Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-96-01

Ein Multiagentenansatz zum Lösen von Fleet-Scheduling-Problemen

Darius Schier und Klaus Fischer

October 1996

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Sema Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry of Education, Science, Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct systems with technical knowledge and common sense which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- ☐ Intelligent Engineering Systems
- ☐ Intelligent User Interfaces
- ☐ Computer Linguistics
- ☐ Programming Systems
- ☐ Deduction and Multiagent Systems
- ☐ Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Dr. Dr. D. Ruland
Director

Ein Multiagentenansatz zum Lösen von Fleet-Scheduling-Problemen

Darius Schier und Klaus Fischer

DFKI-D-96-01

Inhaltsverzeichnis

1. Einleitung

Verteilte Systeme werden in der Informatik immer wichtiger. Mit der Zunahme an Informationsbedürfnissen in der Gesellschaft steigt auch der Bedarf, schnell auf (möglicherweise physikalisch getrennte) Datenmengen zugreifen zu können. Mit der Notwendigkeit einer ständigen Verfügbarkeit von Systemen werden Fragen der Zuverlässigkeit und Sicherheit immer dringlicher. Der Effizienzzuwachs wird teilweise dadurch erreicht, bestehende Systeme durch modernere und schnellere zu ersetzen. Der eigentliche Gewinn resultiert aber aus der Zuweisung einzelner Teilaufgaben auf verschiedene Komponenten, um so durch nebenläufige Ausführung Zeit zu sparen. Durch entsprechende Organisation solch eines verteilten Systems kann auch erreicht werden, daß bei Ausfall einzelner Komponenten deren Aufgaben durch andere übernommen werden können und so der Grad der Verfügbarkeit des Gesamtsystems erhöht wird. In der Künstlichen Intelligenz (KI) spielt in diesem Zusammenhang vor allem der Aspekt der realitätsbezogenen Modellierung eine Rolle, d.h. die 'intelligente' Anpassung an eine sich ändernde Umwelt.
An der DFKI GmbH wird seit 1991 Forschung im Bereich der Multiagentensysteme betrieben. Aspekte dieser Forschungsarbeit umfassen Fragestellungen, wie sich Probleme auf verschiedene Komponenten (Agenten) aufteilen lassen (Organisation), wie diese Agenten zusammen an der Lösung mitwirken können (Interaktion/Kommunikation) und mit welchen Fähigkeiten und Wissen einzelne Agenten ausgestattet sein müssen, um die Lösungsfindung vorantreiben zu können (interne Strukturen).
In diesem Bericht wird vor allem der zweite Punkt näher untersucht: wie lassen sich Kooperationsstrukturen zwischen Agenten modellieren und welche Vorteile bietet die Verteiltheit gegenüber konventionellen Architekturen. Als Anwendung wurde die Domäne der *Fleet-Scheduling-Probleme* gewählt. Das *Vehicle-Routing-Problem (VRP)* als Standardvertreter einer fast unüberschaubaren Klasse von Fleet-Scheduling-Problemen ist einfach zu formulieren, jedoch schwierig zu lösen. Gegeben ist eine Liste von Kunden mit festen Forderungen, die von einer Fahrzeugflotte mit Kapazitätsbeschränkungen in möglichst optimaler Reihenfolge zu bedienen sind. An der Lösung des Problems sind eine Reihe von unterschiedlichen Sparten interessiert: Operations Research, Mathematik, Informatik und nicht zuletzt unter dem Gesichtspunkt der Wirtschaftlichkeit der 'Verursacher' selbst: die Logistik. Die Aufarbeitung der in der Literatur bekannten Verfahrensweisen zur Lösung von Fleet-Scheduling-Problemen war ein weiterer Schwerpunkt der Arbeit.
Die Modellierung als Multiagentensystem, in dem autonom agierende und reagierende Agenten (Fuhrunternehmen, Fahrzeuge) durch Kooperation (Verteilen und Austauschen von Aufträgen) versuchen, ein gemeinsames Ziel (möglichst optimale Lösung des Problems) zu verwirklichen, eröffnet neue Perspektiven.
Speziell die Reaktionsfähigkeit auf eine sich ständig ändernde Umwelt ist eine Anforderung, für die in dieser Form bisher in der Literatur kaum Lösungen zu finden sind. Konventionelle Scheduler, die in der Realität zur Anwendung kommen, liefern in aller

Regel auf eine Eingabe (sprich Auftragsliste) eine statische Ausgabe (Tourplan). Ändert sich die Auftragslage (beispielsweise durch Forderungen eines neuen Kunden, Staumeldungen oder Belieferungsprobleme) müssen alle bisher geplanten Touren verworfen und vollständig neu berechnet werden. Ist dies nicht möglich, da die Belieferung z.B. schon begonnen hat, muß 'von Hand' weiter geplant werden.
Mit dem hier vorgestellten System besteht die Möglichkeit, an bestehenden Plänen festzuhalten und diese, soweit es Zeitrestriktionen und Ladekapazitäten zulassen, zu modifizieren. Es können prinzipiell sogar Kunden zwischen einzelnen Fahrzeugen während der Ausführung des Tourplans ausgetauscht werden, um so die Gesamtqualität erheblich zu verbessern und letztendlich aus betriebswirschaftlicher Sicht Kosten zu minimieren.
Vergleiche mit bekannten Lösungen aus der Literatur werden ebenso angestellt wie Perspektiven aufgezeigt, inwiefern das System erweiterbar ist. So erlaubt beispielsweise die Strukturierung des Multiagentensystems eine physikalische Verteilung der einzelnen Agenten auf verschiedene Systeme. Auch dies ist ein weiterer Schritt in Richtung realitätsbezogene Modellierung, wobei jedes Fahrzeug als lokale Planungseinheit angesehen werden kann. Innerhalb dieser Einheit lassen sich wiederum aus der KI bekannte Standardverfahren einsetzen, um jede einzelne Tour und somit den Gesamtplan aller Agenten zu verbessern.

Gliederung

In Kapitel 2 werden theoretische Vorüberlegungen angestellt und Grundlagen geschaffen, die zum Verständnis der Gesamtproblematik und Einordnung des Problems in größerem Kontext notwendig sind. Das Problem selbst wird in Kapitel 3 charakterisiert und aus der Literatur bekannte Lösungsansätze in Kapitel 4 aufgezeigt. Der Hauptteil der Arbeit ist in den Kapiteln 5 und 6 zu finden. Hier wird das Speditionsszenario und dessen Arbeitsweise vorgestellt, sowie Vergleiche der Qualität der Lösungen mit aus der Literatur bekannten Verfahren angestellt. In Kapitel 7 wird aufgezeigt, inwieweit das System erweiterbar ist und wo Ansätze zur Verbesserung bestehen. Im Anhang wird detailliert auf die Implementierung in der Programmiersprache *Oz* eingegangen.

2. Grundlagen und Begriffe

Zu Beginn soll das 'Handwerkszeug' zurecht gelegt werden, mit dessen Hilfe wir die noch zu untersuchenden Problemstellungen bearbeiten werden. Im ersten Teil soll ein gewisses Grundverständnis für die in diesem Bericht vorgestellten Vorgehensweisen geschaffen werden. Im zweiten Teil werden wir uns mit Gebieten der Theoretischen Informatik beschäftigen, die notwendig sind, um die Probleme und deren 'Hartnäckigkeit' in einem größeren Kontext einordnen zu können.

2.1. Verteilte Künstliche Intelligenz und Multiagentensysteme

Ein relativ junger Zweig der Künstlichen Intelligenz (KI), die sogenannte *Verteilte Künstliche Intelligenz (VKI)* (engl. Distributed Artificial Intelligence (DAI)) erfährt in den letzten Jahren zunehmend an Aufmerksamkeit. Ziel dieses Teilbereichs ist es, autonome Systeme (Agenten) mit eigenem Wissen und Können miteinander kooperieren zu lassen, um so komplexe Aufgabenstellungen bewältigen zu können, zu denen einzelne Entitäten im allgemeinen nicht oder nur unzureichend (z.B. mit sehr hohem Zeitaufwand) fähig sind. Die Einheiten stehen in aller Regel nur mittels Kommunikationskanälen miteinander in Verbindung. Mit diesem Ansatz versucht man, Problemlösestrategien zu modellieren, die der Realität nahe kommen.
Beispiele aus der Biologie lassen sich leicht finden, wie z.B. der Bienen- oder Ameisenstaat. Jedes einzelne Individuum wäre kaum überlebensfähig, die Verteilung von verschiedenen Aufgaben an 'Spezialisten' und die Organisation zu einem Ganzen jedoch führt zu einem System, in dem letztendlich erstaunliche Intelligenzleistungen erbracht werden und das globale Ziel verwirklicht wird: der Schutz jedes einzelnen.
Auf diesen Ansatz läßt sich die Aussage der Gestalt-Schule übertragen (die sich allerdings in diesem Zusammenhang auf Mustererkennung bezog)[1]:

> Das Ganze ist mehr als die Summe seiner Teile.

[Minsky, 1985] geht in seiner Theorie der *Society of Mind* von der Annahme aus, das Gehirn an sich sei bereits ein verteiltes System, eine Ansammlung heterogener Agenten, die miteinander kooperieren und konkurrieren. Einen schönen Überblick über das Gebiet der VKI kann man mit Hilfe von [Durfee, 1991], [Müller, 1993] oder [Schupeta, 1992] gewinnen.

[1] Die Gestalt-Schule war eine Gruppe deutscher Psychologen, die zwischen den beiden Weltkriegen in die USA emigrierten. Sie beschäftigten sich vor allem mit Mustererkennung und Problemlösen, siehe auch [Köhler, 1925, Köhler, 1947, Wertheimer, 1958].

2.1.1. Merkmale

Im Folgenden werden die Eigenschaften verteilter Systeme näher erörtert. Dabei werden sowohl Vorteile als auch die Problematik betrachtet, die der Übergang von 'konventionellen' KI-Architekturen zu verteilt/nebenläufigen Paradigmen mit sich bringt. Die Darstellung hält sich im wesentlichen an [Rich and Knight, 1991].
Gegenüber großen, statischen Anwendungen haben verteilte Problemlösungssysteme eine Reihe signifikanter Vorteile:

- *Modularität* - Kleine, quasi-unabhängige Module lassen sich leichter implementieren und warten als große.

- *Effizienz* - Das vollständige Wissen ist nicht für alle Aufgaben notwendig. Durch Modularisierung gewinnt man die Fähigkeit, sich auf den Problemlöseaufwand zu beschränken, der sich am wahrscheinlichsten bewährt.

- *Nutzbarmachen verteilter/paralleler Hardware* - Einzelne Komponenten verteilter Systeme können auch physikalisch verschiedenen Rechnern zugewiesen werden. Obwohl Computer immer schneller werden, stammt der eigentliche Effizienzzuwachs nicht von immer größeren und schnelleren Prozessoren, sondern wird durch die Verteilung auf 'kleine' Prozessoren mit eigenem Speicher vorangetrieben.

- *Heterogenes Problemlösen* - Problemlösetechniken und Formalismen der Wissensrepräsentation, die ein Teil eines Problems gut lösen, arbeiten nicht zwingend gleich gut bei anderen Teilproblemen.

- *Verschiedene Sichtweisen* - Das zum Lösen eines Problems benötigte Wissen muß nicht im Kopf einer Person (eines Agenten) liegen. Es ist extrem schwierig, für verschiedene Personen ein kohärentes Wissen sicherzustellen, manchmal sogar unmöglich, da ihre Modelle bzgl. der Domäne inkonsistent sind.

- *Physikalisch verteilte Probleme* - Einige Probleme sind 'echt' verteilt, d.h. verschiedene Informationen, die benötigt werden, liegen an physikalisch getrennten Orten.

- *Zuverlässigkeit* - Ist ein Problem über mehrere Agenten auf verschiedenen Systemen verteilt, kann das Problemlösen selbst dann fortfahren, wenn ein System fehlschlägt.

Eine Architektur für verteilte Problemlösung muß die folgenden Eigenschaften aufweisen:

- einen Mechanismus, der garantiert, daß die Aktivitäten der einzelnen Agenten im System koordiniert werden, so daß für das globale Ziel eine Lösung gefunden wird;

- eine Kommunikationsstruktur, die es erlaubt, Informationen (Nachrichten) zwischen Agenten auszutauschen;

- verteilte Versionen der notwendigen Lösungstechniken.

Speziell die ersten beiden Eigenschaften sollen im folgenden Abschnitt näher beleuchtet werden.

2.1.2. Koordination und Kooperation

Ein zentrales Problem beim Entwurf verteilter Lösungssysteme ist die Koordination von Agenten, so daß diese effektiv zusammen arbeiten:

Wer tut was wann ?

Es gibt eine Reihe von Ansätzen zur Lösung dieses Problems. Die ersten beiden der nun folgenden Vorschläge können unter dem Begriff *vertikale Kooperation* zusammengefaßt werden, die beiden letzten charakterisieren die *horizontale Kooperation.*

- Ein Agent ist übergeordnet (*master*). Er erstellt einen Plan, den er in Stücken an andere Agenten (*slave*) weitergibt, die wiederum das tun, was ihnen aufgetragen wird und ihre Ergebnisse zurückgeben. Um das Gesamtziel zu erreichen, können diese Slave-Agenten auch mit anderen kommunizieren.
- Ein Agent ist übergeordnet, der das Problem in Teilprobleme zerlegt. Über die Verantwortlichkeit, welches dieser Teilprobleme von welchem Agenten gelöst werden soll, treten diese untereinander in Verhandlungen.
- Ist kein Agent übergeordnet, obwohl alle Agenten das gleiche Ziel verfolgen, müssen diese bzgl. des Findens und Ausführens eines Plans miteinander kooperieren.
- Im Extremfall ist kein Agent übergeordnet, auch wird nicht garantiert, daß ein einzelnes Ziel von allen Agenten geteilt wird. Sie können untereinander sogar konkurrieren.

Die letztendlich verwirklichte Modell in einem Multiagentensystem ist oft eine Mischung von verschiedenen Ansätzen, was auch in der vorliegenden Arbeit noch deutlich wird.
Der erste Ansatz wird auch *zentralisierte Multiagentenplanung* (engl. multi-agent planning) genannt. Einer der bekanntesten Vertreter der zweiten Ausprägung ist das *Contract-Net-Protocol.* Das Aufteilen eines globalen Problems in Teilprobleme ist in der Literatur unter dem Begriff *Task-Decomposition* bekannt, das Zusammenfügen resultierender Teillösungen zu einer globalen heißt *Task-Reallocation.*
In den beiden folgenden Abschnitten soll die vertikale Kooperation näher charakterisiert werden. Hauptmerkmal ist hier die hierarchische Struktur.

Zentralisierte Multiagentenplanung

Die am wenigsten gestreute Form verteilten Problemlösens ist die zentralisierte Multiagentenplanung. Dabei teilt ein übergeordneter Agent das zu erreichende Ziel in Unterziele und weist diese dann den verschiedenen anderen Agenten zu. Idealerweise resultiert die Zerlegung in eine Menge von Teilproblemen, die unabhängig voneinander sind. Dies ist jedoch nicht immer möglich.
Nach der Dekomposition müssen die Teilprobleme den verfügbaren Agenten zur Ausführung zugeordnet werden. Sind die einzelnen Agenten verschieden, muß der Master-Agent Zugriff auf Informationen der Fähigkeiten der verschiedenen Slaves haben. Diese Informationen machen es möglich, die Teilprobleme den Agenten zuzuordnen, die es am besten lösen können. Sind alle Slave-Agenten identisch, muß der Master auf deren Auslastung achten, damit das globale Ziel so schnell wie möglich erreicht wird. Wurde eine Aufgabe aufgeteilt, ist eine Synchronisation der daran beteiligten Agenten erforderlich,

sofern es sich nicht um vollkommen unabhängige Ziele handelt. Dies muß zur Laufzeit geschehen, da die von den einzelnen Agenten benötigte Zeit in aller Regel nicht vorhersehbar ist.

Beispiel 2.1 Sei die Aufgabe gegeben, ein mehrere Kapitel umfassendes Dokument auf Schreibfehler hin zu untersuchen und in entsprechender Reihenfolge auszudrucken. Das Problem kann auf mehrere 'Korrektur-Agenten' und einen 'Druck-Agenten' verteilt werden. Um das gewünschte Ergebnis zu erhalten ist es notwendig, daß kein Kapitel ausgedruckt wird, bevor dessen Korrektur nicht beendet ist und alle vorhergehenden Kapitel ausgedruckt wurden.

Contract-Net-Protocol

Das von [Davis and Smith, 1983] vorgestellte Contract-Net-Protocol ist eines der Standardverfahren der VKI. Dabei kann ein Agent zwei Rollen annehmen:

- Der *Manager* zerlegt ein Problem in Teilprobleme, delegiert diese an die sogenannten *Kontrahenten* (engl. contractor) bzw. Bieter (engl. bidder) und beobachtet/synchronisiert die Planausführung.
- Der Bieter führt eine Teilaufgabe aus, möglicherweise durch direkte Ausführung, möglicherweise dadurch, selbst ein Manager zu werden und wiederum Teile seines Auftrags an andere Agenten weiterzuleiten.

Manager und Bieter finden einander durch eine Art Handel:

- Ein Manager bietet eine Aufgabe an,
- die Bieter bewerten diese aufgrund ihrer eigenen Fähigkeiten und benötigten Ressourcen,
- die Bieter machen dem Manager ein entsprechendes Angebot,
- der Manager wählt einen einzelnen Bieter aus und gibt ihm den Zuschlag für die Aufgabe.

Ein Agent kann sowohl Manager als auch Bieter sein, d.h. während er auf Ergebnisse seiner Slave-Agenten wartet, kann er mit anderen Aufgaben fortfahren, statt nur untätig zu verharren.

Beispiel 2.2 Das Verhandlungsprotokoll ist Hauptbestandteil des Speditionsszenarios, wie es in Kapitel 5 vorgestellt wird. Speziell Kapitel 5.1.1 kann als Instanz für den hier dargestellten Mechanismus angesehen werden.

Horizontale Kooperation

Im Gegensatz zur vertikalen Kooperation gibt es bei der horizontalen Kooperation keine übergeordneten Agenten, die die Aktionen anderer bestimmen. In extremen Ausprägungen solch eines Systems kann man nicht einmal Annahmen über das Verhalten der einzelnen Agenten machen. Ohne solche Annahmen ist es jedoch schwierig, Problemlöse-Algorithmen zu konstruieren. Der Begriff der *Rationalität* beim Menschen läßt sich analog auf Agenten übertragen, d.h. der Agent handelt immer ganau 'richtig', um sein Ziel zu

erreichen. Da aber rationales Handeln auch von einer Reihe von Faktoren abhängen kann (fehlende Informationen, unzureichende Ressourcen), wird der schwächere, aber nützlichere Begriff der *beschränkten Rationalität* (engl. bounded rationality) eingeführt. Der Agent handelt richtig, um ein Ziel zu erreichen, sofern es seine beschränkten Ressourcen zulassen.
Horizontale Kooperation verlangt im allgemeinen eine Verhandlungsfähigkeit der einzelnen Agenten untereinander. Dies spielt sowohl bei der Kooperation als auch bei konkurrierendem Verhalten eine Rolle. So kann beispielsweise ein Agent einen anderen 'bewußt' in die Irre führen, um sich selbst einen Vorteil zu verschaffen. Sehr interessante Ansätze dieser Art findet man beispielsweise in [Zlotkin and Rosenschein, 1993]. Hier spielen verstärkt Aspekte der Agentenmodellierung eine Rolle, wobei der einzelne Agent bezüglich seiner Umwelt möglicherweise unvollständiges (beschränktes) oder sogar inkonsistentes Wissen hat.

2.1.3. Kommunikation

Prinzipiell lassen sich die Kommunikationsarchitekturen in zwei Klassen einteilen. Obwohl beide Techniken oberflächlich betrachtet sehr verschieden sind, leisten sie in der Praxis bei der Unterstützung von Multiagentensystemen essentiell das Gleiche. Sie können sich sogar gegenseitig simulieren.

Blackboard-Systeme: Die Kommunikation wird hier über eine gemeinsame Wissensstruktur, das sogenannte Blackboard ((Wand)Tafel) vollzogen. Einzelne Agenten können 'Notizen' auf dieser Tafel hinterlassen oder Nachrichten lesen, die von anderen abgelegt wurden. Durch gelesene Nachrichten können entsprechende Verhaltensmuster ausgelöst werden. Durch diese Methodik können Agenten miteinander Informationen austauschen, ohne voneinander Kenntnis zu haben. Eines der ersten Blackboard-Systeme, das in großen KI-Applikationen verwirklicht wurde, war im HEARSAY-II Projekt, mit Hilfe dessen natürliche Spracherkennung modelliert wurde [Erman et al., 1980].

Message-Passing Systeme: Message-Passing Systeme verlangen im allgemeinen mehr Wissen der einzelnen Agenten übereinander, als Blackboard-Systeme. Dieses Wissen erlaubt es ihnen jedoch, direkt in Kontakt miteinander zu treten und Informationen an die Stellen weiterzuleiten, die voraussichtlich am meisten zur Lösung des Problems beitragen können.

2.2. Scheduling-Probleme

In diesem Kapitel werden wir uns allgemein der Klasse von Scheduling-Problemen zuwenden und diese charakterisieren. All diese Probleme haben eine gewisse Abhängigkeit von zeitlichen Faktoren gemein (engl. schedule = Fahrplan). Typische Anwendungsgebiete wären z.B. automatische Kontrollsysteme in der Verfahrenstechnik, die Planung der optimalen Prozessorauslastung bei Computern [Peterson and Silberschatz, 1994] oder das Generieren optimaler Strecken im Transportwesen während der Fahrt. Die folgenden Ausführungen orientieren sich im wesentlichen an [Herrtwich, 1990].
Aufgabe des Scheduling ist es, Prozesse oder Ziele eines Systems mit den vorhandenen Ressourcen zu versorgen, die zu ihrer Ausführung notwendig sind. Dabei basiert jede

Scheduling-Methode auf einem Systemmodell, wie z.B. Annahmen über

- vorhandene Ressourcen,
- zu planende Prozesse,
- Scheduling-Ziele.

