhanukit. On survivable set connectivity. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 25–36. SIAM, 2015.

[CGS+04] Artur Czumaj, Michelangelo Grigni, Papa A. Sissokho, and Hairong Zhao. Approximation schemes for minimum 2-edge-connected and biconnected subgraphs in planar graphs. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 496–505. SIAM, 2004.

[Chu12] Julia Chuzhoy. On vertex sparsifiers with Steiner nodes. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 673–688. ACM, 2012.

[CK04] Chandra Chekuri and Amit Kumar. Maximum coverage problem with group budget constraints and applications. In Klaus Jansen, Sanjeev Khanna, José D. P. Rolim, and Dana Ron, editors, *Approximation, Randomization, and Combinatorial Optimization, Algorithms and Techniques, 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2004, and 8th International Workshop on Randomization and Computation, RANDOM 2004, Cambridge, MA, USA, August 22-24, 2004, Proceedings*, volume 3122 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2004.

[CK12] Julia Chuzhoy and Sanjeev Khanna. An $O(k^3 \log n)$-approximation algorithm for vertex-connectivity survivable network design. *Theory Comput.*, 8(1):401–413, 2012.

[CLL+10] Moses Charikar, Tom Leighton, Shi Li, and Ankur Moitra. Vertex sparsifiers and abstract rounding algorithms. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 265–274. IEEE Computer Society, 2010.

[CMZ12] Markus Chimani, Petra Mutzel, and Bernd Zey. Improved Steiner tree algorithms for bounded treewidth. *J. Discrete Algorithms*, 16:67–78, 2012.

[CNP+11] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

[CP05] Chandra Chekuri and Martin Pál. A recursive greedy algorithm for walks in directed graphs. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 245–253. IEEE Computer Society, 2005.

[Cra15] Meggan E. Craft. Infectious disease transmission and contact networks in wildlife and livestock. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 370(1669), 5 2015.

[CSW+00] Shiva Chaudhuri, K. V. Subrahmanyam, Frank Wagner, and Christos D. Zaroliagis. Computing mimicking networks. *Algorithmica*, 26(1):31–49, 2000.

[CV07] Joseph Cheriyan and Adrian Vetta. Approximation algorithms for network design with metric costs. *SIAM J. Discret. Math.*, 21(3):612–636, 2007.

[CV16] Parinya Chalermsook and Daniel Vaz. New integrality gap results for the firefighters problem on trees. In Klaus Jansen and Monaldo Mastrolilli, editors, *Approximation and Online Algorithms - 14th International Workshop, WAOA 2016, Aarhus, Denmark, August 25-26, 2016, Revised Selected Papers*, volume 10138 of *Lecture Notes in Computer Science*, pages 65–77. Springer, 2016.

[CVV06] Joseph Cheriyan, Santosh S. Vempala, and Adrian Vetta. Network design via iterative rounding of setpair relaxations. *Combinatorica*, 26(3):255–275, 2006.

[CVY08] Leizhen Cai, Elad Verbin, and Lin Yang. Firefighting on trees: $(1 - 1/e)$-approximation, fixed parameter tractability and a subexponential algorithm. In Seok-Hee Hong, Hiroshi Nagamochi, and Takuro Fukunaga, editors, *Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings*, volume 5369 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2008.

[CW09] Leizhen Cai and Weifan Wang. The surviving rate of a graph for the firefighter problem. *SIAM J. Discret. Math.*, 23(4):1814–1826, 2009.

[d'E63] F. d'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10(1):98–108, 1963.

[dFR01] Daniela Pucci de Farias and Benjamin Van Roy. Approximate dynamic programming via linear programming. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 689–695. MIT Press, 2001.

[DH07] Mike Develin and Stephen G. Hartke. Fire containment in grids of dimension three and higher. *Discret. Appl. Math.*, 155(17):2257–2268, 2007.

[DHK05] Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Ken-ichi Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 637–646. IEEE Computer Society, 2005.

[DHK11] Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Contraction decomposition in H-minor-free graphs and algorithmic applications. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 441–450. ACM, 2011.

[DHK14] Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Philip N. Klein. Node-weighted Steiner tree and group Steiner tree in planar graphs. *ACM Trans. Algorithms*, 10(3):13:1–13:20, 2014.