Eine *Ressource* ist jede Systemkomponente, die ein Prozeß für seine Ausführung benötigt. Die Limitierung von Ressourcen (deren Kapazität) und deren Nutzbarmachung macht Scheduling notwendig. Dabei können Ressourcen auch von mehreren Prozessen gleichzeitig genutzt werden (z.B. Lesen von Daten aus einem Read Only Memory).

2.2.1. Scheduling als Suche im Baum

Scheduling kann als Suche nach einer Kombination von Zeitabschnitten angesehen werden, die es allen Prozessen erlaubt, ihre Zeitrestriktionen einzuhalten. Der Suchraum kann, wie in Abbildung 2.1 illustriert, als Baum strukturiert werden.

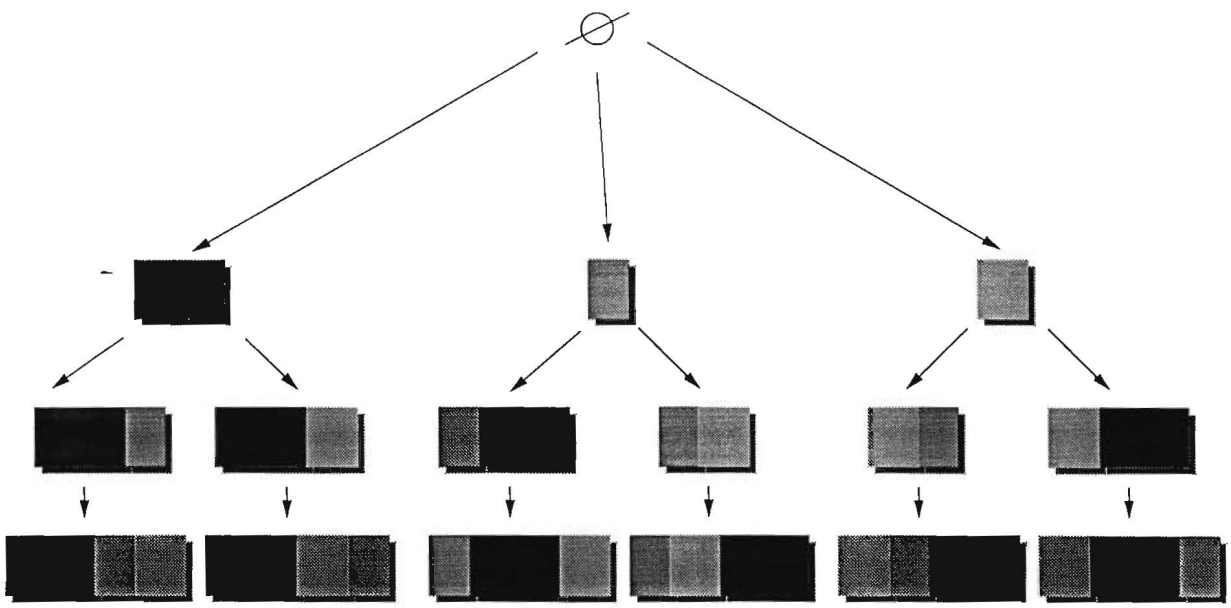

Abbildung 2.1.: Scheduling als Baumsuche

Die Wurzel des Baumes repräsentiert einen leeren Plan. Jeder einzelne Knoten stellt einen partiellen Plan dar, der in seinem direkten Folgeknoten durch einen Zeitabschnitt erweitert wird. Die Blätter des Baumes repräsentieren einen vollständigen Plan. In der Regel stellen nicht alle dieser Blätter eine gültige Verteilung dar, da mitunter Zeitrestriktionen der einzelnen Prozesse in bestimmten Reihenfolgen nicht eingehalten werden können. Aufgabe des Schedulers ist es, in dem Baum nach einem Blatt zu suchen, das einen gültigen Plan darstellt. Den Prozessen können fortlaufend Nummern zugewiesen werden, in welcher Reihenfolge diese später abgearbeitet werden sollen. Die Aufzählung aller möglichen Lösungen nennt man *Enumeration*.

2.2.2. Echtzeit-Scheduling

Ein System wird nach [Stankovic and Ramamrithan, 1988] *Echtzeitsystem* (engl. real-time system) genannt, wenn seine Korrektheit nicht nur durch seine Ausgabewerte bestimmt wird, sondern auch durch den Zeitpunkt, zu dem diese Werte verfügbar sind. Solche Systeme werden benötigt, wenn eine Berechnung auch den Zeitbedarf der Systemumgebung in Betracht ziehen muß.
Die in den 'Fahrplan' einzufügenden Einheiten eines Softwaresystems sind die *Prozesse* (je nach System auch Ziele oder *thread* genannt). Zeitrestriktionen eines Prozesses im Echtzeitsystem werden durch eine oder mehrere Zeitparameter beschrieben, die in Abbildung 2.2 illustriert werden.

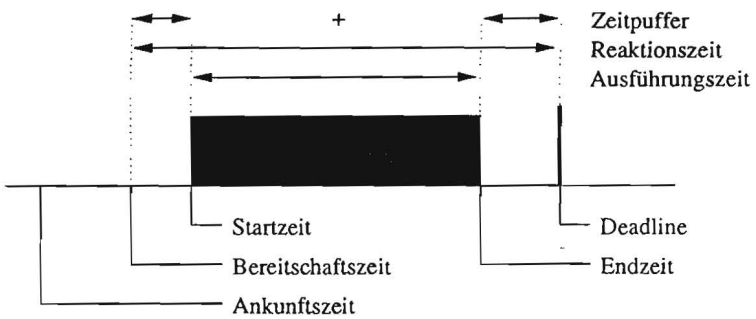

Abbildung 2.2.: Zeitparameter eines Prozesses im Echtzeitsystem

Die *Ausführungszeit E* legt fest, wie lange ein Prozeß Zugang zu einer Ressource benötigt. Die *Ankunftszeit A* definiert den Zeitpunkt, zu dem der Prozeß dem Scheduler bekannt wird, d.h. den Zeitpunkt, zu dem der Scheduler mit dem Planungsvorgang beginnen kann. Die *Startzeit S* ist die Zeit, zu der der Prozeß Zugang zu der Ressource erhält. Die *Endzeit C* gibt das zeitliche Ende eines Prozesses an. Der letzte 'sinnvolle' Zeitpunkt, zu dem ein Prozeß beendet sein muß, wird durch die *Deadline D* festgelegt. Die Zeit zwischen der *Bereitschaftszeit* und der Deadline eines Prozesses wird mit *Reaktionszeit* bezeichnet. Eine Generalisierung der Deadline stellen Wertefunktionen dar, wie beispielsweise in Abbildung 2.3 dargestellt.

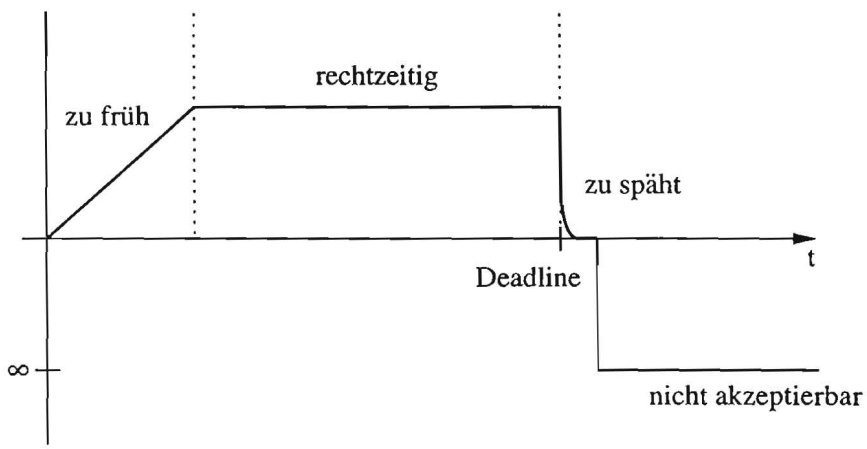

Abbildung 2.3.: Nutzen einer Prozeßausführung in Abhängigkeit der Zeit

Reaktionszeit ohne Ausführungszeit bestimmt den *Zeitpuffer* L eines Prozesses. Dieser ist der maximale Zeitraum, um den der Beginn eines Prozesses verschoben werden kann.

2.3. Komplexitätsmaße

Eine Einführung in die Rekursions- und Komplexitätstheorie würde den Umfang dieser Arbeit sprengen (siehe dazu [Hopcroft and Ullman, 1993] oder [Papademitriou, 1994]). Vielmehr wird versucht einen intuitiven Überblick über die Komplexitätsklassen zu geben, die Gegenstand der Theoretischen Informatik sind und im Kontext dieser Arbeit wichtig sind.

2.3.1. P und NP

Für die vorliegende Arbeit sind vor allem zwei Komplexitätsklassen interessant:

$$P := \bigcup_k DTIME(n^k) \tag{2.1}$$

$$NP := \bigcup_k NTIME(n^k) \tag{2.2}$$

P ist die Klasse der Probleme, die auf einer 'realen' (deterministischen) Rechenmaschine in polynomieller Zeit ($c \cdot n^k, c = const$) berechnet werden können. NP hingegen ist die Klasse von Problemen, die in polynomieller Zeit auf nichtdeterministischen Maschinen gelöst werden können. Diese zeichnen sich dadurch aus, daß sie bei gleicher Eingabe verschiedene Ausgaben produzieren können (die Ausgabe also nicht deterministisch festzulegen ist). Nichtdeterministische Maschinen erheben jedoch eher einen theoretischen Anspruch und werden physikalisch durch deterministische simuliert. Solch eine Simulation nimmt jedoch nach heutigem Wissensstand exponentielle Zeit in Anspruch[2]. Ein Vertreter dieser Klasse soll im nächsten Kapitel näher untersucht werden.
L heißt $NP-vollständig$, wenn $L \in NP$ und jedes Problem $L' \in NP$ auf L polynomiell reduzierbar ist. Die Zugehörigkeit eines Problems zu der Klasse der NP-vollständigen Probleme ist in jedem Falle Evidenz für ein sehr 'hartnäckiges' Problem.

2.3.2. Das Problem des Handlungsreisenden

Wir beginnen mit einem dem Problem des Handlungsreisenden (engl. *Traveling Salesman Problem (TSP)*), verwandten Problem, dem *Hamilton-Zyklus-Problem*, das nach [Hopcroft and Ullman, 1993] wie folgt formuliert werden kann:

> *Hamilton-Zyklus-Problem*: Gegeben sei ein Graph G. Hat G einen Pfad, der jeden Knoten genau ein Mal besucht und dann zu seinem Ausgangspunkt zurückkehrt?

Das *gerichtete Hamilton-Zyklus-Problem* ist das analoge Problem für gerichtete Graphen.

[2] Das Problem, ob $P = NP$ oder $P \subset NP$ ist noch nicht gelöst. D.h. es wurde noch kein Algorithmus gefunden, der ein NP-vollständiges Problem in polynomieller Zeit auf einer deterministischen Maschine löst. Im allgemeinen wird jedoch die zweite Behauptung, daß P eine echte Teilmenge von NP darstellt, angenommen.

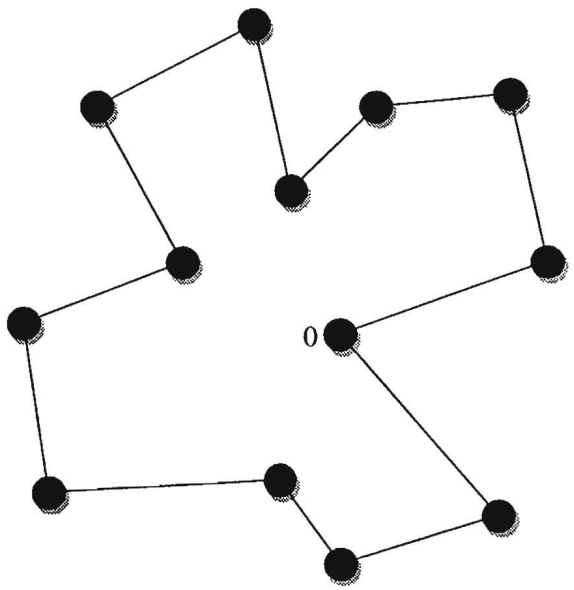

Abbildung 2.4.: Das Problem des Handlungsreisenden

Satz 2.3 *Das (gerichtete) Hamilton-Zyklus-Problem ist NP-vollständig.*

Zum Beweis des Satzes sei auf [Hopcroft and Ullman, 1993] verwiesen.
Das Problem des Handlungsreisenden läßt sich mit Hilfe der obigen Ausführung wie folgt definieren:

> *Problem des Handlungsreisenden*: Gegeben ist ein vollständiger Graph $G = (V, E)$ mit Kantengewichtungen $\omega : E \to \mathbb{N}$. Gesucht ist der Hamilton-Zyklus mit dem geringsten Gewicht.

Das Problem läßt sich auch anschaulicher formulieren, wobei die Herkunft des Namens einsichtig wird: Ein Handlungsreisender soll nacheinander n Städte genau ein Mal besuchen. Die Reise soll wieder am Ausgangspunkt enden. Gesucht wird nach einem Rundweg, dessen Gesamtentfernung minimal ist.

Satz 2.4 *Das Problem des Handlungsreisenden ist NP-vollständig.*

Der Beweis dieses Satzes ist in [Lewis and Papademitriou, 1981] nachzulesen.

3. Problemstellung

In diesem Kapitel werden eine Reihe von Standard-Routing- und Scheduling-Problemen aus der Literatur vorgestellt. Prinzipielles Ziel bei der Lösung eines solchen Problems ist es, mit Hilfe einer Anzahl von Fahrzeugen (Vehicle) eine Reihe von Kunden zu bedienen, wobei zum einen die dabei anfallenden Kosten minimal gehalten werden sollen, zum anderen, je nach Variante des Problems, Restriktionen bzgl. Ladekapazitäten, Be- und Entladezeitfenster, ... eingehalten werden müssen.

3.1. Das Vehicle-Routing-Problem VRP

Das Vehicle-Routing-Problem stellt das eigentliche Standardproblem dar, über das in der Literatur bereits in den 50/60er Jahren erste Arbeiten veröffentlicht wurden ([Dantzig and Ramser, 1959],[Clarke and Wright, 1964]).
Die Grundlage des Problems bildet ein Graph G_R, der ein zu bedienendes Kundennetz mit einem Depot verbindet:

Definition 3.1 (Routing-Graph) *Ein Routing-Graph ist ein (im allgemeinen ungerichteter) Graph $G_R = (V \cup \{0\}, E)$, mit 0 als ausgezeichnetem Depot:*

$$\begin{aligned} i \in V &\Leftrightarrow c_i \text{ ist Kunde,} \\ w_{ij} \in E &\Leftrightarrow \text{Kante } w_{ij} \text{ verbindet Kunde } i \text{ mit } j\text{, bzw. Depot 0;} \\ & \quad i, j \in V \cup \{0\} \end{aligned}$$

Bei einem Routing-Graph wird in der Regel von einer euklidischen Einbettung in der Ebene ausgegangen, d.h. eventuelle X- und Y-Koordinaten werden als solche im zweidimensionalen Raum interpretiert. Über eine Distanzmatrix D wird jedem $w_{ij} \in E$ eine Gewichtung d_{ij} zugeordnet, die die Entfernung zwischen Kunde i und j repräsentiert. Analog dazu kann eine Zeitmatrix Z die Zeit t_{ij} definieren, die zum Übergang von Kunde i nach j entlang der Kante w_{ij} benötigt wird.

Definition 3.2 (Auftrag) *Unter einem Auftrag a_i bzgl. des VRP versteht man die Quantität an Transportgütern, die Kunde i fordert.*

Die Beschaffenheit des Transportgutes soll uns hier nicht weiter interessieren (wie z.B. Flüssigkeiten, Feststoffe, Gase), da bei dem Standard-VRP von einem homogenen Fuhrpark bzw. einer homogenen Auftragsmenge ausgegangen wird, die keine weitere Differenzierung erfordert. Erweiterungen sind in Kapitel 3.4 zu finden.

Beispiel 3.3 Die folgende Datenmenge[1] definiert ein Vehicle-Routing-Problem bezüglich Routing-Graph und Auftragsmenge hinreichend. Dabei handelt es sich um einen vollständig verknüpften Graphen.

CUST NO.	XCOORD.	YCOORD.	DEMAND
1	35.00	35.00	0.00
2	41.00	49.00	10.00
3	35.00	17.00	7.00
4	55.00	45.00	13.00
5	55.00	20.00	19.00
...			

Da hier die X- und Y-Koordinaten im geometrischen Raum interpretierbar sind, ergibt sich als Distanz d_{ij} zwischen zwei Kunden i und j die Entfernung in der Euklid'schen Ebene:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

Ziel des Vehicle-Routing-Problems ist es, mit einer (im allgemeinen beliebig großen) Fahrzeugflotte einen optimalen, gültigen Tourplan $\mathcal{T}$ zu erstellen, der aus m Rundreisen (Touren) besteht. Bei jeder Tour sind also die Ladekapazität Q_i, sowie die maximal erlaubte Einsatzzeit $DeadLine_i$ des ausführenden Fahrzeugs zu beachten, die zu keiner Zeit der Tour überschritten werden dürfen.

Definition 3.4 (Tour) *Sei $G_R = (V \cup \{0\}, E)$ ein Routing-Graph, $V = \{1, \ldots, n\}$ eine Menge von Kunden, 0 das Depot. Eine* ***Tour*** *ist ein Tupel $T_i = (i_0, \ldots, i_k), i_j \in V \cup \{0\}, i \in \{1, \ldots, m\}, |T_i| = k$.*
Sei $Q_i \in \mathbb{N}$ die maximale Ladekapazität von Tour T_i, $DeadLine_i \in \mathbb{N}$ der Zeitpunkt, zu dem Tour T_i spätestens beendet sein muß. T_i ist ***gültig*** *bzgl. des Standard-VRP, falls gilt:*

$$i_0 = i_k = 0 \tag{3.1}$$

$$p \neq q \Leftrightarrow i_p \neq i_q \qquad p, q \in \{1, \ldots, k\} \tag{3.2}$$

$$w_{i_l i_{l+1}} \in E \qquad l \in \{0, \ldots, k-1\} \tag{3.3}$$

$$\sum_{l=1}^{k-1} a_{i_l} \leq Q_i \tag{3.4}$$

$$\sum_{l=1}^{k-1} t_{i_l i_{l+1}} \leq DeadLine_i \tag{3.5}$$

Wir schreiben auch $i_p \overset{i}{\vdash} i_q$ falls Kunde i_q in Tour T_i direkt nach i_p bedient wird, bzw. $i_p \vdash i_q$ falls die Tour aus dem Kontext ersichtlich ist. Die Menge der an einer Tour T_i beteiligten Kunden wird mit $\Pi T_i = \{i_1, \ldots, i_{k-1}\}$ denotiert.

Term (3.1) besagt, daß eine gültige Tour im Depot beginnen und enden muß, (3.2) formalisiert die Aussage, daß die Kunden auf einer gültigen Tour paarweise verschieden sind, d.h. kein Kunde zweimal bedient wird. Knoten i_{l+1} muß von Knoten i_l aus erreichbar sein (3.3), die Summe der von den Kunden geforderten Mengen darf die Gesamtkapazität des Fahrzeugs nicht übersteigen (3.4), die Gesamtfahrzeit des Fahrzeuges darf die $DeadLine_i$ nicht überschreiten (3.5).

[1] Es handelt sich hierbei um einen Auszug des Problems r101, das Bestandteil von einer Reihe von Tests ist, die in Kapitel 6.1 näher beschrieben werden.

Definition 3.5 (Tourplan) *Ein* ***Tourplan*** $\mathcal{T} = (T_1, \ldots, T_m)$ *ist eine Partitionierung von* V. $\mathcal{T}$ *ist gültig, falls*

$$T_i \text{ gültig} \qquad \forall i \in \{1, \ldots, m\} \tag{3.6}$$

$$\bigcup_{i=1}^{m} \Pi T_i = V \tag{3.7}$$

$$\Pi T_i \cap \Pi T_j = \emptyset \quad \forall i, j \in \{1, \ldots, m\}, i \neq j \tag{3.8}$$

Im Depot werden zu Beginn der Rundreise Güter aufgeladen, die im Laufe der Tour auf die Kunden entsprechend ihren Forderungen aufgeteilt werden.
Der Begriff der Optimalität eines Tourplans ist stark mit dem Kostenbegriff verbunden.

Definition 3.6 (Kosten einer Tour) *Die* ***Kosten*** *einer Tour sind gegeben durch eine gewichtete Funktion* $c : 2^V \rightarrow \mathbb{R} \cup \{\infty\}$. *Die Kosten eines Tourplans sind somit durch* $\sum_{i=1}^{m} c(T_i)$ *festgelegt.*

Je nach Lösungsverfahren (vgl. Kapitel 4) wird eine entsprechende Kostenfunktion gewählt. Dabei stehen vor allem räumliche und zeitliche Aspekte im Vordergrund. Die Kostenfrage wird dabei im allgemeinen auf die variablen Kosten beschränkt. Die aus betriebswirtschaftlicher Sicht ebenso anfallenden Fixkosten werden bei dem Standard-VRP in aller Regel vernachlässigt.

3.2. Zeitrestriktionen im VRPTW

Das Standard-VRP kann durch Hinzunahme von Zeitrestriktionen bzgl. jedem einzelnen Kunden zu dem sogenannten *VRPTW* (engl. *Vehicle Routing Problem with Time Windows*) erweitert werden. In der Literatur findet man auch die Abkürzungen VRSPTW (Vehicle Routing and Scheduling Problem with Time Windows, [Solomon, 1987]) oder TCVRP (Time-Constrained Vehicle Routing Problem [Kolen et al., 1987]). Man spricht von *routing*, falls ausschließlich räumliche Aspekte eine Rolle spielen, *scheduling* beinhaltet immer eine zeitliche Abhängigkeit.
Das VRPTW stellt eine wichtige Generalisierung des Standardproblems dar. Hier spielen die in der Realität auftretenden Öffnungszeiten bei Kunden eine entscheidende Rolle. Immer wichtiger wird auch das sogenannte *Just In Time* ([Mille, 1991]). Dabei verlagern Unternehmen Lagerkapazitäten auf die Straße oder Schiene, um so wertvolle Ressourcen vor Ort einsparen zu können. Bei diesem Vorgehen muß jedoch eine zügige und zeitlich akkurate Belieferung gesichert sein.
Die im vorangegangenen Kapitel definierten Begriffe lassen sich erweitern. Zuerst sollen folgende Abkürzungen festgelegt werden:

est (earliest start time) ist der Zeitpunkt, zu dem frühestens mit einer Aktion bei einem Kunden begonnen werden kann,

dda (due date) ist der letztmögliche Zeitpunkt, zu dem bei einem Kunden mit einer Aktion begonnen worden sein muß,

dur (duration) ist die Zeitdauer, die eine Aktion bei einem Kunden benötigt,

cst (current start time) ist der Zeitpunkt, zu dem tatsächlich eine Aktion bei einem Kunden begonnen wird.