[Die12] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[DK99] Yevgeniy Dodis and Sanjeev Khanna. Design networks with bounded pairwise distance. In Jeffrey Scott Vitter, Lawrence L. Larmore, and Frank Thomson Leighton, editors, *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 750–759. ACM, 1999.

[DW71] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

[FBN15] Stefan Fafianie, Hans L. Bodlaender, and Jesper Nederlof. Speeding up dynamic programming with representative sets: An experimental evaluation of algorithms for Steiner tree on tree decompositions. *Algorithmica*, 71(3):636–660, 2015.

[Fei98] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[FF56] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[FGM+06] Lisa Fleischer, Michel X. Goemans, Vahab S. Mirrokni, and Maxim Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 611–620. ACM Press, 2006.

[FHL+00] Stephen Finbow, Bert Hartnell, Q. Li, and K. Schmeisser. On minimizing the effects of fire or a virus on a network. *JCMCC. The Journal of Combinatorial Mathematics and Combinatorial Computing*, 33, 2000.

[FJW06] Lisa Fleischer, Kamal Jain, and David P. Williamson. Iterative rounding 2-approximation algorithms for minimum-cost vertex connectivity problems. *J. Comput. Syst. Sci.*, 72(5):838–867, 2006.

[FKM+07] Stephen Finbow, Andrew D. King, Gary MacGillivray, and Romeo Rizzi. The firefighter problem for graphs of maximum degree three. *Discret. Math.*, 307(16):2094–2105, 2007.

[FKN12] Moran Feldman, Guy Kortsarz, and Zeev Nutov. Improved approximation algorithms for directed Steiner forest. *J. Comput. Syst. Sci.*, 78(1):279–292, 2012.

[FM09] Stephen Finbow and Gary MacGillivray. The firefighter problem: a survey of results, directions and questions. *Australas. J Comb.*, 43:57–78, 2009.

[Fog03] Patricia Fogarty. Catching the fire on grids, 2003.

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004.

[Gas10] Elisabeth Gassner. The Steiner forest problem revisited. *J. Discrete Algorithms*, 8(2):154–163, 2010.

[GHP17] Gramoz Goranci, Monika Henzinger, and Pan Peng. Improved guarantees for vertex sparsification in planar graphs. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 44:1–44:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[GK11] Anupam Gupta and Jochen Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, 16(1):3–20, 1 2011.

[GKR00] Naveen Garg, Goran Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group Steiner tree problem. *J. Algorithms*, 37(1):66–84, 2000.

[GKR10] Anupam Gupta, Ravishankar Krishnaswamy, and R. Ravi. Tree embeddings for two-edge-connected network design. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1521–1538. SIAM, 2010.

[GLL19] Fabrizio Grandoni, Bundit Laekhanukit, and Shi Li. $O(\log^2 k/\log\log k)$-approximation algorithm for directed Steiner tree: a tight quasi-polynomial-time algorithm. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 253–264. ACM, 2019.

[GNR+04] Anupam Gupta, Ilan Newman, Yuri Rabinovich, and Alistair Sinclair. Cuts, trees and l_1-embeddings of graphs. *Combinatorica*, 24(2):233–269, 2004.

[Gor15] Przemyslaw Gordinowicz. Planar graph is on fire. *Theor. Comput. Sci.*, 593:160–164, 2015.

[Gru10] Hermann Gruber. On balanced separators, treewidth, and cycle rank. *CoRR*, abs/1012.1344, 2010.

[GTW13] Anupam Gupta, Kunal Talwar, and David Witmer. Sparsest cut on bounded treewidth graphs: algorithms and hardness results. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 281–290. ACM, 2013.

[Hal76] Rudolf Halin. S-functions for graphs. *Journal of Geometry*, 8(1):171–186, Mar 1976.

[Har95] Bert Hartnell. Firefighter! an application of domination. In *Manitoba Conference on Combinatorial Mathematics and Computing*, 1995.

[Har04] Stephen G. Hartke. Attempting to narrow the integrality gap for the firefighter problem on trees. In James Abello and Graham Cormode, editors, *Discrete Methods in Epidemiology, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, March 18-19, 2004*, volume 70 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 225–231. DIMACS/AMS, 2004.

[He13] M. Hauptmann and M. Karpinski (eds.). A compendium on steiner tree problems, 2013.

[Het00] Herbert W. Hethcote. The mathematics of infectious diseases. *SIAM Review*, 42(4):599–653, 2000.

[HK03] Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 585–594. ACM, 2003.

[HKK+07] Eran Halperin, Guy Kortsarz, Robert Krauthgamer, Aravind Srinivasan, and Nan Wang. Integrality ratio for group Steiner trees and directed Steiner trees. *SIAM J. Comput.*, 36(5):1494–1511, 2007.

[HKN+98] Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *J. Comput. Syst. Sci.*, 57(3):366–375, 1998.

[HKR+11] Stephan Held, Bernhard Korte, Dieter Rautenbach, and Jens Vygen. Combinatorial optimization in VLSI design. In Vasek Chvátal, editor, *Combinatorial Optimization - Methods and Applications*, volume 31 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 33–96. IOS Press, 2011.

[HL00] Bert Hartnell and Qiyan Li. Firefighting on trees: How bad is the greedy algorithm? *Congressus Numerantium*, 145, 01 2000.

[HRZ01] Christopher S. Helvig, Gabriel Robins, and Alexander Zelikovsky. An improved approximation scheme for the group Steiner problem. *Networks*, 37(1):8–20, 2001.

[IKM11] Yutaka Iwaikawa, Naoyuki Kamiyama, and Tomomi Matsui. Improved approximation algorithms for firefighter problem on trees. *IEICE Trans. Inf. Syst.*, 94-D(2):196–199, 2011.

[Jai01] Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[JLR+17] Mark Jones, Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Ondrej Suchý. Parameterized complexity of directed Steiner tree on sparse graphs. *SIAM J. Discret. Math.*, 31(2):1294–1327, 2017.

[JMV+02] Kamal Jain, Ion I. Mandoiu, Vijay V. Vazirani, and David P. Williamson. A primal-dual schema based approximation algorithm for the element connectivity problem. *J. Algorithms*, 45(1):1–15, 2002.

[Kar89] Richard M Karp. A $2k$-competitive algorithm for the circle. Technical report, 1989.

[KKL04] Guy Kortsarz, Robert Krauthgamer, and James R. Lee. Hardness of approximation for vertex-connectivity network design problems. *SIAM J. Comput.*, 33(3):704–720, 2004.

[KKN12] Rohit Khandekar, Guy Kortsarz, and Zeev Nutov. Approximating fault-tolerant group-Steiner problems. *Theor. Comput. Sci.*, 416:55–64, 2012.

[Kle05] Philip N. Klein. A linear-time approximation scheme for planar weighted TSP. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 647–657. IEEE Computer Society, 2005.

[KM10] Andrew D. King and Gary MacGillivray. The firefighter problem for cubic graphs. *Discret. Math.*, 310(3):614–621, 2010.

[KPZ19] Nikolai Karpov, Marcin Pilipczuk, and Anna Zych-Pawlewicz. An exponential lower bound for cut sparsifiers in planar graphs. *Algorithmica*, 81(10):4029–4042, 2019.

[KR13] Robert Krauthgamer and Inbal Rika. Mimicking networks and succinct representations of terminal cuts. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1789–1799. SIAM, 2013.

[KR14] Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Inf. Process. Lett.*, 114(7):365–371, 2014.

[KR17] Robert Krauthgamer and Inbal Rika. Refined vertex sparsifiers of planar graphs. *CoRR*, abs/1702.05951, 2017.

[KV18] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer-Verlag Berlin Heidelberg, 6th edition, 2018.

[Lau03] Monique Laurent. A comparison of the Sherali-Adams, Lovász-Schrijver, and Lasserre relaxations for 0-1 programming. *Math. Oper. Res.*, 28(3):470–496, 2003.

[LMS98] Nathan Linial, Avner Magen, and Michael E Saks. Low distortion euclidean embeddings of trees. *Israel Journal of Mathematics*, 106(1):339–348, 1998.

[LPS19] Yang P. Liu, Richard Peng, and Mark Sellke. Vertex sparsifiers for c-edge connectivity. *CoRR*, abs/1910.10359, 2019.

[Man60] Alan S. Manne. Linear programming and sequential decisions. *Manage. Sci.*, 6(3):259–267, 4 1960.