Die beiden Schranken est_i und dda_i legen so bei einem Kunden i ein Zeitfenster fest, innerhalb dessen die Belieferung begonnen werden muß. Diese Werte sind vorgegeben. Der tatsächliche Zeitpunkt cst_i wird zur Laufzeit berechnet.

Definition 3.7 (Auftrag) *Unter einem Auftrag a_i bzgl. des VRPTW versteht man die Quantität q_i an Transportgütern, die Kunde i frühestens zum Zeitpunkt est_i fordert. Der Belieferungsvorgang muß spätestens zum Zeitpunkt dda_i begonnen haben und dauert dur_i Zeiteinheiten.*

$$a_i = (q_i, est_i, dur_i, dda_i)$$

Beispiel 3.8 Wird das Beispiel 3.3 um Zeitrestriktionen erweitert, erhält man:

CUST NO.	XCOORD.	YCOORD.	DEMAND	READY TIME	DUE DATE	SERVICE TIME
1	35.00	35.00	0.00	0.00	230.00	0.00
2	41.00	49.00	10.00	161.00	171.00	10.00
3	35.00	17.00	7.00	50.00	60.00	10.00
4	55.00	45.00	13.00	116.00	126.00	10.00
5	55.00	20.00	19.00	149.00	159.00	10.00
...						

READY TIME entspricht dabei *est*, DUE DATE *dda* und SERVICE TIME *dur*. Kunde 1 wird bei diesem Beispiel als Depot definiert, über DUE DATE bzgl. des Depots wird demnach die $DeadLine_i$ einer jeden Tour T_i bestimmt.

Definition 3.4 kann wörtlich übernommen werden, lediglich die Gültigkeit einer Tour muß erweitert werden:

Definition 3.9 (Tour) *Eine Tour $T_i = (i_0, \ldots, i_k)$ heißt **gültig** bzgl. des VRPTW, falls Bedingungen (3.1) bis (3.4) gelten und:*

$$cst_{i_j} + dur_{i_j} + t_{i_j i_{j+1}} \leq dda_{i_{j+1}} \quad j \in \{0, \ldots, k-1\}, i_j \vdash i_{j+1} \tag{3.9}$$

$$cst_k \leq DeadLine_i \tag{3.10}$$

Im allgemeinen wird bei dem VRPTW davon ausgegangen, daß die Beladevorgänge keine Zeit in Anspruch nehmen, d.h. das Fahrzeug steht zum Zeitpunkt 0 am Depot vollbeladen bereit.

Beispiel 3.10 Die folgende Tabelle stellt eine Auftragsmenge bzgl. des VRPTW dar:

Kunde	Bedarf	est	dda	dur
0	-	-	200	-
1	10	10	50	10
2	50	100	120	50
3	10	10	200	10

Abbildung 3.1 illustriert den zugrundeliegenden Routing-Graph und eine gültige Tour. Die Werte an den Kanten entsprechen den Distanz-/Zeiteinheiten, die zwischen je zwei Kunden liegen. Jedem Knoten wurde das entsprechende Zeitfenster zugewiesen.
Angenommen, das Fahrzeug fährt bei der Ausführung der Tour zum Zeitpunkt 0 weg, so ist die tatsächliche Startzeit bei Kunde 1 $cst_1 = 12$ und die Aktion bei diesem Kunden zum Zeitpunkt 22 beendet. Um Kunde 2 zu bedienen, muß eine Wartezeit in Kauf genommen werden. Diese ergibt sich aus $est_2 - (cst_1 + dur_1 + t_{12}) = 63$ und somit $cst_2 = est_2 = 100$. Die tatsächliche Startzeit bei Kunde 3 ergibt sich aus $cst_3 = \max(est_3, cst_2 + dur_2 + t_{23}) = 170$.

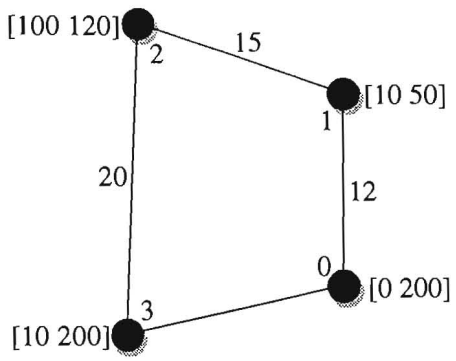

Abbildung 3.1.: Beispiel einer gültigen Tour mit Zeitrestriktionen

3.3. Das Pickup-and-Delivery-Problem PDP

Bisher wurde das Problem betrachtet, eine Menge von Kunden von einem zentralen Depot aus zu beliefern. Dieses VRP(TW) kann dahingehend erweitert werden, daß neben den Entladeaktionen (*Delivery*) auch die Beladung (*Pickup*) Teil der Planung wird. Die Tour muß weiterhin im Deopt beginnen und enden. Bei jedem Auftrag wird nun zwischen Start- und Endknoten unterschieden. Analog zu VRP/VRPTW kann man Versionen des Problems mit und ohne Zeitrestriktionen untersuchen. Wir definieren hier nur noch den Begriff des Auftrags bzgl. des PDPTW:

Definition 3.11 (Auftrag) *Unter einem Auftrag a_{ij} bzgl. des PDPTW versteht man die Quantität q_i an Transportgütern, die beim Kunden i frühestens zum Zeitpunkt est_i abgeholt werden darf, zum Zeitpunkt dda_i mit dem Vorgang begonnen worden sein muß und die Aktion dur_i Zeiteinheiten dauert. Die Güter dürfen frühesten zum Zeitpunkt est_j beim Kunden j abgeladen werden, es muß spätestens zum Zeitpunkt dda_j mit dem Entladen begonnen worden sein, die Dauer für den Vorgang beträgt dur_j Zeiteinheiten.*

$$a_{ij} = (q_i, est_i, dur_i, dda_i, est_j, dur_j, dda_j)$$

Wir werden *est_s*, *dda_s* und *dur_s* als Bezeichnung für die Zeitrestriktionen der Beladeaktion verwenden, *est_e*, *dda_e* und *dur_e* für die Entladeaktion.

3.4. Variationen

Das Standard Vehicle-Routing-Problem bildet die Grundlage für eine Reihe von Variationen, die in der Literatur diskutiert werden und unter dem Begriff *Fleet-Scheduling* zusammengefaßt werden. Einen weitreichenden Überblick dazu bietet [Golden and Assad, 1988]. Hier sollen lediglich einige Erweiterungen skizziert werden:

- Das VRP geht davon aus, daß die Kapazitäten aller Fahrzeuge gleich sind. Die Aufgabe dieser Beschränkung stellt jedoch nur eine unwesentliche Erweiterung dar, die Komplexität des Problems wird nicht gesteigert.
- Wird von einem inhomogenen Fuhrpark ausgegangen (Tankwagen, Pritschenwagen, Kipper, ...), muß bei jedem Fahrzeug ein Test auf Kompatibilität bzgl. des zu transportierenden Gutes durchgeführt werden.

- Besonders bei der Erweiterung zum PDP wird eine Variante interessant, bei der die Ordnung der Güter auf dem Fahrzeug eine Rolle spielt. Die Güter, die zuletzt aufgeladen wurden, müssen i.allg. zuerst wieder abgeladen werden.
- Das VRP mit Rückladungen (engl. *backhauls*) erlaubt es einzelnen Kunden dem Fahrzeug Güter mitzugeben, die wieder im Depot abgeladen werden müssen.
- Bei dem *Dead-End-VRP* muß das Fahrzeug nach Beendigung seiner Tour nicht wieder zum Depot zurückkehren.
- Erlaubt man das Aufteilen von einzelnen Auftragsgütern auf mehrere Fahrzeuge, führt dies im allgemeinen zu besserer Auslastung der Touren. Dieser Punkt wird noch ausführlich in Kapitel 7.2 besprochen.
- Die Existenz mehrerer Depots und/oder Speditionen erweitert das Anwendungsgebiet enorm. Verhandlungen zwischen einzelnen Speditionen können zur Gesamtverbesserung beitragen (s. Kapitel 7.3).
- Das *Dial-A-Ride*-Problem ähnelt dem PDPTW stark. Hier müssen im allgemeinen Personen transportiert werden. Oft gibt es dabei auch eine feste Deadline, zu der das Fahrzeug die Personen zu einem bestimmten Ort (nicht zwingend Depot) gebracht haben muß. Ein Beispiel wäre das Schulbussystem in den USA.

4. Lösungsansätze

4.1. Literatur

In diesem Kapitel werden einige aus der Literatur bekannte Methoden vorgestellt, die das jeweilige Problem lösen. Da das Standard-VRP sehr stark dem in Kapitel 2.3.2 charakterisierten Problem des Handlungsreisenden ähnelt, gibt es für diese Problemklasse eine Fülle von Ansätzen, die direkt von dem Traveling-Salesman-Problem übertragbar sind und auch für große Auftragsmengen nahezu optimale Lösungen liefern.
Bei der Erweiterung des Standardproblems um Zeitfenster zum VRPTW kommen jedoch neue Restriktionen hinzu, die eine Ausweitung der bisher bekannten Algorithmen und Heuristiken notwendig machen. Wir wenden uns weiterhin, sofern nicht anders vermerkt, dem VRPTW zu.
Prinzipiell unterscheidet man in der Literatur zwischen exakten und heuristischen Verfahren, die nun vorgestellt werden sollen.

4.1.1. Exakte Lösungsverfahren

Algorithmen, die die optimale Lösung für das VRPTW berechnen, beruhen allesamt auf einem der beiden Standardprinzipien impliziter Enumeration: *Dynamische Programmierung* und *Set Partitioning*. Dynamische Programmierung tritt in aller Regel als Subroutine in Branch-and-Bound-Ansätzen auf. Die detaillierte Beschreibung der Methoden in den folgenden beiden Paragraphen sind an [Desrochers et al., 1988] angelehnt.

Dynamische Programmierung

Bestandteile Dynamischer Programmierung sind Zustände, Übergänge zwischen Zuständen und Gleichungen, die den Wert einer Zielfunktion in jedem Zustand festlegen. Sei $G_R = (V \cup \{0\}, E)$ ein Routing-Graph mit Knotenmenge V und Kantenmenge E wie in Definition 3.1 beschrieben. Weiter sei die Entfernung d_{ij} für alle $(i, j) \in E$ durch eine Distanzmatrix D gegeben. Jeder Knoten repräsentiert einen Zustand, jede Kante einen Übergang zwischen zwei Zuständen und der Wert $l(j)$ bzgl. des Zustands j den kürzesten Pfad vom Depot 0 zum Knoten j. Die Gleichungen, um diese Werte zu berechnen sind gegeben durch:

$$\begin{aligned} l(0) &= 0, \\ l(j) &= \min_{(i,j)\in E} \{l(j) + d_{ij} | j \in V\} \end{aligned}$$

Die Laufzeit dieses Algorithmus ist von der Größe von G polynomiell abhängig. Der praktische Nutzen Dynamischer Programmierung ist somit auf relativ kleine Problemin-

stantiierungen beschränkt. Eine Möglichkeit, die Laufzeitprobleme wenigstens teilweise zu umgehen, wird nun beschrieben.

Branch-and-Bound

Die prinzipielle Idee des Branch-and-Bound-Ansatzes ist es, bei der Enumeration des Suchbaums (vgl. Kapitel 2.2.1) diesen nur teilweise aufzuspannen und in Teilprobleme zu zerlegen (branch).
Zunächst wird im Suchbaum ein gültiger Tourplan bestimmt. Diese Lösung stellt eine obere Schranke für die Qualität der optimalen Lösung dar. Bei der Generierung der Teillösungen erhält man für jedes Teilproblem eine untere Schranke, nämlich die Qualität der Lösung des Problems, soweit es verzweigt wurde. Ist diese untere Schranke größer als die zuvor berechnete obere Schranke, muß dieser Zweig nicht weiter untersucht werden, da die Qualität der Endlösung nicht mehr unter der bisher besten liegen kann. Ganze Segmente des Suchbaumes können somit abgeschnitten werden (bound).
Genau diesen Ansatz verfolgen beispielsweise [Christofides et al., 1981], wobei der Algorithmus mit Dynamischer Programmierung startet. Die Teilprobleme werden jeweils so zerlegt, daß die Rekursion über den neuen Suchraum lediglich polynomielle Zeit benötigt und zu einer Schranke für den minimalen Wert der ursprünglichen Lösung führt. Weitere Ansätze lassen sich in [Kolen et al., 1987] oder [Desrochers et al., 1992] finden.

Set-Partitioning

Das VRP bzw. VRPTW kann als Mengen-Partitionierungsproblem umformuliert werden, wobei Vorgehensweisen der Matrizenrechnung eingesetzt werden. Variablen (Spalten) entsprechen gültigen Touren:
Sei Θ die Menge aller gültigen Touren bzgl. des gestellten Problems. Für jede Tour $T_i \in \Theta$ wird der Wert γ_i als die Summe der Kosten ihrer Kanten definiert und $\delta_{ij}, j \in I\!N$ als binäre Konstante, die gleich 1 ist, falls Kunde j von Tour T_i bedient wird und 0 sonst. x_i sei 1 falls Tour T_i benutzt wird und 0 sonst. Das Set-Partitioning-Problem ist, die Summe

$$\sum_{T_i \in \Theta} \gamma_i x_i$$

zu minimieren, wobei

$$\sum_{T_i \in \Theta} \delta_{ij} x_i = 1, \quad j \in I\!N$$

$$x_i \in \{0,1\} \text{ für } T_i \in \Theta$$

[Desrosiers et al., 1986] erweitern diesen Algorithmus sogar auf den Fall eines inhomogenen Fuhrparks. [Dumas, 1985] entwickelte einen Ansatz für das PDPTW, wobei er Probleme mit 30 Kunden in 100 Sekunden auf einer CDC Cyper 173 löste. [Desrochers et al., 1992] stellen ein Verfahren vor, mit dem es gelang Beispiele des VRPTW mit 100 Kunden optimal zu lösen. Diese Testdaten werden in Kapitel 6.1 detailliert untersucht, da sie als Vergleichsdaten für die Qualität unseres Systems dienen.

4.1.2. Heuristische Verfahren

Exakte Lösungsverfahren liefern zwar optimale Ergebnisse, beanspruchen jedoch im allgemeinen unverhältnismäßig viel Zeit, wenn der Suchraum expandiert. Optimale Lösungen

für das VRPTW sind heute sogar nur bis zu einer Größenordnung von etwa 100 Kunden bekannt. Dies verlangt nach heuristischen Verfahren, die zwar nicht den Anspruch erheben, optimale Lösungen zu liefern, diese jedoch in vorhersehbarer Zeit liefert, die oftmals nur Bruchteile der Zeit zur Generierung exakter Ergebnisse beträgt. In den folgenden Kapiteln soll ein kurzer Überblick über die gebräuchlichsten Ansätze gegeben werden. Prinzipiell unterscheidet man zwischen *Konstruktionsverfahren*, *Nachoptimierungsverfahren* und *Unvollständiger Optimierung*.

Konstruktionsverfahren

Konstruktionsverfahren sind in aller Regel zweigeteilte Verfahren, die zwischen einem Auswahl- und Einfügekriterium unterscheiden. Zum einen spielt eine Rolle, welcher Kunde als nächstes in eine bestehende Tour eingeplant werden soll, zum anderen an welcher Stelle dies geschehen soll.
[Solomon, 1983] war einer der Ersten, der vorhandene Lösungsverfahren des VRP auf das VRPTW übertrug. Zur Vertiefung der nun dargelegten Vorgehensweisen seien daneben auch [Solomon, 1987] und [Desrochers et al., 1988] empfohlen.

Savings: Wahrscheinlich der erste und bekannteste Ansatz zum heuristischen Lösen des VRP ist das Savings-Verfahren von [Clarke and Wright, 1964]. Zu Beginn des Verfahrens wird jedem Kunden eine gesonderte Tour zugewiesen. Iterativ werden nun jeweils zwei bestehende Touren T und T' über je zwei Kunden, die sich am Ende ihrer jeweiligen Tour befinden, miteinander verbunden. Die resultierende Tour T'' darf natürlich bestehende Ladekapazitäten des letztendlich ausführenden Fahrzeugs nicht überschreiten. Ein Maß für die eingesparten Kosten (savings) ist dabei

$$sav_{ij} = d_{i0} + d_{0j} - \mu d_{ij}, \quad \mu \geq 0.$$

Mit $\mu = 1$ erhält man beispielsweise den Gewinn, der sich ergibt, wenn man Kunden i und j auf der gleichen Tour bedient, statt sie jeweils getrennt vom Depot aus anzufahren.

Beispiel 4.1 Abbildung 4.1 zeigt einen Zusammenschluß zweier Touren mit einem Savings-Wert von 10 Längeneinheiten für $\mu = 1$.

Beim Übergang zum VRPTW sind nun noch Zeitrestriktionen zu beachten, d.h. sowohl die Orientierung, in der die beiden Touren miteinander verbunden werden, muß berücksichtigt werden, als auch die Zeitfenster bei jedem einzelnen Kunden, speziell bei der erweiternden Tour. Werden Kunden i und j zweier verschiedener Touren T und T' miteinander verbunden, spricht man von konformer Orientierung, falls $T = (\ldots, i, 0)$ und $T' = (0, j, \ldots)$ bzw. umgekehrt.
Das hier vorgestellte Savings-Verfahren bevorzugt Verbindungen zwischen zwei Kunden, die räumlich nah beieinander liegen. Dabei spielt die 'zeitliche Entfernung' im Sinne von Differenzen zwischen den jeweiligen Zeitrestriktionen, eine untergeordnete Rolle. Die Folge sind in aller Regel lange Wartezeiten. Um sowohl räumlicher als auch zeitlicher Nähe gerecht zu werden, schlägt [Solomon, 1987] ein modifiziertes Savings-Verfahren vor, in dem Wartezeiten limitiert werden. Soll z.B. T und T' über die Kante (i, j) verbunden werden, geschieht dies nur, falls die Summe der Wartezeiten in der resultierenden Tour T'' einen Schwellwert W nicht überschreitet. Solomon nennt diesen Ansatz *Savings with Waiting Time Limits*.

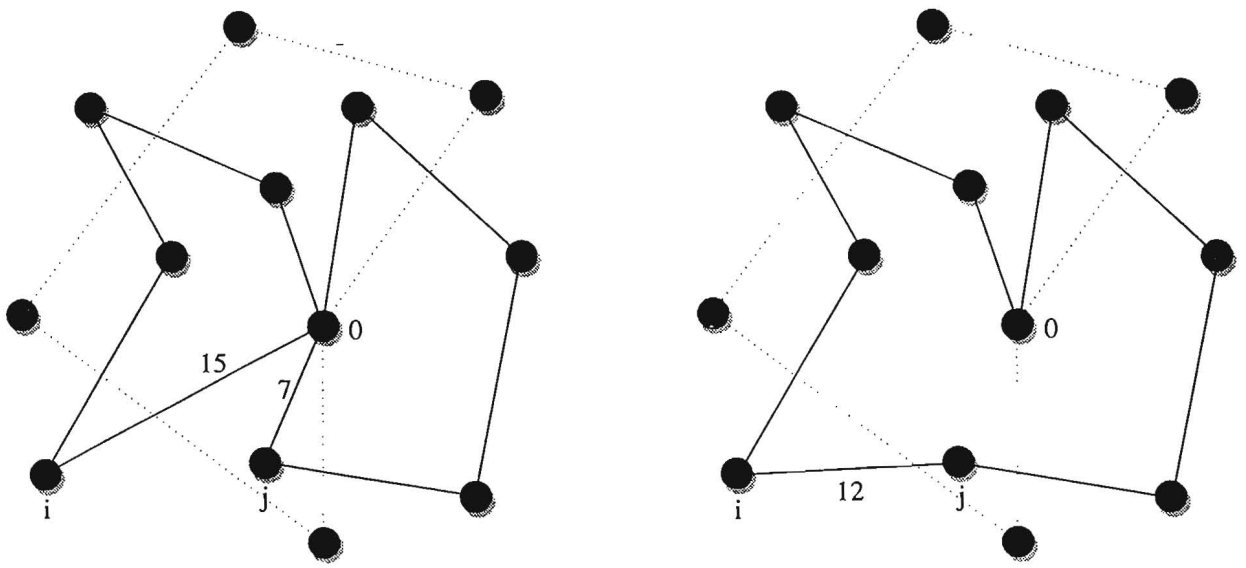

Abbildung 4.1.: Zusammenschluß zweier Touren beim Savings-Algorithmus

Nearest Neighbor: Das Nearest-Neighbor-Verfahren gehört zu der Klasse der sequentiellen Konstruktionsverfahren und wurde in folgender Form erstmals in [Solomon, 1983] vorgestellt. Hier steht das Auswahlkriterium im Vordergrund, das Einfügekriterium wird praktisch ganz vernachlässigt, da neue Kunden immer am Ende einer bestehenden Tour eingeplant werden.
Anfangs besteht eine Tour lediglich aus dem Depot. Nun wird iterativ zu dem jeweils letzten Kunden der aktuellen Tour der Kunde aus der verbleibenden Auftragsliste ausgesucht, der sich zu diesem am 'nächsten' befindet (nearest neighbor). Der Begriff der Nähe wird weiter unten konkretisiert. Auch hier sind wiederum Zeit- und Laderestriktionen bzgl. der Fahrzeuge zu beachten. Kann kein Kunde mehr an die aktuelle Tour angehängt werden, wird eine neue initialisiert und das Verfahren arbeitet auf dieser weiter.

Beispiel 4.2 In Abbildung 4.2 wird versucht, am Ende der bestehenden Tour nach Kunde i den Kunden j einzuplanen.

Der Begriff der Nähe sollte auch hier wiederum räumliche als auch zeitliche Komponenten vereinen. Sei dazu

$$t'_{ij} = cst_j - (cst_i + dur_i)$$

die Zeitdifferenz zwischen dem Beenden eines Auftrags beim Kunden i und dem Beginn des neuen Auftrages beim Kunden j und

$$t''_{ij} = dda_j - (cst_i + dur_i + t_{ij})$$

die dem Fahrzeug verbleibende Zeit, bis die Belieferung bei Kunde j begonnen haben muß . Dann kann die Nähe zweier Kunden definiert werden durch

$$c_{ij} = \delta_1 d_{ij} + \delta_2 t'_{ij} + \delta_3 t''_{ij} \text{ mit } \delta_1 + \delta_2 + \delta_3 = 1 \text{ und } \delta_1, \delta_2, \delta_3 \geq 0.$$

Insertion: Grundlegend differenziert man zwischen zwei Variationen des Insertion-Algorithmus: sequentielles und paralleles Einfügen. Beide Varianten finden Anwendung in dem in Kapitel 5 beschriebenen Speditionsszenario.

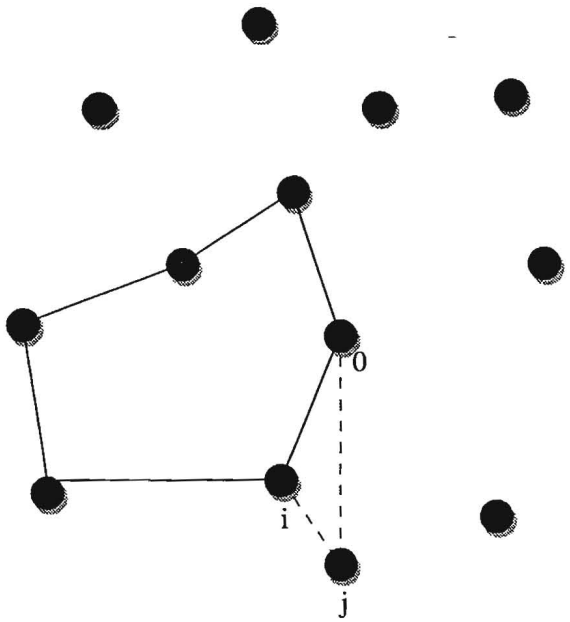

Abbildung 4.2.: Nearest Neighbor

Sequentielles Einfügen: Die Insertion-Methode unterscheidet zwischen drei Kriterien, die für die Qualität der Lösung eine entscheidende Rolle spielen:

- Initialisierungskriterium, I
- Einfügekriterium E und
- Auswahlkriterium A.

Zu Beginn des Verfahrens wird eine neue Tour initialisiert. Dies geschieht mittels I, das einen Kunden liefert, der zuerst in die leere Tour eingeplant wird. Mögliche Kriterien sind z.B. die Reihenfolge der Belieferungszeitpunkte (*est*, *dda*) oder die Entfernung zum Depot. Bei der zeitorientierten Initialisierung hat sich vor allem die Sortierung nach dem letztmöglichen Zeitpunkt, zu dem die Belieferung begonnen haben muß (*dda*), als fruchtbar erwiesen. Bei den distanzorientierten Kriterien ist das Sortieren nach absteigender Entfernung zum Depot hervorzuheben (farthest Customer), d.h. der vom Depot am weitesten entfernte Kunde wird zuerst eingeplant.
Sei $T_i = (i_0, \ldots, i_k)$ die aktuell gültige Tour. Zuerst wird für jeden sich noch in der Auftragsliste befindlichen Kunden u der beste gültige Einfügeplatz zwischen zwei Kunden i_p und i_q, $i_p \overset{i}{\vdash} i_q, p, q \in \{1, \ldots, k\}$ berechnet. Dies geschieht mittels des Einfügekriteriums

$$E(i(u), u, j(u)) = \min[E(i_p, u, i_q)], \quad p, q \in \{1, \ldots, k\}.$$

Der Kunde u', der am 'besten' die aktuelle Tour erweitert, wird über das Auswahlkriterium bestimmt:

$$A(i(u'), u', j(u')) = \mathit{optimum}[A(i(u), u, j(u))],$$

für noch nicht eingeplante Kunden u, die zu einer weiteren gültigen Tour führen. Demnach wird Kunde u' zwischen $i(u')$ und $j(u')$ eingefügt. Können keine weiteren Kunden

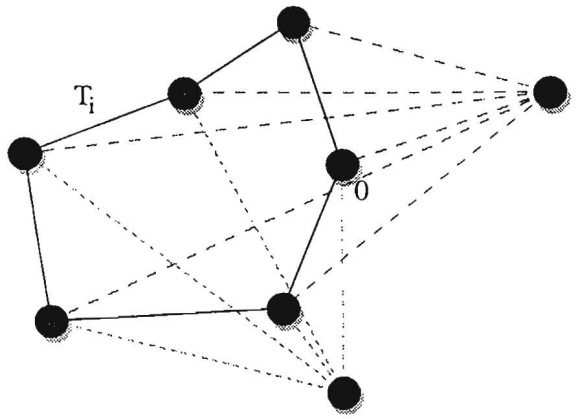

Abbildung 4.3.: Sequentielles Insertion-Verfahren

gefunden werden, die die aktuelle Tour zu einer gültigen erweitern, wird eine neue mittels I initialisiert. Das Verfahren ist beendet, wenn die Auftragsliste leer ist.
[Solomon, 1987] schlägt drei mögliche Varianten vor, das Einfüge- und Auswahlkriterium zu bestimmen. An dieser Stelle soll lediglich die erste Möglichkeit vorgestellt werden, da diese sich qualitativ von den beiden anderen abhebt. Dem interessierten Leser sei die lesenswerte Quelle empfohlen.
Sei also

$$\begin{aligned} E_1(i,u,j) &= d_{iu} + d_{uj} - \mu d_{ij}, \quad \mu \geq 0; \\ E_2(i,u,j) &= cst_{j(u)} - cst_j, \end{aligned}$$

wobei $cst_{j(u)}$ der neue Zeitpunkt ist, zu dem die Aktion beim Kunden j beginnen kann, falls u eingeplant wurde. E_1 verfolgt dabei räumliche und E_2 zeitliche Aspekte. Insgesamt folgt:

$$\begin{aligned} E(i,u,j) &= \alpha_1 E_1(i,u,j) + \alpha_2 E_2(i,u,j), \quad \alpha_1, \alpha_2 \geq 0, \alpha_1 + \alpha_2 = 1; \\ A(i,u,j) &= \lambda d_{0u} - E(i,u,j), \quad \lambda \geq 0. \end{aligned}$$

Diese Insertion-Heuristik versucht den Nutzen zu maximieren, den man erhält, wenn man einen Kunden auf einer bestehenden Tour einplant, statt ihn direkt zu bedienen. Wählt man beispielsweise $\mu = \alpha_1 = \lambda = 1$ und $\alpha_2 = 0$, ist $A(i_p, u, i_q)$ der Gewinn in Entfernungseinheiten, falls man Kunden u in der gleichen Tour T_i einplant, in der sich i_p und i_q ($i_p \overset{i}{\vdash} i_q$) befinden, statt u direkt vom Depot aus anzufahren.
Natürlich liefern verschiedene Werte für μ und λ verschiedene mögliche Kriterien für den besten neu einzuplanenden Kunden und dessen besten Einfügepunkt. Wählt man beispielsweise $\alpha_2 = 0$ und $c_{0u} = d_{0u}$ erhält man den Ansatz wie in [Mole and Jameson, 1976] für VRP-Probleme beschrieben.
Insgesamt handelt es sich um eine Generalisierung des mit Zeitrestriktionen behafteten *Nearest-Neighbor*-Ansatzes, wie auf Seite 25 beschrieben. Hier wird jedoch jeder gültige Einfügepunkt in Betracht gezogen, statt nur das Ende der aktuellen Tour.

Paralleles Einfügen: Die sequentielle Einfügevariante läßt sich größtenteils auf die parallele übertragen. Die Sichtweise erfolgt hier nun nicht mehr aus der Perspektive des einzelnen Fahrzeugs (das Fahrzeug sucht sich aus der Auftragsliste den Kunden aus, der am besten in die bestehende Tour paßt), sondern aus der darüberliegenden.
Zu Beginn des Verfahrens wird die Auftragsmenge nach einem beliebigen Kriterium sortiert. Ähnlich dem Initialisierungskriterium beim sequentiellen Ansatz, kann beispielsweise eine Ordnung bzgl. der Entfernung zum Depot oder des letztmöglichen Belieferungszeitpunktes (*dda*) gewählt werden. Der Tourplan besteht anfangs aus nur einer leeren Tour. In diese wird nun der erste sich in der Auftragsliste befindliche Kunde eingeplant. Dies geschieht analog zum Sequential-Insertion bzgl. des Auswahlkriteriums. Dieses wird jedoch nicht auf alle Kunden, wohl aber auf alle Fahrzeuge angewandt. Kann der aktuell einzuplanende Kunde in keine der vorhandenen Touren eingefügt werden, wird eine neue angelegt, solange bis alle Kunden eingeplant sind.

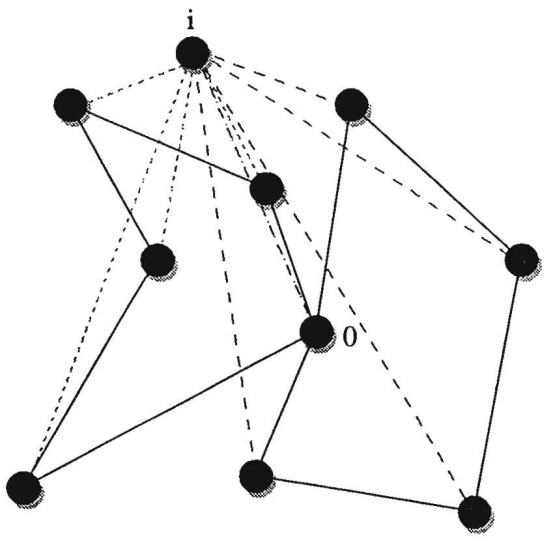

Abbildung 4.4.: Paralleles Insertion-Verfahren

Beispiel 4.3 Abbildung 4.3 zeigt den Versuch, beim sequentiellen Insertion-Verfahren die verbleibenden Kunden in die aktuell geplante Tour T_i einzufügen. Im Gegensatz dazu zeigt Abbildung 4.4 das Vorgehen beim parallelen Ansatz, wo der aktuell einzuplanende Kunde i versuchsweise in allen bestehenden Touren hinzugefügt wird.

Sweep: Die Sweep-Heuristik unterscheidet zwischen zwei Stufen:

- Cluster-Bildung
- Scheduling

In der ersten Phase wird jedem Kunden eine Tour zugewiesen, wie in der ursprünglichen Sweep-Heuristik von [Gillet and Miller, 1974] vorgestellt. Dazu wird der Routing-Graph $G_R = (V \cup \{0\}, E)$ als Euklid'sche Ebene interpretiert, d.h. jedem Knoten des Graphen werden X- und Y-Werte im karthesischen Koordinatensystem so zugeordnet, daß das Depot im Ursprung liegt. Jedem Kunden $i \in V$ kann dadurch ein Polarwinkel zugeordnet

werden. Bei einem beliebigen Kunden i_0 beginnend, werden alle Kunden in Reihenfolge der Größe ihrer Polarwinkel in einer Tour geclustert, solange eventuelle Ladekapazitäten nicht überschritten werden. In der darauffolgenden Scheduling-Stufe werden die so gewonnenen Sektoren zu gültigen Touren zusammengefaßt und u.U. optimiert. Der Prozeß läßt sich so für n verschiedene Initialknoten i_0 wiederholen. Analog dazu kann ein *backward-sweep* durchgeführt werden, d.h. die Kunden werden mit absteigendem Polarwinkel zu Clustern zusammengefaßt. Insgesamt erhält man also $2n$ Variationen, aus denen die beste ausgewählt wird.

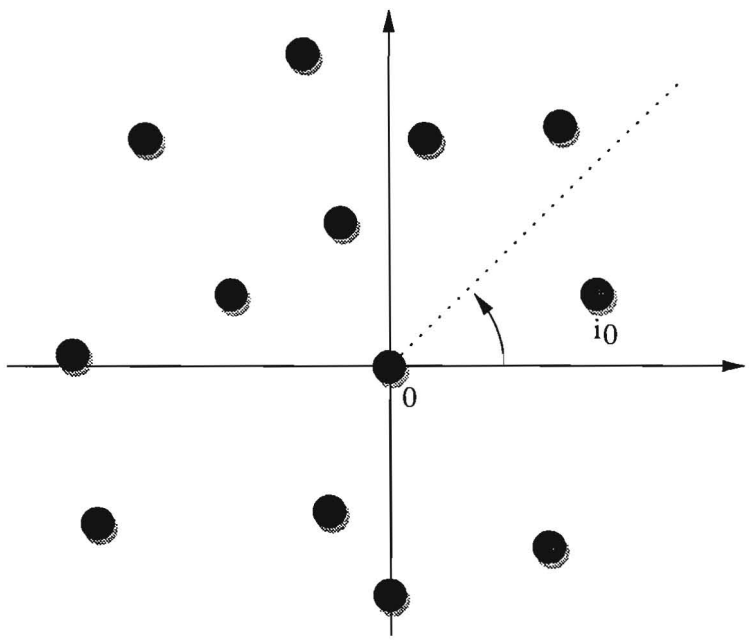

Abbildung 4.5.: Sweep-Heuristik

[Solomon, 1987] erweitert die Sweep-Heuristik auf das VRPTW. Analog zu dem oben beschriebenen Verfahren wird zunächst, ausgehend von den rein räumlichen Daten, für jeden Sektor eine Tour erstellt. Dies geschieht mittels eines Konstruktionsverfahrens wie z.B. dem Insertion. Aufgrund der Zeitfenster bleiben u.U. einige der Kunden ungeplant, da sie eine ungültige Erweiterung darstellen würden. Mit diesen wird der Cluster-Scheduling-Prozeß solange wiederholt, bis alle Kunden eingeplant sind.
Da die Sweep-Heuristik fast ausschließlich räumliche Aspekte in Betracht zieht, ist diese vor allem für das reine VRP geeignet.

Direkter Vergleich der Heuristiken

[Solomon, 1987] hat die vorgestellten Verfahren direkt miteinander verglichen. Dazu hat er eine von [Christofides et al., 1979] vorgestellte Standardmenge von Routing-Problemen um Zeitrestriktionen erweitert. Die Charakterisierung der Mengen sowie die veröffentlichten Ergebnisse werden in Kapitel 6.1 detailliert behandelt, da sie auch bei dem in dieser Arbeit vorgestellten System als Grundlage zur Bestimmung der Qualität dienten.

Nachoptimierungsverfahren

Heuristische Ansätze liefern in aller Regel Ergebnisse, deren Qualität durch darauf aufsetzende Verbesserungsverfahren nochmals gesteigert werden kann. Grundsätzlich wird versucht, jede einzelne Tour lokal zu optimieren, um dadurch den Gesamt-Tourplan zu verbessern.

k-opt: Eines der bekanntesten Nachoptimierungsverfahren bzgl. Fleet-Scheduling-Problemen ist wohl das von [Lin, 1965] vorgestellte *k-opt* oder *k-exchange* zur Verbesserung von Lösungen des Problems des Handlungsreisenden. [Lin and Kerninghan, 1973] generalisierten den Ansatz, der von vielen anderen Autoren auf abgewandelte Probleme angewandt wurde.
Prinzipielle Idee des k-opt ist es, in einer bestehenden Tour k Kanten durch (nicht zwingend andere) k Kanten zu ersetzen. Bzgl. des Problems des Handlungsreisenden ist dies bei festem k in konstanter Zeit möglich, wie in [Desrochers et al., 1988] beschrieben. Bei Hinzunahme von Zeitfenstern wird diese linear ($c \cdot n, c = const$), da eine Änderung an einem Punkt eine Modifikation der gesamten Tour notwendig machen kann. Weil mit steigendem k die Anzahl der möglichen Austauschvorgänge enorm zunimmt (n^k), werden in aller Regel nur 2-opt und 3-opt Verfahren angewandt.

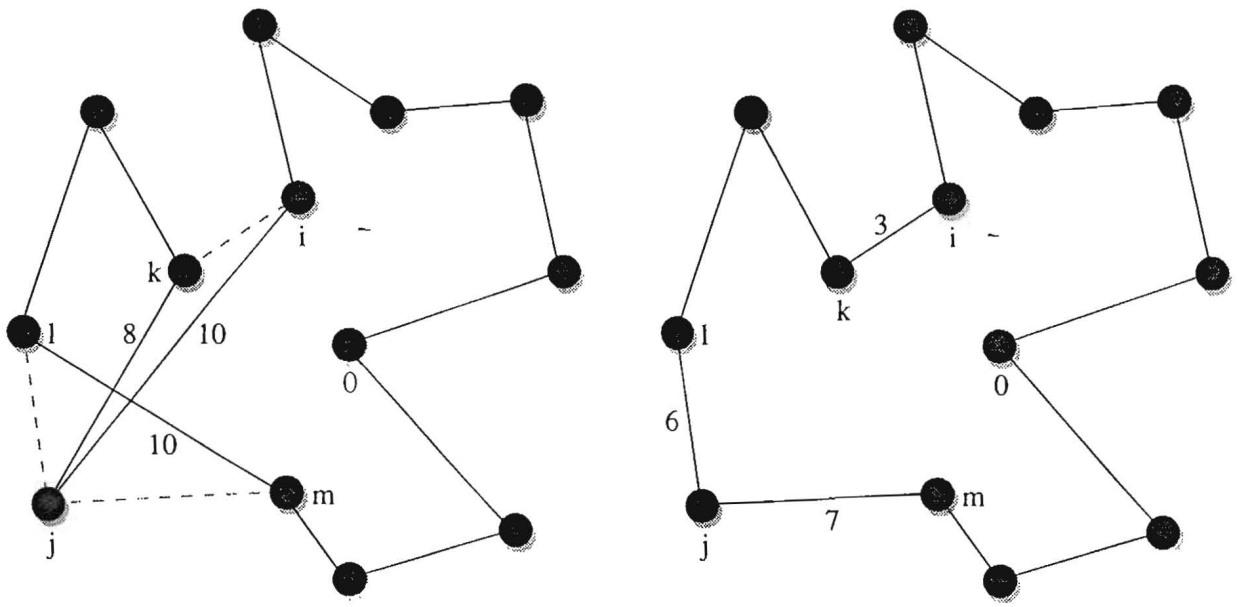

Abbildung 4.6.: Verbesserung der Tour durch einen 3-opt-Schritt

Beispiel 4.4 Abbildung 4.6 zeigt, wie die lokale Tour durch einen 3-opt-Schritt verbessert werden konnte. Kanten w_{ij}, w_{jk} und w_{lm} wurden durch w_{ik}, w_{lj} und w_{jm} ersetzt. Die Verbesserung beträgt

$$(\omega(w_{ij}) + \omega(w_{jk}) + \omega(w_{lm})) - (\omega(w_{ik}) + \omega(w_{lj}) + \omega(w_{jm})) = 12$$

[Savelsberg, 1990] beschreibt eine Implementierung des k-opt-Verfahrens, angewandt auf das VRPTW mit Rückladungen.

Simulated-Trading: Die Implementierung der Simulated-Trading Heuristik war ein Hauptbestandteil der vorliegenden Arbeit. Das Verfahren ist detailliert im Kapitel 5.4 beschrieben.

Unvollständige Optimierung

Unvollständige Optimierung beruht auf den in Kapitel 4.1.1 beschriebenen exakten Lösungsverfahren. Sie dient dazu, die im allgemeinen sehr hohen Laufzeiten der optimalen Scheduler zu reduzieren.
Ein Beispiel unvollständiger Optimierung ist der Branch-and-Bound-Ansatz, bei dem der Suchbaum nur partiell evaluiert wird. Dazu kann eine Zeitschranke gesetzt werden, bis zu der man das Verfahren durchführt. Die bis dahin beste Lösung wird übernommen. Eine andere Möglichkeit besteht darin, aufgrund von Heuristiken weitere Zweige des Suchbaums abzuschneiden.

4.2. Vorläufer des Oz-Speditionsszenarios

Ein erster Ansatz am DFKI das PDPTW mit Mitteln der VKI zu lösen war das MAGSY-Speditionsszenario MARS (Modeling a Multi-Agent Scenario for Shipping Companies) [Fischer and Kuhn, 1993]. MAGSY[1] ist eine Entwicklungsplattform für Multiagentensyteme unter UNIX, basierend auf einem OPS5-System [Fischer, 1993]. Es bietet echte Nebenläufigkeit, da jedem Agent ein eigener Unix-Prozeß mit einer eindeutigen Kommunikationsadresse (Unix-Port) zugeordnet wird. Mit der Entwicklung von *Oz* am DFKI (vgl. Kapitel 5.6) wurde ein einfaches PDP ohne Zeitrestriktionen zu Demonstrationszwecken von Christian Schulte und Andreas Schroth implementiert. Vorbild für das System war die MARS-Implementierung. Dieses wurde zunächst in einem Fortgeschrittenenpraktikum zu einem PDPTW erweitert [Schier, 1994]. Die dort gemachten Erfahrungen führten letztendlich zu der vorliegenden Arbeit. Diese Neuimplementierung soll im Gegensatz zum MARS-System *MAS-MARS* genannt werden.