[Mat99] Jiří Matoušek. On embedding trees into uniformly convex banach spaces. *Israel Journal of Mathematics*, 114(1):221–237, 1999.

[MM10] Konstantin Makarychev and Yury Makarychev. Metric extension operators, vertex sparsifiers and lipschitz extendability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 255–264. IEEE Computer Society, 2010.

[MR01] Cristopher Moore and J. M. Robson. Hard tiling problems with simple tiles. *Discret. Comput. Geom.*, 26(4):573–590, 2001.

[MRC90] R. Kipp Martin, Ronald L. Rardin, and Brian A. Campbell. Polyhedral characterization of discrete dynamic programming. *Oper. Res.*, 38(1):127–138, 1990.

[MSD+96] Milena Mihail, David Shallcross, Nate Dean, and Marco Mostrel. A commercial application of survivable network design: ITP/INPLANS CCS network topology analyzer. In Éva Tardos, editor, *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, 28-30 January 1996, Atlanta, Georgia, USA*, pages 279–287. ACM/SIAM, 1996.

[MW03] Gary MacGillivray and Ping Wang. On the firefighter problem. *JCMCC. The Journal of Combinatorial Mathematics and Combinatorial Computing*, 47, 2003.

[NPS11] Joseph Naor, Debmalya Panigrahi, and Mohit Singh. Online node-weighted Steiner tree and related problems. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 210–219. IEEE Computer Society, 2011.

[Nut10] Zeev Nutov. Approximating Steiner networks with node-weights. *SIAM J. Comput.*, 39(7):3001–3022, 2010.

[Nut12] Zeev Nutov. Approximating minimum-cost connectivity problems via uncrossable bifamilies. *ACM Trans. Algorithms*, 9(1):1:1–1:16, 2012.

[PDGF+15] Nathalie Peyrard, Simon De Givry, Alain Franc, Stephane Robin, Regis Sabbadin, Thomas Schiex, and Matthieu Vignes. Exact and approximate inference in graphical models: variable elimination and beyond. *arXiv preprint arXiv:1506.08544*, 2015.

[PSS10] Stefan Porschen, Tatjana Schmidt, and Ewald Speckenmeyer. Complexity results for linear XSAT-problems. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6175 of *Lecture Notes in Computer Science*, pages 251–263. Springer, 2010.

[RK03] Jonathan M. Read and Matt J. Keeling. Disease evolution on networks: the role of contact structure. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 270(1516):699–708, 4 2003.

[Rot11] Thomas Rothvoß. Directed Steiner tree and the Lasserre hierarchy. *CoRR*, abs/1111.5473, 2011.

[RS84] Neil Robertson and Paul D. Seymour. Graph minors. III. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.

[RW89] Gabriele Reich and Peter Widmayer. Beyond Steiner's problem: A VLSI oriented generalization. In Manfred Nagl, editor, *Graph-Theoretic Concepts in Computer Science, 15th International Workshop, WG '89, Castle Rolduc, The Netherlands, June 14-16, 1989, Proceedings*, volume 411 of *Lecture Notes in Computer Science*, pages 196–210. Springer, 1989.

[SW16] Raazesh Sainudiin and David Welch. The transmission process: A combinatorial stochastic process for the evolution of transmission trees over networks. *Journal of Theoretical Biology*, 410:137 – 170, 2016.

[Tho98] Mikkel Thorup. All structured programs have small tree-width and good register allocation. *Inf. Comput.*, 142(2):159–181, 1998.

[Vaz01] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, Heidelberg, 2001.

[vLV] Erik Jan van Leeuwen and Daniel Vaz. The firefighter problem on bounded-treewidth graphs. Unpublished manuscript.

[WM02] Ping Wang and Stephanie A Moeller. Fire control on graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 41:19–34, 2002.