[1] verfügbar per ftp: `ftp.dfki.uni-sb.de` im Verzeichnis `pub/VKI/MAGSY`

5. Das Speditionsszenario

In diesem Kapitel sollen die Ideen und Strukturen beschrieben werden, die der Implemantation dieser Arbeit zugrunde liegen. Dabei steht vor allem der Weg der Lösungsfindung und die in der Arbeit verwirklichten Programmierparadigmen im Vordergrund. Dazu sei auch [Fischer and Kuhn, 1993] empfohlen, deren Ausführung die Grundlage für den vorliegenden Bericht ist.
Wir wollen uns der Multiagentensysteme zur Lösung des VRP bedienen. Dabei sei gleich anfangs gesagt, daß das nun vorzustellende Szenario nicht nur das VRP, sondern auch das PDP und dessen Abwandlungen mit Zeitrestriktionen zu lösen in der Lage ist. Obwohl VRPTW und PDPTW sehr ähnlich erscheinen, verlangen sie doch verschiedene Auswertungsstrategien und Kostenfunktionen, um möglichst optimale Ergebnisse zu liefern. Das hier vorgestellte Speditionsszenario wurde bzgl. des VRPTW optimiert und liefert in dieser Variante die besten Ergebnisse.
Ein weiterer wichtiger Aspekt der Arbeit wird in Kapitel 5.3 näher erörtert, nämlich die Fähigkeit ein dynamisches Problem zu lösen und das Vermögen des Systems, auf eine sich ändernde Umwelt zu reagieren.

5.1. Agentenstruktur

Wir wollen uns zunächst dem Aufbau der Agentenstruktur zuwenden. Wie bereits eingangs im Kapitel 2.1.1 dargelegt, verlangen Multiagentensysteme andere Strukturen als große, homogene KI-Applikationen. Alle Agenten haben ihr eigenes Wissen und Können, sollen jedoch zusammen dazu beitragen, das Gesamtproblem zu lösen. Dies erfordert eine Koordination der im hohen Maße nebenläufigen Aktionen. Zur Lösung dieses Problems wurde das in Kapitel 2.1.2 vorgestellte *Contract-Net-Protocol* verwendet, dessen Instantiierung nun vorgestellt werden soll.

5.1.1. Das Contract-Net-Protocol

Die hierarchische Struktur des Modells eignet sich hervorragend, diese auf das Speditionsszenario zu übertragen. Dabei findet eine vertikale Kooperation auf zwei Ebenen statt, zum einen zwischen dem *Broker* und den Speditionen, zum anderen zwischen den Speditionen und ihren Fahrzeugen. Den prinzipiellen Aufbau verdeutlicht Abbildung 5.1. Ein Charakteristikum von Multiagentensystemen ist es, Wissen einzelner Einheiten so modular als möglich zu halten. Genau dieses Vorgehen wird mit Hilfe des Protokolls untersützt: einzelne Agenten halten lediglich einen Nachrichtenkanal zwischen dem im Protokoll direkt über- bzw. untergeordneten Agenten aufrecht, ohne nähere Informationen über dessen Auslastung, Position, ... zu haben. Letztendlich muß der Master seinem Slave 'glauben', die ihm zugeordnete Teilaufgabe korrekt bearbeitet, und vollständige Angaben

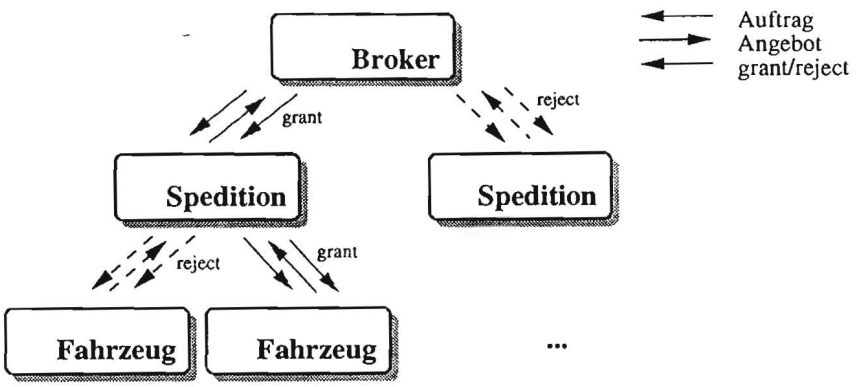

Abbildung 5.1.: Contract-Net-Protocol

bzgl. der dafür aufzubringenden Kosten gemacht zu haben. Läßt man diese Prämisse fallen, kann ein 'bewußt' konkurrierendes Verhalten zwischen Agenten modelliert werden. Dieser Ansatz verlangt jedoch eine Erweiterug des Standardverfahrens und wird in Kapitel 7.3 diskutiert.

5.1.2. Broker

Der Broker stellt die oberste Instanz des Protokolls dar. Er verwaltet die Auftragsmenge und steht in direktem Kontakt einerseits zu den Kunden, andererseits zu den Speditionen. Der Broker bietet den Speditionen einzelne oder Mengen von Aufträgen an und holt dafür Angebote ein, d.h. er verhandelt direkt mit den einzelnen Fuhrunternehmen, und wählt aus der Menge der Angebote das ihm am besten[1] erscheinende aus. Dieses Angebot erhält den Zuschlag zur Beförderung des Transportgutes (*grant*), die restlichen Speditionen erhalten eine Absage (*reject*).

5.1.3. Spedition

Die Spedition ihrerseits übt sowohl Slave- (bzgl. Broker) als auch Masterfunktionen (bzgl. ihrer Fahrzeugflotte) aus. Die vom Broker als *contractor* entgegengenommenen Aufträge werden nun als *master* an jedes einzelne Fahrzeug weitergeleitet. Der Mechanismus ist ähnlich zu dem der darüberliegenden Ebene. Das Fuhrunternehmen holt Informationen von jedem Fahrzeug ein, mit welchem Mehraufwand der angebotene Kunde bedient werden kann. Diese Informationen werden verglichen, bzgl. des 'besten' Fahrzeuges wiederum ein Angebot für den Broker berechnet und an diesen weitergegeben. Erhält die Spedition die Zusage vom Broker, wird diese an das Fahrzeug weitergegeben, das den neuen Auftrag nun fest in die bestehende Tour einplant. Die Spedition selbst braucht keine Informationen, wo sich die einzelnen Fahrzeuge gerade befinden, lediglich eine Verbindung muß aufrecht erhalten werden.

[1] Die Qualität eines Angebotes kann eine Reihe von Faktoren beinhalten, so beispielsweise den Preis, die Zuverlässigkeit des Fuhrunternehmens, regionale Aspekte, ... In der vorliegenden Implementierung orientiert sich die Bewertung ausschließlich am Preis.

5.1.4. Fahrzeug (Vehicle)

Das Fahrzeug steht an der untersten Stufe des Protokolls und ist somit ausschließlich *contractor*. Die eigentiche Plangenerierung, die im folgenden Kapitel erläutert wird, findet hier statt. Dazu erhält das Fahrzeug von seiner Spedition den Auftrag, einen neuen Kunden in seine bestehende Tour einzufügen. Die anfallenden Kosten werden berechnet und an die Spedition zurückgegeben.

5.2. Planen

Die Qualität der berechneten Gesamtlösung hängt in hohem Maße von der Plangenerierung ab. Dabei ist natürlich die gewählte Heuristik entscheidend. Aus den in Kapitel 4.1.2 vorgestellten Verfahren wurde das sequentielle (Seite 26) und parallele Einfügen (Seite 28) implementiert. Wie sich noch in Kapitel 6.1 zeigen wird, hat sich die sequentielle Variante im Vergleich zu den restlichen Heuristiken als qualitativ am besten erwiesen.
Wie sich aus Anfragen bei Fuhrunternehmen ergab, ist das ausschlaggebende Kriterium für ein 'gutes' Kostenmaß zum einen die Anzahl der benötigten Fahrzeuge (Fixkosten), zum anderen die zurückgelegte Entfernung (variable Kosten) pro Einheit. Rein zeitliche Aspekte spielen dabei nur eine sekundäre Rolle, solange diese die maximal erlaubte Einsatzzeit des Fahrers nicht überschreiten. Ein 'vernünftiger' Planungsvorgang sollte also der Distanzminimierung gegenüber der zeitlichen den Vorzug geben.
Die Problematik, die sich beim Einplanen neuer Kunden stellt, ist die Möglichkeit zur Reaktion auf eine sich ständig ändernde Umwelt. Die Fahrzeuge können sich bereits auf einer Tour befinden, wenn sie den Auftrag erhalten, Zusatzkosten für einen neu einzufügenden Kunden zu berechnen. Diese Berechnung nimmt natürlich auch Zeit in Anspruch, um die sich unter Umständen die Ausführung der Gesamttour verschieben kann. Hier spielen Aspekte des *Echtzeit-Scheduling* eine Rolle. Die Tour selbst ist dabei im Sinne von Kapitel 2.2 als Ressource aufzufassen. Beim Einplanen ist es notwendig, die aktuelle Tour soweit einzugrenzen, so daß die Neuplanung sich zeitlich nicht mit den noch zu bedienenden Kunden überschneidet.

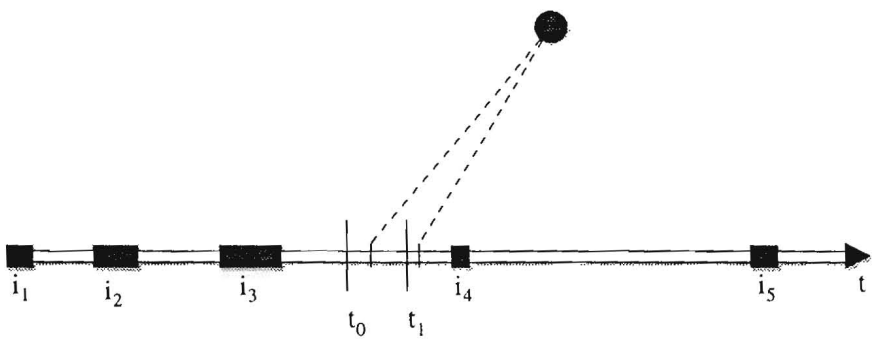

Abbildung 5.2.: Echtzeit-Einplanung neuer Kunden

Beispiel 5.1 Abbildung 5.2 zeigt eine unzulässige Tourerweiterung. Zum Zeitpunkt t_0 beginnt der Planungsvorgang und dauert $\Delta t = t_1 - t_0$. Da aber bei Abschluß der Planung eine rechtzeitige Belieferung nicht mehr möglich ist (da die Zeit zu weit fortgeschritten ist), muß der Gesamtplan verworfen werden.

Die dem System zugrundeliegende Programmiersprache *Oz* unterstützt nebenläufige Ausführung (s. Kapitel 5.6). Da die Antwortzeit auf eine Anfrage bzgl. Einfügens eines Kunden im Bereich von Bruchteilen von Sekunden liegt, reicht es aus, den aktuellen Tourplan nur ab dem Beginn der nächsten Aktion bei einem Kunden einzugrenzen, wie die Abbildung 5.3 illustriert.

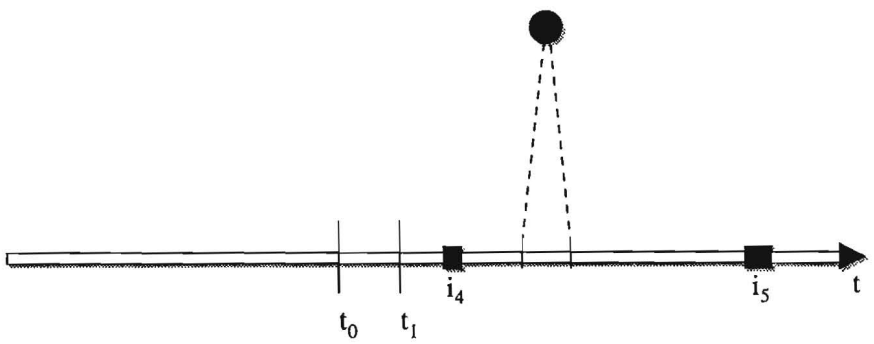

Abbildung 5.3.: Einschränkung des Tourplans bei Echtzeit-Planung

5.3. Offenes und geschlossenes Problemlösen

In aller Regel erhalten Speditionen einzelne Aufträge vom Broker, für die es schnellstmöglichst ein Angebot zu machen gilt. Die Spedition hat keine Kenntnis darüber, wieviel Aufträge noch eingehen werden und ist demnach immer nur auf den aktuellen Auftrag fixiert, unter Berücksichtigung der vorangegangenen. Der aktuelle Auftrag wird nach dem im vorangegangenen Kapitel beschriebenen Mechanismus in den aktuellen Tourplan so kostengünstig als möglich einzuplanen versucht. Die einzige Möglichkeit diesen Einfügepunkt zu bestimmen besteht darin, jede einzelne Tour einen 'Kostenvoranschlag' für den aktuellen Auftrag berechnen zu lassen. Diese Vorgehensweise entspricht genau dem Ansatz des parallelen Einfügens (vgl. Seite 28), bei dem jede Tour als potentieller Lieferant angesehen wird. Hierbei gilt es, ein *offenes* Problem zu lösen, d.h. ein Problem, das nicht statisch vorliegt, sondern sich ständig ändert. Da bei jedem neuen Kunden nur ein Bruchteil der Gesamtinformation vorhanden ist, die notwendig wäre, um diesen Auftrag im letztendlichen Kontext 'optimal' einzufügen, ist hier ein sich nahezu aufdrängender Handlungsbedarf für Nachoptimierungsverfahren gegeben. Deren Aufgabe ist es, die zum Teil verworrenen Steinchen zu einem Mosaik zu formen.
Ist die vollständige Auftragsmenge hingegen bekannt, kann man von einem *geschlossenen* Problem sprechen. Sämtliche in Kapitel 4.1 vorgestellten Verfahren (mit Ausnahme des parallelen Einfügens, wenn bei diesem das Sortierkriterium weggelassen wird) sind ausschließlich zum Lösen dieses Problemtyps in der Lage. Da die Kundenliste und deren Forderungen bekannt sind, diese sich auch nicht mehr dynamisch ändern können, ist es möglich Sortierkriterien anzuwenden, die allein die Qualität der Lösung schon erheblich verbessern. Verfahren wie Nearest-Neighbor oder das sequentielle Einfügen gehen sogar noch einen Schritt weiter, die aktuell generierte Tour sucht sich einen Kunden aus der Auftragsliste aus.
Eine Gegenüberstellung bzgl. der Qualität zwischen offenem und geschlossenem Problemlösen in Zusammenhang mit den Solomon-Testdaten wird in Kapitel 6.1 gegeben.

5.4. Simulated-Trading

5.4.1. Idee

Das *Simulated-Trading* wurde am Lehrstuhl von Prof. Bachem am Mathematischen Institut in Köln entwickelt [Bachem et al., 1992].
Es handelt sich um ein Nachoptimierungsverfahren, das auf bereits vorhandene Initiallösungen angewandt werden kann. Prinzipielle Idee des Simulated-Trading ist es, durch 'geschicktes' Austauschen von Aufträgen zwischen den Fahrzeugen, jede einzelne Tour zu verbessern und somit den Gesamttourplan kostengünstiger zu gestalten.
In diesem Kapitel wird zunächst die Idee des Verfahrens beschrieben. Die Ausführungen stützen sich dabei auf [Malich, 1991] und [Bachem et al., 1993]. In dem darauffolgenden Kapitel wird die Anbindung der Heuristik an das Speditionsszenario erklärt.

5.4.2. Definitionen

Definition 5.2 *Ist T eine Tour und $i \in T$, dann sind die Kosten $c^-(T,i)$ für das Entfernen von i aus T gegeben durch*

$$c^-(T,i) = c(T) - c(T - \{i\}).$$

Analog gilt für die Einfügekosten $c^+(T,i)$ mit $i \notin T$

$$c^+(T,i) = c(T + \{i\}) - c(T).$$

5.4.3. Beschreibung des Algorithmus

Prinzipieller Aufbau

$\mathcal{T} = (T_1, \ldots, T_t)$ sei ein gültiger Tourplan
repeat
    Sell-and-Buy-Phase
    finde Trading-Match
    if Trading-Match gefunden **then**
        aktualisiere Tourplan
    fi
until Zeitlimit erreicht

Algorithmus 1: Der Simulated-Trading-Algorithmus

Der Algorithmus ist prinzipiell dreigeteilt. Zuerst finden hypothetische Austauschvorgänge von Kunden zwischen den einzelnen Touren statt (Sell-and-Buy). Wie sich noch zeigen wird, liefern diese Vorgänge im allgemeinen keinen gültigen Tourplan. Dieser muß zuerst ausfindig gemacht und aktualisiert werden.
Nach der Aktualisierung erhält man einen gültigen Tourplan (unter Umständen wird der alte wieder übernommen), auf den das Verfahren erneut anwendbar ist. Dies geschieht solange, bis ein Zeitlimit erreicht ist oder eine feste Anzahl von Iterationen durchgeführt wurden.

Sell-And-Buy-Phase

```
for CurLevel = 1 to MaxLevel
    for CurTour = 1 to MaxTour
        CurTour kauft, verkauft oder tut nichts
        if Kauf then
            füge Kaufknoten in Trading-Graph ein
        fi
        if Verkauf then
            füge Verkaufsknoten in Trading-Graph ein
            aktualisiere Verkaufsliste
        fi
    next CurTour
next CurLevel
```

Algorithmus 2: Sell-And-Buy-Phase

Während der Sell-And-Buy-Phase werden die eigentlichen Austauschvorgänge simuliert. Dies geschieht in sogenannten *Entscheidungsebenen* (engl. decision level). In jeder Ebene kann eine Tour einen Kunden an einer Börse anbieten (sell), aus der Börse auslösen (buy) oder, falls keine geeignete Aktion durchzuführen ist, gar nichts tun.
Die zum Verkauf anstehenden Kunden werden während dieser Phase in einer *Verkaufsliste* (engl. selling list) an der Börse von dem *Stockmanager* verwaltet. Der Stockmanager bestimmt die Aktion, die eine Tour auszuführen hat. Statt die Aktion in fest vorgegebener Reihenfolge zu bestimmen, wird der auszuführende Vorgang durch eine Zufallsfunktion bestimmt, die mit einer Wahrscheinlichkeit von 0.4 eine Verkaufsaktion wählt. Die Anzahl der Entscheidungsebenen ist fest vorgegeben, kann aber, wie wir später noch sehen werden, dynamisch geändert werden (vgl. Seite 42).
Bei jeder Kauf- bzw. Verkaufsaktion einer Tour wird diese in einem sogenannten *Trading-Graph* protokolliert, der in der darauffolgenden Trading-Matching-Search-Phase daraufhin untersucht wird, ob durch diese Austauschaktionen ein neuer gültiger Tourplan entstanden ist. Abbildung 5.4 zeigt einen solchen Graphen, der in Beispiel 5.7 noch diskutiert wird.

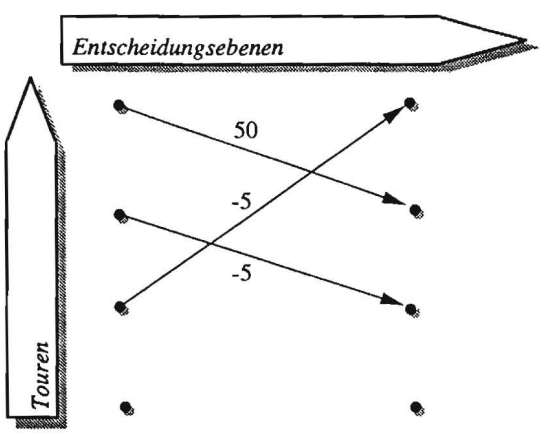

Abbildung 5.4.: Ein Beispiel für einen Trading-Graph

Verkaufen eines Kunden

Definition 5.3 *Sei eine Tour* $T = (0, i_1, \ldots, i_{h-1}, i_h, i_{h+1}, i_k, 0)$ *als Tupel gegeben. Der* ***Verkauf eines Kunden*** i_h *wird definiert durch*

$$T - \{i_h\} = (0, i_1, \ldots, i_{h-1}, i_{h+1}, \ldots, i_k, 0)$$

Ist die Tour leer, d.h. $T = \emptyset$, kann kein Kunde verkauft werden. Ansonsten wird für jeden Kunden i_h eine Wahrscheinlichkeit p_h berechnet, mit der er in der Tour T verkauft wird. Diese ist abhängig von den anfallenden Kosten $c^-(T, i_h)$:

$$p_h = \begin{cases} 1/k & \text{falls } \forall h = 1, \ldots, k : c^-(T, i_h) = 0 \\ \frac{c^-(T,i_h)}{\sum_{j=1}^{k} c^-(T,i_j)} & \text{sonst, } h \in \{1, \ldots, k\}. \end{cases}$$

So kann jeder Kunde i einer Tour T auf einer normierten Leiste zwischen 0 und 1 abgetragen werden, wobei die Breite des jeweiligen X-Abschnitts der Wahrscheinlichkeit entspricht, mit der ein Kunde zum Verkauf angeboten wird.

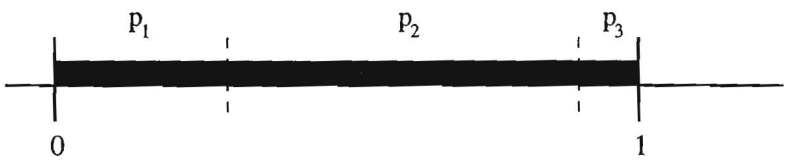

Abbildung 5.5.: Verteilung von Verkaufswahrscheinlichkeiten

Welcher Kunde letztendlich verkauft wird, bestimmt eine Zufallsfunktion, die einen Wert $r \in [0, 1]$ liefert. Dieser Kunde wird dann an der Börse angeboten und dort zusammen mit Informationen über die beim Verkauf angefallenen Kosten (*Savings*) in der Verkaufsliste eingetragen.
Der Stockmanager trägt einen Knoten in den Trading-Graph ein, der folgende Informationen beinhaltet:

- Art der Aktion: buy
- Identifikation der Tour
- Kunde
- Ebenennummer
- Kosten

Kaufen eines Kunden

Definition 5.4 *Sei eine Tour* $T = (0, i_1, \ldots, i_k, 0)$ *als Tupel gegeben. Das billigste* ***Einfügen eines Kunden*** i_h *wird definiert durch*

$$T \cup \{i_h\} = (0, i_1, \ldots, i_j, i_h, i_{j+1}, \ldots, i_k, 0)$$

wobei für $\forall l = 0, \ldots, k$ *gilt*

$$c^+((0, i_1, \ldots, i_j, i_h, i_{j+1}, \ldots, i_k, 0)) \leq c^+((0, i_1, \ldots, i_l, i_h, i_{l+1}, \ldots, i_k, 0))$$

Sei T eine Tour und $S = (i_1, \ldots, i_k)$ eine Liste von Kunden in der Verkaufsliste mit den dazugehörigen Savings $s(1), \ldots, s(k)$. Analog zum Verkauf wird ein Kunde i_h mit Hilfe einer Zufallsfunktion $\forall h = 1, \ldots, k$ aus der Verkaufsliste ausgewählt.

$$p_h = \begin{cases} 1/k & \forall a, b \in \{1, \ldots, k\} : s(a) - c^+(T, i_a) = s(b) - c^+(T, i_b) \\ \frac{s(h) - c^+(T, i_h) - m}{\sum_{j=1}^{k} (s(j) - c^+(T, i_j)) - km} & \text{sonst,} \end{cases}$$

wobei m definiert wird als

$$m = \min_{j=1,\ldots,k} s(j) - c^+(T, i_j)$$

Auch hier findet eine Normierung der Wahrscheinlichkeiten über $[0, 1]$ statt. m garantiert dabei, daß die untere Schranke des Intervalls bei negativem kleinsten Gewinn $s(j) - c^+(T, i_j)$ nicht kleiner 0 wird. Analog sorgt m bei positivem kleinsten Gewinn für eine Verschiebung nach links entlang der X-Achse.

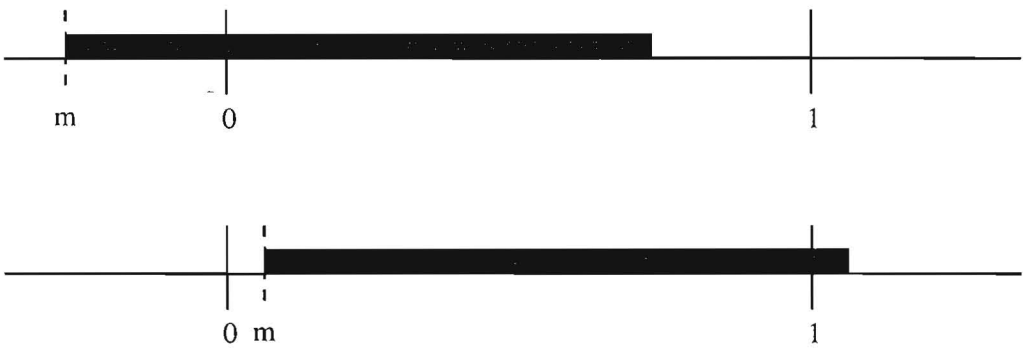

Abbildung 5.6.: X-Achsen-Verschiebung m

Bei dieser Definition treten zwei Probleme auf:

1. der am wenigsten attraktive Kunde erhält immer eine Wahrscheinlichkeit von 0, da für diesen Kunden $s(h) - c^+(T, i_h) = m$;
2. werden nur Kunden mit negativem Gewinn angeboten, ist es oft besser keinen einzukaufen, abhängig von dem zu erwartenden Verlust.

Um dies zu umgehen, wird der Wert von m um 3% verkleinert, um sicherzugehen, daß der am wenigsten attraktive Kunde eine positive Wahrscheinlichkeit erhält. Des weiteren wird ein *Dummy*-Verkaufs-Kunde i_{k+1} der Verkaufsliste mit $s(k+1) - c^+(T, i_{k+1}) = 0$ angehängt. Wird dieser Dummy-Kunde ausgewählt, wird nichts getan und mit der Iteration fortgefahren.
Kunden, die an der Börse gekauft werden, bleiben weiterhin in der Verkaufsliste enthalten, damit sie in späteren Ebenen auch noch von anderen Touren eingekauft werden können. Dies führt zu sehr komplexen Verschiebungen, was nach der Sell-And-Buy-Phase in aller Regel zu einem ungültigen Tourplan führt, da beispielsweise ein in der Verkaufsliste angebotener Kunde von mehreren Touren eingekauft wurde oder für andere Kunden kein Käufer gefunden werden konnte.
Analog zum Verkauf eines Kunden wird vom Stockmanager ein Knoten in den Trading-Graph eingefügt, der Informationen bzgl. Art der Aktion, Touridentifikation, ... enthält.

Trading-Matching-Search-Phase

Der bei der Sell-And-Buy-Phase entstehende neue Tourplan ist im allgemeinen ungültig. Dies macht eine Suche im Trading-Graph notwendig, einen gültigen Tourplan zu finden. Zunächst sollen der Trading-Graph selbst und das Trading-Matching definiert werden.

Definition 5.5 (Trading-Graph) *Sei t, n, $Max \in N$ mit $t \leq n$, die Sell-And-Buy-Phase von Ebene 1 bis $Max-1$ gelaufen. Der entsprechende* ***Trading-Graph*** *ist definiert durch den Graphen G:*

$$G = (V = V_s \cup V_b, E) \quad mit \quad V \subset \{1,\ldots,t\} \times \{1,\ldots,Max\} \times \{1,\ldots,n\}.$$

Wobei

$$v = (i,l,k) \in V_s \Leftrightarrow \text{tour } T_i^{l-1} \text{ verkauft Kunde } k \text{ in Ebene } l \tag{5.1}$$

$$v = (i,l,k) \in V_b \Leftrightarrow \text{tour } T_i^{l-1} \text{ kauft Kunde } k \text{ in Ebene } l \tag{5.2}$$

$$(v,w) \in E \Leftrightarrow v \in V_s, w \in V_b, v = (.,.,k), w = (.,.,k) \tag{5.3}$$

Weiterhin ist die Gewichtung $\omega(e)$ einer Kante $e = (v,w)$ mit $v = (i,l,k) \in V_s$ und $w = (j,m,k) \in V_b$ gegeben durch

$$\omega(e) = c^-(T_i^{l-k},k) - c^+(T_j^{m-1},k). \tag{5.4}$$

Gleichung (5.1) besagt, daß genau dann ein Verkaufsknoten v in der aktuellen Ebene l in den Trading-Graph eingefügt wurde, wenn ein Kunde k von der Tour T_i^{l-1} verkauft wurde. Analog wird nach (5.2) ein Kaufknoten v in den Trading-Graph eingefügt, falls ein Kunde k von T_i^{l-1} gekauft wurde. Eine gerichtete Kante e von v nach w wird genau dann eingetragen, wenn v ein Verkaufs- und w ein Kaufknoten bzgl. des gleichen Kunden k ist (5.3).
Jedes Matching des Trading-Graph korrespondiert mit einem Austausch von Kunden. Ist der Wert des Matching positiv, konnte der aktuelle Tourplan verbessert werden. Um die Gültigkeit des neuen Tourplans zu garantieren, müssen zusätzliche Restriktionen eingeführt werden.

Definition 5.6 (Trading-Matching) *Sei $G = (V = V_s \cup V_b, E)$ ein Trading-Graph mit $V \subset \{1,\ldots,t\} \times \{1,\ldots,Max\} \times \{1,\ldots,n\}$. Weiterhin sei $\emptyset \neq M \subset E$ eine Menge von Kanten, und*

$$M_v = \{v \in V \mid \exists w \in V\ (v,w) \in M\}$$

die Menge von gesättigten Knoten. M ist ein ***Trading-Matching*** $\Leftrightarrow$

$$\forall v \in M_v \ : \ \exists_1 (v,.) \in M \tag{5.5}$$

$$\forall (i,l,k) \in M_v \quad \forall l^* = 1,\ldots,l-1 \ : \ (i,l^*,.) \in M_v. \tag{5.6}$$

Der ***Wert*** *eines Trading-Matching M ist definiert durch*

$$\omega(M) = \sum_{e \in M} \omega(e). \tag{5.7}$$

Bedingung (5.5) formalisiert die Aussage, daß zu jedem Verkaufsknoten, der an dem Matching beteiligt ist, genau ein Kaufknoten vorhanden ist. (5.6) besagt, daß für jedes Matching in Ebene l bzgl. einer Tour T_i^l die Aktionen der vorausgegangenen Ebenen $l^* < l$

dieser Tour i auch gematcht sein müssen. Diese Restriktion wird auch *Levelbedingung* genannt.
Die Bedingung impliziert, daß die Kaufaktionen die 'wertvolleren' Handlungen sind, da nur sie etwaige Verletzungen der Levelbedingung aufheben können. Dies spiegelt sich auch in der Wahrscheinlichkeit wieder, mit der eine Kaufaktion 60:40 einer Verkaufaktion vorgezogen wird. Um beispielsweise einen Kunden neu in eine bestehende Tour aufzunehmen, muß u.U. sichergestellt werden, daß vorher Kunden verkauft wurden, um den neuen Kauf erst zu ermöglichen.

Beispiel 5.7 In Abbildung 5.4 liegt ein Trading-Graph vor, der zu einem Matching führt. Jedem beteiligten Verkaufsknoten kann genau ein Kaufknoten zugewiesen werden (5.5). Allen Aktionen in Ebene 2 gehen Aktionen in Ebene 1 voraus, die ebenfalls zu dem Trading-Matching gehören (5.6). Der Gesamtgewinn des Matchings beträgt 40 Kosteneinheiten.
Abbildung 5.7 dagegen ist ein Beispiel für einen Trading-Graph in dem kein Trading-Matching möglich ist.

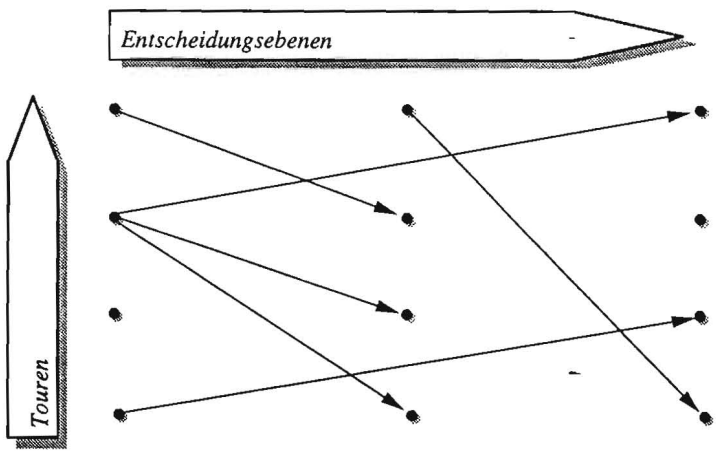

Abbildung 5.7.: Ein Trading-Graph ohne Trading-Matching

Satz 5.8 *Das Problem, in einem Trading-Graph G ein Trading-Matching zu finden, ist NP-vollständig, selbst wenn G ein Graph mit nur drei Ebenen ist.*

Der Beweis ist in [Bachem et al., 1993] zu finden.

Es scheint verwunderlich, warum ein NP-vollständiges Problem durch ein anderes NP-vollständiges Problem gelöst werden soll. Der Trading-Graph und der damit verbundene Suchraum ist in aller Regel jedoch so klein, daß die Suche selbst, relativ zur restlichen Programmausführung, kaum ins Gewicht fällt.

Aktualisierung

Nach den Austauschvorgängen und dem Finden eines Trading-Matchings M muß der Tourplan $\mathcal{T} = (T_1, \ldots, T_t)$ aktualisiert werden:
Da nach Definition 5.6 das Trading-Matching nach einem Kundenaustausch wieder einen gültigen Tourplan findet, sofern einer vorhanden ist, führt der Aktualisierungsschritt zu einem neuen gültigen Tourplan.

```
for i = 1 to t -
    l* = max{l|(i, l, .) ∈ M_v} ∪ 0
    T_i = T_i^{l*}
next i
```

Algorithmus 3: Aktualisieren eines Tourplans

5.4.4. Erweiterungen

Um die Qualität und Effizienz der Ausführung zu steigern, wurden dem Verfahren Heuristiken hinzugefügt, die im folgenden erklärt werden sollen.

Verschlechterung

Der Tourplan der Initiallösung ist im allgemeinen nicht optimal. Dies führt dazu, daß bestimmte Tourabschnitte ΔT_i zwar zueinander in optimaler Reihenfolge liegen, der Abschnitt jedoch einen insgesamt kostengünstigeren Tourplan verhindert. Diese Gruppierungen aufzubrechen ist unter Umständen ohne eine temporäre Verschlecherung ω^* nicht möglich, d.h. beim Trading-Matching wird zumindest anfangs auch ein negativer Wert zugelassen (vgl. 5.4.5).

Dynamische Entscheidungsebenen

Zu Beginn des Simulated-Trading Laufes werden in aller Regel 'einfache' Austauschvorgänge sehr schnell gefunden. Diese 'offensichtlichen' Verschiebungen von Kunden innerhalb eines Tourplans $\mathcal{T}$ werden bereits mit wenigen Entscheidungsebenen erreicht. Daher wird das Verfahren mit wenigen Ebenen gestartet und dynamisch je nach Abnahme der Effizienz gesteigert. In der Praxis hat sich bewährt, mit zwei Ebenen zu beginnen und falls sich die Qualität des Gesamttourplans nach drei Iterationen nicht verbessert hat, diese um eins zu erhöhen. Maximal wurde eine Tiefe von 6 bis 8 gewählt.

5.4.5. Parameter

Entscheidend für die Qualität des Simulated-Trading ist die Wahl der Parameter, die den Verlauf der Programmausführung bestimmen. Diese sind im Einzelnen:

- Die maximale Anzahl der Entscheidungsebenen Max sollte nicht zu klein gewählt werden, da es sonst zu keinen komplexen Austauschvorgängen kommt. Auch zu große Max verbessern die Qualität der Lösung kaum noch, benötigen aber unverhältnismäßig viel Zeit. $Max \in \{2, \ldots, 8\}$ hat sich als vorteilhaft erwiesen.

- Die Verschlechterung ω^* wird dynamisch gesteuert. Der Initialwert des zu akzeptierenden Trading Wertes wird durch $\omega^* = -0.02 * c(\mathcal{T})$ gegeben, d.h. anfangs wird eine Verschlechterung des Gesamttourplans von 2% erlaubt. Diese wird pro Iteration kontinuierlich verkleinert.

5.5. Simulated-Trading im Szenario

Das ursprünglich auf Transputern implementierte Simulated-Trading (vgl. auch [Malich, 1991]) stellt eine nahezu ideale Erweiterung des MAS-MARS Systems dar. Jede Spedition kann durch ein Modul erweitert werden, das die Funktionalität der beschriebenen Heuristik aufweist. Die Spedition selbst übernimmt dabei die Master-Funktion des Stockmanagers, um die einzelnen Aktionen und beteiligten Agenten zu koordinieren. Abbildung 5.8 illustriert die zentrale Stellung.

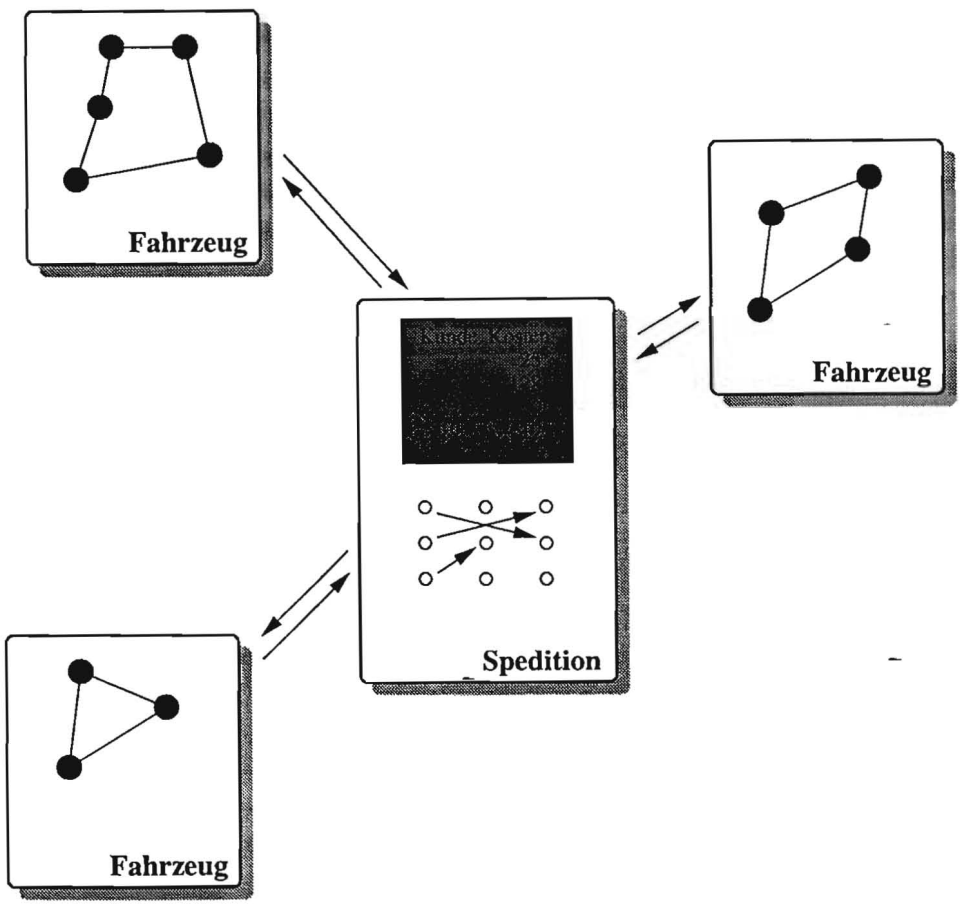

Abbildung 5.8.: Die Agentenstruktur des Simulated-Trading im Szenario

5.5.1. Sell-And-Buy-Phase

Innerhalb einer jeden Entscheidungsebene versuchen die einzelnen Fahrzeugagenten einen Kunden von der Börse in ihre bestehende Tour gewinnbringend einzufügen, einen Kunden aus der aktuellen Tour zu verkaufen, oder falls keine geeignete Aktion durchzuführen ist, gar nichts zu tun.
Der Fahrzeugagent erhält zusammen mit der Verkaufsliste von seiner Spedition den Auftrag, sich an dem Trading-Prozeß zu beteiligen. Dieser bestimmt nun selbstständig, was er als nächstes tut. In diesem Punkt unterscheidet sich der im letzten Kapitel vorgestellte Ansatz von unserer Implementierung (vgl. Seite 37), bei dem von einer übergeordneten Instanz dem Slave-Agenten (Fahrzeug) eine Aktion 'aufgezwungen' wird.

Zum Finden eines Trading-Matching ist es entscheidend, daß zu Verkäufen entsprechende Käufer gefunden werden (vgl. Seite 41). Deswegen wird zuerst versucht, einen Kunden aus der Börse auszulösen und diesen in die aktuelle Tour einzuplanen. Wird kein entsprechender Kunde gefunden, wird im zweiten Schritt eine Zufallsheuristik angewandt, die in Abhängigkeit der Länge der aktuellen Tour T_i bestimmt, ob ein Kunde verkauft oder nichts getan wird. Die Wahrscheinlichkeit $r(T_i)$ für einen Verkauf wird durch

$$r(T_i) = \max(1 - |\Pi T_i| * 0.1, 0.4)$$

festgelegt, wobei $|\Pi T_i|$ die Anzahl der Kunden ist, die von Tour T_i beliefert werden. Durch die Heuristik soll bewirkt werden, daß kurze Touren (Touren, die nur wenige Kunden bedienen) bevorzugt Verkaufsaktionen tätigen, damit sie unter Umständen aufgelöst werden. So beträgt die Wahrscheinlichkeit, daß auf einer Tour T_i mit nur einem Kunden dieser auch verkauft wird 90%. Die minimale Verkaufswahrscheinlichkeit beträgt analog zur ursprünglichen Heuristik 40%.
Der Fahrzeugagent meldet an den Stockmanager die Art der Aktion (sell, buy, void), die dabei angefallenen Kosten/Gewinn und den Kunden, der eingefügt oder entfernt wurde. Der Stockmanager selbst trägt nun diese Daten zusammen mit der Ebenennummer und einer Fahrzeug-Kennung (Id) in den Trading-Graph ein und aktualisiert bei Bedarf die Verkaufsliste. Wird als Aktion *void* zurückgeliefert, wird einfach mit dem Trading-Prozeß fortgefahren.
Die einzelnen Ebenen werden vom Stockmanager synchronisiert, d.h. die nächste Ebene beginnt erst, wenn von allen Fahrzeugagenten eine Rückmeldung erhalten wurde. Würde der Stockmanager nur einen Anfangszeitpunkt bestimmen, ab dem die Agenten eine festgelegte Anzahl von Entscheidungsebenen durchführen sollen, hätte dies zur Folge, daß kürzere Touren u.U. schon fertig wären, bevor längere überhaupt die zweite Ebene erreicht hätten. D.h. kürzere Touren würden bei dem Gesamtprozeß 'übergangen'.

5.5.2. Trading-Matching-Search-Phase

Ist die maximale Entscheidungsebene erreicht, und haben alle Fahrzeugagenten ihre Aktionen beendet, beginnt die Phase, in der in dem zuvor generierten Trading-Graphen ein Matching gesucht wird.
Da Kunden in der Verkaufsliste nie aus dieser entfernt werden, kommt es in aller Regel vor, daß einige dieser Kunden von mehreren Fahrzeugen in ihre Tour eingeplant (gekauft) wurden. Für andere Kunden aus der Liste mag es wiederum überhaupt keine Interessenten geben. Aufgabe der Suche ist es nun das Matching mit dem größten Wert (vgl. Definition 5.6) zu finden, falls ein solches überhaupt existiert. D.h. zu je einem Verkauf muß genau ein Käufer gefunden werden. Es ist nicht zwingend notwendig, daß jeder Austausch von Kunden mit einem positiven Wert verbunden ist. Entscheidend ist, daß der Gesamtwert des Matching einen Schwellwert nicht unterschreitet (vgl. Kapitel 5.4.4). Dazu muß aber u.U. partiell eine Verschlechterung hingenommen werden.
Entscheidendes Kriterium für die Gültigkeit des Trading Matching ist die auf Seite 41 definierte Levelbedingung. Aufgrund dieser Bedingung reicht es nicht aus, zu jedem Verkaufsknoten im Trading-Graph einen Kaufknoten zu finden. Stattdessen muß sichergestellt sein, daß die jeweils vorangegangenen Aktionen, auch an einem Match beteiligt sind (vgl. auch Beispiel 5.7). Wurde also ein Paar gefunden, das einen gültigen Kundenaustausch darstellt, werden alle Vorgängerknoten in einer *Queue* abgelegt und daraufhin untersucht, ob für diese ein korrespondierender Knoten im Graphen gefunden werden kann.