[WS11] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

Index

Notation

$G[f]$, 9
DAG, 8
FPT, 12
LP, 13
XP, 12

approximation algorithm, 11
  $\alpha(n)$-inapproximable, 12
  EPTAS, 12
  FPTAS, 12
  PTAS, 11
  approximation factor, 11
  approximation ratio, 11
  efficient polynomial-time approximation scheme, 12
  fully-polynomial-time approximation scheme, 12
  polynomial time approximation scheme, 11
arborescence, 8
  in-arborescence, 8
  out-arborescence, 8

cut, 7
  cutset, 7
  mincut, 7
  separate, 7
  weight, 7

directed acyclic graph, 8
  rooted, 8

graph
  arc, 7
    head, 8
    tail, 8
  in-degree, 8
  out-degree, 8
  path, 7
    length, 7
    shortest path metric, 7

linear program, 13
  $1/k$-integral, 14
  constraints, 13
  decision variables, 13
  extreme point solution, 13
  feasible solution, 13
  half-integral, 14
  integer program, 13
  integral, 14
  integrality gap, 14
    against $1/k$-integral solutions, 14
  mixed integer program, 13
  objective function, 13
  optimum solution, 13
  relaxation, 14
  round, 14
  solving, 13
  support, 13
  value, 13

parameterized problem, 12
  fixed-parameter tractable, 12
  slice-wise polynomial, 12
planar graph, 9
  k-outerplanar graph, 9
  faces, 9
  outer face, 9
  outerplanar graph, 9
  plane graph, 9
  subgraph of G induced by f, 9

tree
  ℓ-th ancestor, 8
  d-ary, 8
  ancestor, 8
  children, 8
  depth, 8

height, 8
lowest common ancestor, 8
parent, 8
subtree of G rooted at v, 8
tree decomposition, 9
bag, 9
belongs, 10
node, 10
width, 10
treewidth, 10

Network Design

$\mathrm{tc}(Y)$, 31
$\mathrm{tc}^*_Z(Y)$, 36
$d(F)$, 49
$h(F)$, 49
k-ECSS, 59
DSF, 19
DST, 18
GKR, 21
GSF, 18
GST, 17
GroupSNDP, 58
MDGSF, 49
MDSTGSF, 51
SNDP, 57

connection set, 31
connectivity definition, 33, 36
global connectivity definition, 33, 36
local connectivity definition, 33, 36
connectivity-K mimicking network, 60, 70, 71
terminals, 71
connectivity-K-linked, 71
violating cut, 71
with respect to its neighbors, 71
consistent, 35, 63, 67

directed Steiner forest, 19
terminal pair, 21
directed Steiner tree, 18
terminal, 18
dynamic programming
optimal substructure, 30
state, 30
subinstance, 30

group Steiner forest, 18
group pair, 21
group pairs, 49
minimum-density group Steiner forest, 49
group Steiner tree, 17
group, 17, 20

junction tree, 53

local connectivity definition, 76

metric tree embedding, 17
distortion, 19
mimicking network, 58

restricted group survivable network design problem, 58
demand, 58, 61
groups, 58, 61

solution tree, 40
combination nodes, 40
subproblem nodes, 40
survivable network design problem, 57
demand, 60
terminals, 60

transitive closure, 31

vertex sparsifier, 58
quality-q sparsifier, 59
terminal, 58

Firefighter Problem

(M, k', δ)-good gadget, 91
    LP-friendly:, 91
    Integrally adversarial:, 91
    special vertices, 91
    Uniform depth:, 91
$(\mathcal{U}, \mathcal{B})$-risky, 91
$(\mathcal{U}, \mathcal{B})$-safe, 91
$G_{\phi,B}$, 128, 133
    drones, 129, 133
    gate vertices, 129, 133
T'_ϕ, 132
T_ϕ, 128, 129
η-heavy, 103
η-light, 103
η-separable, 103
$\mathcal{S}$, 91
k-completable, 122
    dispersed, 121
k-weak-completable, 121
CM-1-in-3-SAT, 118
Max-FF, 85, 88
LP-Hartke, 98
Min-FF, 85
Outer-FF, 118
Tree-FF, 88
2-branching, 98

Cubic Monotone 1-in-3-SAT, 118
    assignment, 118
    clause, 118
    variables, 118

delay vertex, 119
    delayed, 119

firefighter problem, 85, 88
    budget, 88
    burning, 85
    protect, 85
    saved, 85, 87
    source, 88

influenced, 120

partially protected, 106
    good, 106

schedulable, 125
spider, 94
    foot, 94
    leg, 94
squad, 121
    save, 121
    separate, 121
strategy, 87
    apply, 87
    budget, 87
    layered, 89
    respecting, 133
    saved, 87
    valid strategy, 87

tour, 122
    appearance, 122
    slice, 122
    subtour, 122