5.5.3. Aktualisierung

Als Ergebnis der Suchphase erhält der Stockmanager eine Liste von Knotenpaaren, die jeweils einen gültigen Kundenaustausch darstellen. Wurde kein Matching gefunden, wird eine leere Liste an ihn zurückgegeben. Aufgrund dieser Liste kann der Stockmanager genau feststellen, welche Touren an dem Match letztendlich beteiligt waren. Die entsprechenden Informationen werden an den jeweiligen Fahrzeugagenten weitergegeben, der wiederum seine lokale Tour aufgrund dieser Daten aktualisiert.

5.6. Oz

Oz ist eine am DFKI entwickelte Hochsprache, die für nebenläufige, symbolische Berechnungen entworfen wurde [Smolka, 1995]. Sie basiert auf einem neuen Berechnungsmodell, das eine uniforme und einfache Grundlage für verschiedene Programmierparadigmen unterstützt, funktionale, constraint-logische und nebenläufig objektorientierte Programmierung eingeschlossen (siehe auch [Müller et al., 1995]). Oz sieht sich als Nachfolger von Programmiersprachen wie Lisp, Prolog oder Smalltalk, denen Aspekte der Nebenläufigkeit, Reaktionsfähigkeit oder Echtzeitkontrolle fehlen und stellt so eine geeignete Plattform zur Konzipierung von Multiagentensystemen dar.
Oz basiert auf Logischen Variablen, d.h. Variablen können benutzt werden, bevor ihnen ein Wert zugewiesen wurde. Wird während einer Berechnung eine Variable benötigt, die noch nicht bekannt ist, wird die Berechnung suspendiert und automatisch zu dem Zeitpunkt wieder aufgenommen, zu dem der Wert bekannt wird. Auch kann mit Variablen gerechnet werden, deren Wert nur teilweise spezifiziert wurde (wie z.B. $V \in \{1, \ldots, 10\}$). Diese Einschränkungen werden durch Constraints (engl. Zwang, Einschränkung) beschrieben. Ein von Oz angebotener Mechanismus zur Graphensuche wird noch in Kapitel 7.4 näher untersucht.
DFKI Oz[2] ist eine interaktive Implementierung von Oz, basierend auf GNU Emacs. Es stellt eine Reihe von vordefinierten Klassen zu Verfügung, die beispielsweise Socket-Kommunikation oder dynamisches Linken von C/C^{++}-Modulen unterstützen.
Oz ist eine moderne Programmiersprachen, die viele Konzepte der KI-Programmierung erstmals geschlossen umsetzt. Dabei wurden gerade die guten Eigenschaften bestehender Systeme übernommen und erweitert. Der gute Gesamteindruck wird lediglich durch das gänzliche Fehlen brauchbarer Werkzeuge zur Fehlersuche getrübt, was sich besonders bei größeren Anwendungen durch einen höheren Zeitaufwand bei der Programmierung bemerkbar macht. Solche Werkzeuge befinden sich jedoch bereits in der Entwicklung.

[2]DFKI Oz ist kostenlos für verschiedene Plattformen (Unix, Linux, RS6000, ...) unter `http://www.dfki.uni-sb.de/ps/` (WWW) bzw. `ps-ftp.dfki.uni-sb.de` (ftp) erhältlich.

6. Ergebnisse

Die von unserem System generierten Ergebnisse sollen nun mit denen aus der Literatur bekannten verglichen werden. Die einzelnen Detail-Tabellen, auf die in diesem Kapitel verwiesen wird, sind im Anhang A zu finden.

6.1. Solomon

Wie bereits in Kapitel 4.1.2 erwähnt, hat [Solomon, 1987] eine Reihe von Testdaten für das VRPTW generiert[1]. Diese Daten bauen auf einer Menge von Problemen auf, die [Christofides et al., 1979] für das VRP ohne Zeitrestriktionen entwickelt haben. Von diesen wurden z.T. die Koordinaten und die Auftragsquantitäten übernommen.
Insgesamt handelt es sich um sechs verschiedene Datensätze, die alle ihre eigenen Charakteristika bzgl. Geometrie, Anzahl der von einer Tour bedienbaren Kunden und Zeitrestriktionen aufweisen. Tabelle 6.1 vermittelt einen Überblick.

	Schedulinghorizont	Clusterbildung
R1	klein	keine
R2	groß	keine
RC1	klein	mittel
RC2	groß	mittel
C1	klein	groß
C2	groß	groß

Tabelle 6.1.: Charakterisierung der Solomon-Datensätze

Schedulinghorizont bedeutet die (relative) zeitliche Begrenzung, wann eine Tour spätestens beendet werden muß. Von großem Schedulinghorizont läßt sich also auf wenige, relativ lange Touren schließen. Clusterbildung gibt (relativen) Aufschluß darüber, wie nah einzelne Gruppen von Kunden zueinander liegen. Ein Beispiel für einen geclusterten Routing-Graph stellen Abbildung 6.2 und 6.3 dar.
Je Datensatz gibt es zwischen 8 und 12 einzelne Probleme, die sich untereinander nur bzgl. der Zeitfenster unterscheiden. Die geographische Lage und die Quantität des zu befördernden Transportgutes bleiben gleich.
Solomon hat in seiner Arbeit verschiedene heuristische Lösungsansätze untersucht (vgl. Seite 29) und diese miteinander verglichen. Die von ihm erhaltenen Ergebnisse sollen hier aufbereitet werden, um sie im Anschluß mit den eigenen zu vergleichen.
Im einzelnen wurden folgende Verfahren analysiert:

- Savings (mit und ohne Wartezeitlimitierung)

[1] Die Daten sind direkt beim Autor (SOLOMON@neu.edu) erhältlich.

- Sequentielles Insertion-Verfahren (mit verschiedenen Auswahl- und Einfügekriterien)
- Nearest Neighbor
- Sweep

Das sequentielle Einfügen wurden in drei Variationen durchgeführt, bei denen sich die Auswahl- und Einfügekriterien änderten. Da die auf Seite 26 beschriebenen Kriterien im allgemeinen jedoch die besten Ergebnisse bzgl. der drei Varianten liefern, wird auf die Ausführung der verbleibenden zwei verzichtet. Analog wird auf die Auflistung des Savings-Verfahrens ohne Wartezeitlimitierung verzichtet, da deren Begrenzung immer der allgemeineren Heuristik überlegen war.
Es wurden je Heuristik mehrere Testläufe durchgeführt, wobei sich lediglich die Parameter änderten. Der jeweils beste Lauf wurde zur Wertung herangezogen.

Beispiel 6.1 Das Insertion-Verfahren wurde mit folgenden Parametern variiert (siehe dazu auch Seite 27):

$$(\mu, \lambda, \alpha_1, \alpha_2) \in \{(1,1,1,0), (1,2,1,0), (1,1,0,1), (1,2,0,1)\}.$$

Zwei Initialisierungskriterien wurden getestet:

- der am weitesten entfernte Kunde (*farthest customer*),
- der Kunde mit der frühesten *dda* (*earliest deadline*).

Dies ergibt insgesamt acht Testläufe.

Da die detaillierte Ausführung aller Datensätze zu umfangreich wäre, wird dies nur examplarisch für die Menge R1 getan. Tabelle 6.2 gibt einen Überblick über die generierten Ergebnisse. Dabei wurden jeweils 100 Kunden angefahren. Die entsprechenden Daten ergeben sich als Durchschnitt über alle zu einer Klasse gehörenden Probleme. Analoge Tabellen bzgl. den restlichen Problemmengen sind in der Quelle zu finden.

Heuristik	# Fahrzeuge	Distanz	Dauer	Wartezeit
Savings	15,1	1517,2	2925,2	617,4
Insertion	13,6	1436,7	2695,5	258,8
Nearest Neighbor	14,5	1600,1	2968,7	368,0
Sweep	14,6	1499,0	2817,4	317,7

Tabelle 6.2.: Vergleich heuristischer Lösungsmethoden bzgl. R1

Tabellen A.4 und A.5 zeigen im Vergleich die von unserem System berechneten Ergebnisse im Detail. Die durchschnittliche Lösung ist in Tabelle 6.3 zu finden.

	# Fahrzeuge	Distanz	Dauer	Wartezeit
Initial	13,6	1417,9	2698,1	280,2
Trading	13,25	1273,1	2626,5	353,4

Tabelle 6.3.: R1 im Mittel

Die obere Hälfte entspricht dem sequentiellen Insertion-Verfahren, welches durch das Simulated-Trading weiter verbessert wurde (untere Hälfte). Die Nachoptimierung steigert die Qualität der Lösung um etwa weitere 10% gegenüber der Initiallösung. Bemerkenswert ist auch, daß bereits unsere Initiallösung die Distanz der besten Solomon-Lösung unterbietet.

6.1.1. Vergleich mit optimalen Lösungen

[Desrochers et al., 1992] lösten drei der sechs oben beschriebenen Problemgruppen mit einem Branch & Bound-Ansatz optimal bzgl. der Distanz (vgl. S. 23). Dies waren R1, RC1 und C1. Die jeweiligen Probleme einer Datenmenge wurden dazu wiederum in drei Klassen unterteilt, die sich in der Anzahl der zu beliefernden Kunden unterschieden (25, 50, 100). Da das Verfahren extrem zeitaufwendig ist, konnten die Probleme, die 100 Kunden zu beliefern hatten, leider nur teilweise gelöst werden (so von der R1-Datenmenge lediglich zwei, bzgl. RC1 keine). Da nur bei einer Auftragsmenge von 25 Kunden vollständige Ergebnisse vorliegen (Tabelle A.8), sollen diese mit unseren in Tabelle A.6 und A.7 verglichen werden. Grafik 6.1 illustriert dies.

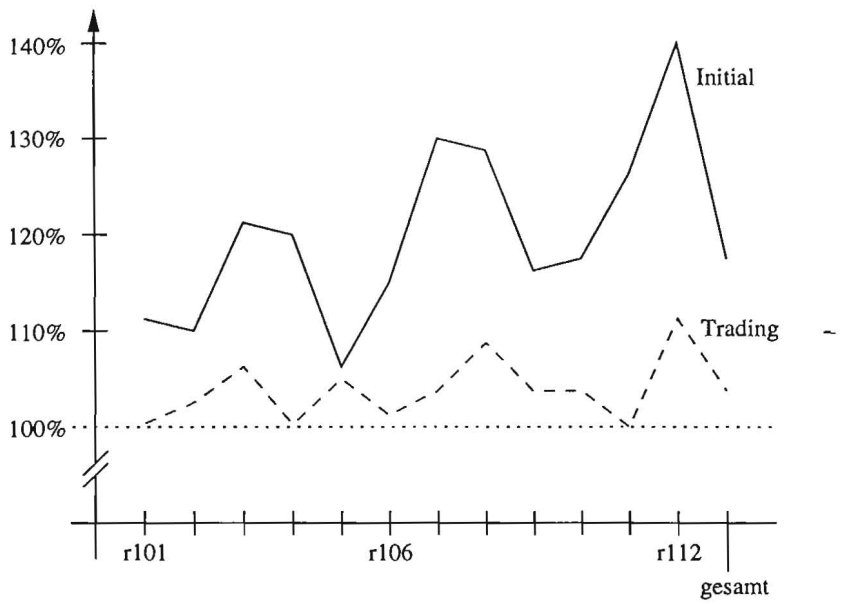

Abbildung 6.1.: Vergleich von Lösungen bzgl. Datensatz R1 mit 25 Kunden

Die gepunktete Linie entspricht dem Optimum (100%) an Distanz. Die oberste, durchgezogene Kurve visualisiert Tabelle A.6 und gibt somit unsere Initiallösung wieder, die mit dem sequentiellen Einfügen mit entsprechender Sortierung gewonnen wurde. Die gebrochene Linie entspricht Tabelle A.7. Der Unterschied zur optimalen Lösung bzgl. der Distanz beträgt im Schnitt lediglich 4%.
Die Anzahl der Fahrzeuge nach Anwendung des Simulated-Trading ist identisch mit der optimalen Lösung, wenn auch im Detail Unterschiede auftreten. So benötigt unser System bei der r105-Problemstellung ein Fahrzeug weniger, als das Optimum, muß deswegen aber auch eine größere Entfernung zurücklegen. An diesem Beispiel läßt sich auch gut erkennen, daß die kürzestmögliche Distanz nicht automatisch ein Minimum an Fahrzeugen mit sich bringt.

6.1.2. Einfluß der Sortierverfahren

Wie bereits in Kapitel 5.3 beschrieben, ist für die Qualität der berechneten Lösung zum einen entscheidend, ob ausreichende Informationen bzgl. aller Kunden zu Beginn der Berechnung zur Verfügung stehen (offenes vs. geschlossenes Problemlösen), zum anderen ist einflußreich, wie geschickt die Auftragsmenge geordnet wird, falls es sich um ein geschlossenes Problem handelt.
Tabelle A.1 dokumentiert den einzigen offenen Lösungsversuch, d.h. die Kunden wurden mit Hilfe des parallelen Insertion-Verfahrens versucht, in der vorgegebenen Reihenfolge in die bestehenden Touren einzuplanen. Da nur Informationen der Kunden bekannt sind, die bisher eingeplant wurden, ist die Qualität der Lösung erwartungsgemäß schlechter, als die folgenden.
Deutlich zu erkennen ist der Qualitätssprung von unsortierten Eingaben zu sortierten (Tabelle A.2). Der Übergang zum sequentiellen Einfügen (Tabellen A.3, A.4) verbessert die Lösung erneut, deren Qualität nur noch durch das Simulated-Trading gesteigert werden kann (Tabelle A.5).

6.1.3. Zusammenfassung der Ergebnisse

Insgesamt liegt die mit dem Simulated-Trading erreichte Verbesserung bzgl. der Distanz im Schnitt bei 10-12%. Vor allem kleine Auftragsmengen (vgl. Tabelle A.7) aber auch geclusterte Problemsätze konnten mühelos auf (fast) optimalen Lösungen abgebildet werden. Da, wie bereits erwähnt, nur von den wenigsten Problemen auch wirklich solche Vergleichsergebnisse vorliegen, hilft oft eine Inspektion des Tourplans. Abbildung 6.2 illustriert eine Initiallösung für das Problem C208 , die mit dem Parallel-Insertion berechnet wurde. Abbildung 6.3 zeigt eine Lösung für das gleiche Problem, jedoch nach Anwendung des Simulated-Trading.

6.2. Bachem

Tabelle A.9 gibt die Ergebnisse wieder, die [Bachem et al., 1993] bzgl. der Solomon-Datenmenge R1 veröffentlicht haben. Leider lassen sich diese nicht direkt mit Tabelle A.5 vergleichen, da lediglich die Zeitdauer bekannt ist, wie lange eine Tour dauert. Über Distanzeinheiten wurde keine Aussage gemacht. Die von uns gewählte Insertion-Heuristik gibt im allgemeinen einer distanzorientierten Optimierung den Vorzug (vgl. Kapitel 5.2), wobei sich die Zeit nicht zwingend in gleichem Maße reduzieren muß. Die Zeitdauer in Tabelle A.5 setzt sich aus Fahrt-, Warte- und Service-Zeiten bei den einzelnen Kunden zusammen. Sollten diese mit Tabelle A.9 verglichen werden, ist zu beachten, daß dort die Service-Zeiten bereits subtrahiert wurden. Die Entladeaktionen bei den Kunden benötigen insgesamt 1000 Zeiteinheiten.
Da die Trading-Heuristik eine Reihe von 'Zufälligkeiten' beinhaltet, ist die generierte Lösung nicht immer gleich. Tabelle A.9 verleiht einen Überblick über die besten Ergebnisse, die mit diesem Verfahren erzielt werden konnten. Eine Erweiterung der von uns implementierten Heuristik, wie sie auch in ähnlicher Form in Köln Anwendung findet, wird in Kapitel 7.4 diskutiert.

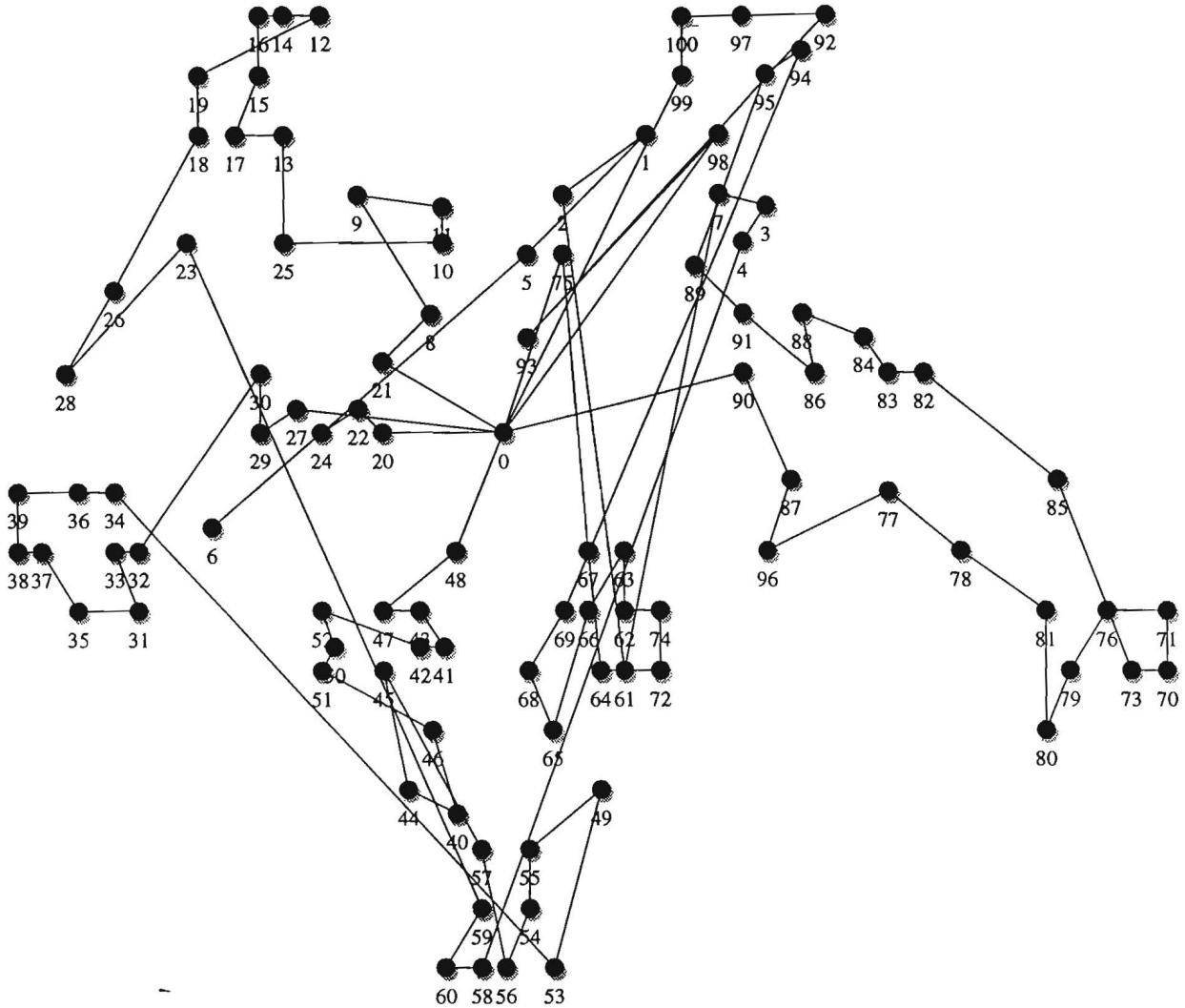

Abbildung 6.2.: Initiallösung C208

6.3. Daimler

Von der Forschungsabteilung der Daimler Benz AG wurde uns freundlicherweise ein Datensatz zu Verfügung gestellt, dessen Routing-Graph 876 Knoten enthält. Es handelt sich um ein reales Beispiel aus dem Werknahverkehr mit sehr engen Zeitfenstern. Aufgabe der Fahrzeugflotte ist die Auslieferung von Backwaren einer Großbäckerei. Abbildung 6.4 zeigt die Knotenmenge des entsprechenden Graphen.
Insgesamt handelt es sich um ein modifiziertes VRPTW, da drei verschiedene Fahrzeugtypen zur Verfügung stehen, die sich in ihren Kapazitäten und Kosten unterscheiden:

Typ	Kapazität	Fixkosten	variable Kosten
1	410	180,00	0,0050
2	481	198,00	0,0055
3	792	216,00	0,0070

Eine von der Daimler Benz AG mit einem modifizierten Savings-Verfahren gefundene Lösung (fast ohne Nachoptimierung) enthält 60 Touren und Planungskosten von 13005,59 Geldeinheiten (GE) bei 392958 Längeneinheiten (LE). Der größte Anteil fällt dabei mit 10980 GE auf die Fixkosten.

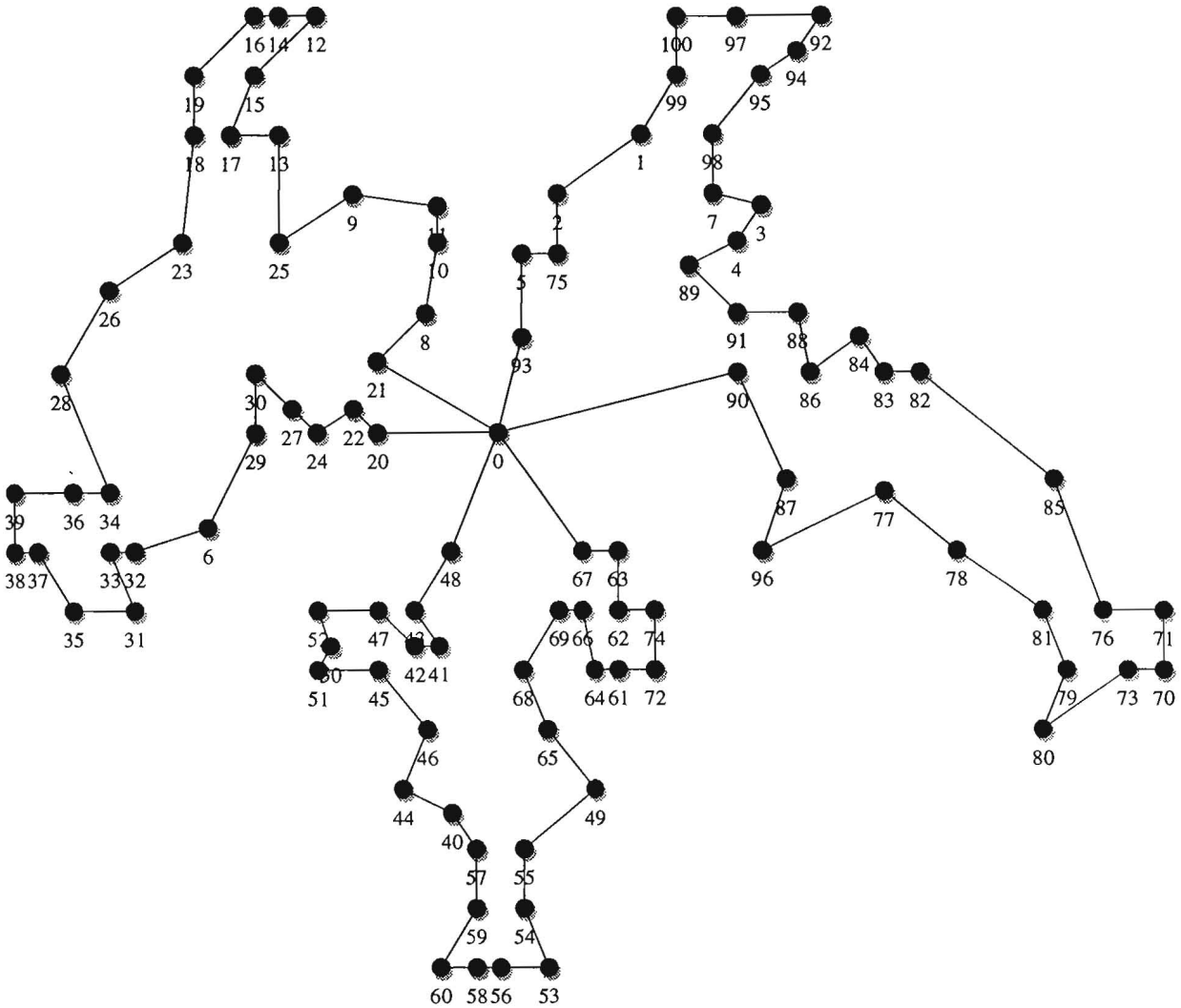

Abbildung 6.3.: Ein 'offensichtlich' optimaler Tourplan C208

Eine von uns gefundene Initiallösung benötigte 61 Touren, jedoch nur 390070 LE. Für die Gesamtkosten ergeben sich 13175.33 GE, die durch das Simulated-Trading um weitere 49 GE auf 390021 GE gesenkt wurden.

6.4. Zusammenfassung

Insgesamt läßt sich feststellen, daß das MAS-MARS-System in Bezug auf die Qualität der berechneten Lösungen sehr gute Ergebnisse liefert, die zum Teil aus der Literatur bekannte 'Referenzlösungen' überbieten. Dabei bietet das Paradigma der Multiagentensysteme Möglichkeiten, die weit über den reinen Planungsvorgang hinaus gehen und eine realitätsbezogene Modellierung der Problematik ermöglichen. Eine Verteilung der einzelnen Agenten auf physikalisch getrennte Systeme ist ohne weiteres realisierbar. So wäre es durchaus denkbar, in jedem Fahrzeug in der Realität eine eigene Planungseinheit zu installieren. Die gleichen Konzepte, wie sie im vorangegangenen Kapitel beschrieben wurden, wären problemlos übertragbar. Die Kommunikation zwischen Spedition und den Fahrzeugen könnte über Funk stattfinden, die Kommunikation zwischen Broker und Spedition beispielsweise

über Telefon. Eine Planung und prototypische Realisierung solch einer Vorgehensweise in der Realität findet derzeit im DFKI-Projekt TELETRUCK statt.
Weitere Ansätze zur Erweiterungen des offenen Systems werden im folgenden Kapitel beschrieben.

Abbildung 6.4.: Die Daimler-Traube mit 876 Kunden

7. Ausblick

7.1. InteRRaP

Das System *InteRRaP (**Inte**gration of **R**eactive Behaviour and **Ra**tional **P**lanning)* basiert auf einer Agentenarchitektur, die entwickelt wurde, um eine Basis für die Modellierung dynamischer *agent societies* (Vereinigung von Agenten) zu schaffen [Müller and Pischel, 1993]. Hauptmerkmal von InteRRaP ist die Kombination von Verhaltensmustern mit expliziten Planungseigenschaften. Verhaltensmuster auf der einen Seite erlauben einem Agenten schnell und flexibel auf eine sich ändernde Umwelt zu reagieren. Die Fähigkeit, auf der anderen Seite Pläne zu erstellen, ist notwendig, um schwierigere Aufgaben zu bewältigen.
InteRRaP unterscheidet zwischen einer hierarchischen Wissensbasis und einer mehrstufigen Kontrolleinheit (engl. control unit). Abbildung 7.1 verdeutlicht die einzelnen Bestandteile.

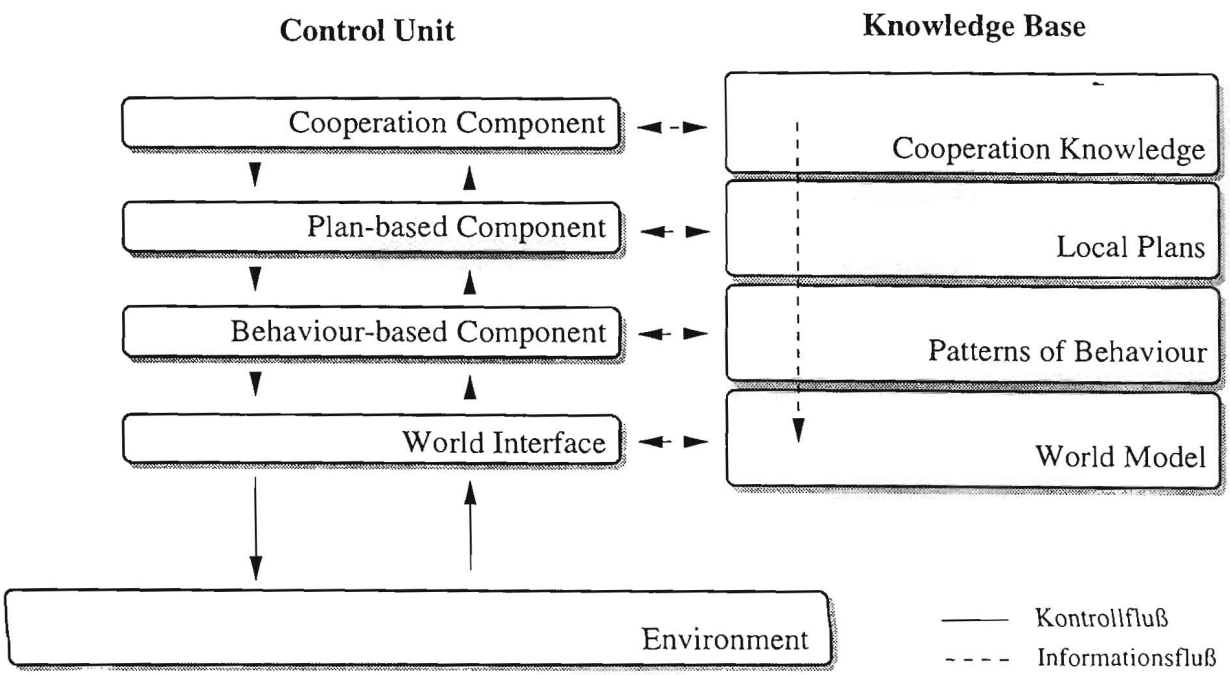

Abbildung 7.1.: InteRRaP Agenten Modell

Diese Struktur läßt sich auf die Agenten-Gemeinschaft im Speditions-Szenario übertragen und soll am Beispiel des Fahrzeug-Agenten skizziert werden.

7.1.1. Wissensbasis

World Model: Die unterste Ebene der Wissensbasis eines Agenten enthält dessen Weltbild. Dieses Wissen repräsentiert die Objekte der Welt und die Relationen, die zwischen diesen gelten. Dies entspräche im Szenario den geographischen Gegebenheiten, Wissen der eigenen Merkmale wie Ladebeschränkungen oder Standort.

Patterns of Behaviour: Die darüberliegende Ebene enthält das Wissen um primitivreaktive Verhaltensmuster und den prinzipiell ausführbaren Aktionen. Das Wissen des Fahrzeug-Agenten, zu eingehenden Aufträgen ein Angebot zu berechnen oder das Wissen, Entfernungen auf einer Straßenkarte (Guide) nachzusehen, wäre eine solche Entsprechung.

Local Plans: Eine Sammlung von Planskeletten wird in der dritten Ebene angeboten. Diese Pläne werden, aufbauend auf primitiven Verhaltensmustern oder nicht instantiierten Subplänen, rekursiv definiert. Analog dazu ist das Wissen des zu benutzenden Planverfahrens (Insertion, Sweep, ...) eine Art Planskelett, das vom Fahrzeug durch entsprechende Aktionen instantiiert wird.

Cooperation Knowledge: Das Wissen bzgl. Kooperation und Koordination wird in der obersten Stufe der Hierarchie repräsentiert. Dies entspricht z.B. der Schnittstelle der Fahrzeugagenten zu der darüberliegenden Spedition über das Contract-Net-Protocol. Ein weiterer Ansatz horizontaler Kooperation soll in Kapitel 7.2 erläutert werden.

7.1.2. Kontrolleinheit

Die Kontrolleinheit des Agenten reflektiert im wesentlichen die hierarchische Struktur der Wissensbasis.

World Interface: Die unterste Ebene entspricht der Ein-/Ausgabeschnittstelle des Agenten, d.h. Kommunikation mit anderen Agenten, Wahrnehmung der Umwelt und Aktionen. Durch Wahrnehmung gewonnene Informationen werden in der *World-Model*-Wissensbasis abgelegt (beispielsweise Staumeldungen), die wiederum Verhaltensmuster der darüberliegenden Ebenen anstoßen (Suche einen Umweg).

Behaviour-based Component BBC: Hier werden grundlegende Verhaltensmuster angestoßen, die als Art Reaktion zu verstehen sind. So z.B. das Melden eines Angebots nach dem Versuch, einen neuen Kunden in die bestehende Tour einzuplanen.

Plan-based Component PBC: Ist die Situation zu komplex für die BBC, oder verlangt die Situation keine solche schnelle Reaktion, wird die Kontrolle der planbasierten Komponente übergeben. Diese Komponente beinhaltet die Fähigkeit des Agenten, zur Plangenerierung und Entscheidungsfindung. Ein typisches Beispiel aus dem Speditionsszenario wäre das tatsächliche Einplanen eines neuen Kunden aufgrund des deklarativen Wissens aus der Wissensbasis.

Cooperation Component CC: Die Kooperationskomponente übernimmt dann die Kontrolle, wenn ein Problem nicht von einem Agenten allein gelöst wird oder wenn Hilfe von

anderen Agenten angestrebt-wird. So z.B. auch beim Simulated-Trading-Prozeß, bei dem alle Agenten versuchen, einen besseren globalen Plan aufgrund von lokalen Planverbesserungen zu erreichen.

7.2. Erweitertes Contract-Net-Protocol

Wie bereits im vorangegangenen Kapitel angedeutet, läßt sich das Speditionsszenario um einen weiteren Mechanismus zwischen der Spedition und deren Fahrzeugen erweitern, falls die Gütermenge der einzelnen Aufträge die Kapazität der Fahrzeuge übersteigt oder durch Aufteilen eines oder mehrerer Aufträge auf verschiedene Fahrzeuge eine bessere Lösung für den Gesamttourplan erreicht wird. Die Ausführungen folgen [Fischer and Kuhn, 1993].
Das reine Contract-Net-Protocol ist nicht ausreichend, um diese Art der Aufgabenteilung durchzuführen. Dazu wird es zu dem *Extended Contract Net* Protokoll *(ECN)* erweitert; die beiden Mitteilungen *grant* und *reject* werden zu *temporal grant*, *temporal reject*, *definitive grant* und *definitive reject* ausgeweitet.
Zunächst gibt der Speditionsagent an alle seine Fahrzeugagenten einen vom Broker erhaltenen Auftrag weiter. Diese berechnen nun ein Angebot für die Menge an Transportgütern, die sie fähig sind zu transportieren. Die Spedition wählt aus diesen das beste[1] Angebot aus und gibt dem entsprechenden Agenten ein *temporal grant*. Allen anderen Fahrzeugen wird ein *temporal reject* mitgeteilt. Bezieht sich das Angebot nicht auf die gesamte Auftragsmenge, wird diese vom Gesamtvolumen abgezogen und der verbleibende Auftrag wird den Fahrzeugen wieder angeboten, solange bis ein vollständiges Angebot bzgl. des letzten Auftragteils vorliegt. Die Spedition berechnet nun aufgrund der Teilangebote seiner Fahrzeuge ein Gebot für den Broker. Erst wenn von diesem ein grant erteilt wird, erhalten alle Slave-Agenten der Spedition ein *definitive grant* bzw. *definitive reject* entsprechend der vorherigen Nachricht. Erhält die Spedition den Zuschlag für den Auftrag nicht, werden alle Fahrzeuge mit einem *definitive reject* informiert.
Der Fahrzeugagent muß seine lokalen Pläne zwischenspeichern, solange er temporal grants erhält, erst wenn ihm definitive Zu- oder Absagen mitgeteilt werden, kann er diese verwerfen.

7.3. Preisverhandlungen zwischen Speditionen

Der im vorangegangenen Abschnitt gemachte Ansatz kann wiederum erweitert werden. Analog zu dem Vorgehen ließe sich ein Modell entwerfen, das beschreibt, was geschehen soll, falls der vollständige Auftrag nicht von einer Spedition bearbeitet werden kann, oder der Agent der Ansicht ist, einen Teilauftrag nur mit unverhältnismäßig hohen Kosten ausführen zu können. Hier wäre eine Kooperation auf Speditionsebene denkbar.
Die Master-Spedition übernimmt eine Art Broker-Funktion gegenüber kooperierenden Speditionen. Dazu werden diesen 'unangenehme' Teilaufträge von der übergeordneten Spedition angeboten und entsprechende Angebote eingeholt. Im Gegensatz zu der echten Broker-Speditions-Ebene wären hier Preisverhandlungen vorstellbar. Dabei spielen vor allem Aspekte der Spiel- und Verhandlungstheorie eine Rolle.

Die Master-Spedition gibt letztendlich an den Broker ein Gesamtangebot ab.

7.4. Oz-Constraint-Mechanismen

Die in dieser Arbeit vorgestellten Ergebnisse lassen sich gerade auch im Hinblick auf das Simulated-Trading weiter verbessern. Ein Beispiel für eine weitere Nachoptimierung ist das auf Seite 30 beschriebene Kantenaustauschverfahren k-opt von [Lin and Kerninghan, 1973]. Simulated-Trading, so wie es in [Malich, 1991] beschrieben wurde, führt nach jeder Kauf-/Verkaufaktion solch eine lokale Touroptimierung durch, was insgesamt zu einem besseren Tourplan führt. Natürlich wäre das k-opt-Verfahren auch bereits auf die Initiallösung anwendbar.
Eine interessantere Alternative stellen die von Oz selbst angebotenen Constraint-Mechanismen dar: *Finite Domains* und *Solve Combinator*. Diese sollen hier nur kurz vorgestellt werden, da es zu diesen Themen ausgezeichnete Dokumentationen gibt.

7.4.1. Finite Domains

Finite-Domain-Variablen sind Variablen, die einen Integer-Wert zwischen 0 und einer oberen Schranke symbolisieren [Müller et al., 1994b], d.h. es wird der Wertebereich (Domain) angegeben, aus welchem die Wertzuweisung der Variable stammen soll. Diese Zuweisung erfolgt über sogenannte *finite domain constraints*.

Beispiel 7.1 Folgender Quellcode weist der Variable `V` einen Wert zwischen 5 und 10 zu:

```
local V in
   V :: 5#10
   { Browse V }
end
```

Im Browser, einem Ausgabetool von Oz erscheint folgende Ausgabe:

```
_{ 5..10 }
```

Das FD-Modul stellt eine Reihe dieser Constraint-Mechanismen zu Verfügung, wie z.B. `X :< 10`, `Y*2 =: X`, ..., mit denen sich die Wertebereiche eingrenzen lassen (s. auch [Henz et al., 1994]).

7.4.2. Search Combinator

Das *Search*-Modul bietet einen einfach zu handhabenden Mechanismus an, in einem aufgespannten Constraint-Netz Lösungen zu suchen, d.h. letztendlich Graphensuche druchzuführen [Schulte et al., 1994]. Das Suchproblem (*query*) wird dem Modul in Form einer Prozedur übergeben. Zu dessen Lösung stehen nach [Henz et al., 1994] mehrere Ansätze zur Verfügung:

[1] Auch hier stellt sich die Frage nach der Güte eines Angebots, analog zu Kapitel 5.1.2. Zusätzlich muß jedoch das entsprechende Angebot in Abhängigkeit der beförderten Menge bewertet werden, wie z.B. Angebot/Menge.

- one solution,
- all solutions,
- best solution.

Wie sich das Finite-Domain- und Search-Modul zur Optimierung lokaler Touren nutzen läßt, zeigt folgendes Kapitel.

7.4.3. Anwendung

Fleet-Scheduling-Probleme, genau wie andere Scheduling-Probleme sind mit Constraints geradezu gespickt (maximale Ladekapazität des Fahrzeugs, Zeitfenster beim Kunden, ...). In [Müller et al., 1994a] und [Müller et al., 1994b] ist ein Beispiel für einen Fertigungsleitstand zu finden, bei dem versucht wird, eine optimale Auslastung von Maschinen mit anstehenden Aufträgen zu finden.
Analog zu diesem Beispiel läßt sich mit Hilfe des FD-Moduls das Problem spezifizieren (berechne bzgl. der in der Tour befindlichen Kunden eine optimale Rundreise) und dem Search-Modul übergeben. Dieses berechnet dann z.B. mit einem Branch & Bound Ansatz die beste Lösung. Jedes einzelne Fahrzeug kann lokal als Planungseinheit angesehen werden. Ziel der Planung ist die optimale Tour bezüglich den zu beliefernden Kunden.
Eine optimale Lösung für den gesamten Tourplan mit Hilfe der von Oz angebotenen Constraint-Mechanismen wäre zu zeitintensiv. Die Dezentralisierung der Planung bringt einen Effizienzzuwachs mit sich und führt zu einer Kooperation von *Constraint-Solvern* in Form eines Multiagentensystems und ist somit Beitrag zur Lösung von Constraint-Problemen.

Im folgenden soll die Vorgehensweise skizziert werden, wie man bei einer bekannten Auftragsmenge einer Tour bzgl. des VRPTW eine optimale Reihenfolge der Kunden berechnen kann. Der Einfachheit halber sei davon ausgegangen, daß die Tour ursprünglich nur eine Rundreise beschreibt, d.h. zwischenzeitlich nicht zum Depot zurückkehrt. Da alle Kunden bereits in der ursprünglichen Tour vom Depot aus beliefert werden, muß die Kapazitätsbeschränkung nicht beachtet werden.
Seien also die Auftragsmenge `Orders` einer bestehenden Tour in Form einer Liste, sowie die Kapazitäts- und Zeitbeschränkungen bzgl. des ausführenden Fahrzeuges bekannt. Zuerst werden die Aufträge in entsprechende Planschritte konvertiert:

```
fun{ AssignSteps Orders }
   { Map Orders
     fun{ $ O }
        step(cust:O.e
             dda:O.dda_e
             demand:_
             dur:O.dur_e
             est:O.est_e
             id:O.id
             next:_
             cst:_)
     end }
```

```
end
Plan={ AssignSteps Orders }
```

Die features demand, next und cst sind Ziel der Wertzuweisung. Dabei soll cst implizit eine Ordnung über den Planschritten angeben, die beiden anderen werden im Anschluß aktualisiert. Zunächst wird *cst* durch das beim Kunden definierte Zeitfenster eingegrenzt:

```
proc{ DeadLine Pl }
   { ForAll Pl
     proc{ $ P }
        P.cst::{ Max P.est { Duration '0' P.cust } }
        #{ Min DeadLine-{ Duration P.cust '0' }-P.dur P.dda }
     end }
end
```

Duration soll eine zuvor definierte Funktion sein, die die Zeitdauer angibt, um die Strecke zwischen zwei Punkten zurückzulegen. DeadLine ist der Zeitpunkt, zu dem das Fahrzeug wieder am Depot angelangt sein soll. Das Zeitfenster kann dabei sogar noch weiter eingeschränkt werden, die est um die Dauer des Weges vom Depot zum entsprechenden Kunden, die dda um die Zeit vom Kunden wieder zum Depot. Diese beiden Restriktionen gelten für alle Aufträge.
Da die einzelnen Aufträge zeitlich sequentiell ablaufen, kann gesagt werden, daß entweder Auftrag A vor Auftrag B ausgeführt wird oder umgekehrt. Dabei können wiederum die Schranken für cst begrenzt werden, da ja zwischen je zwei Kunden eine zeitliche Entfernung liegt:

```
proc{ Sequence Pl}
   { ForAllTail Pl
     proc{ $ P|Ps }
        { ForAll Ps
          proc{ $ P2 } Dur={ Duration P.cust P2.cust } in
             or P.cst+Dur+P.dur =<: P2.cst
             [] P2.cst+Dur+P2.dur =<: P.cst
             ro
          end }
     end }
end
```

Die Suchprozedur. die nun das Problem beschreibt, ist die folgende:

```
proc{ Query Plan }
   Plan={ AssignSteps Orders }
   { DeadLine Plan }
   { Sequence Plan }
   { FD.enums { Map Plan fun{ $ P } P.cst end } }
end
```

Über enums wird dem Suchmodul mitgeteilt, daß die Enumeration über cst laufen soll. Die Vorgehensweise ist die. daß mit dem ersten cst der Liste der kleinstmögliche Wert

zugeordnet wird und über die Constraints versucht wird, die restlichen zu instantiieren. Die Suche selbst wird gestartet mit:

```
{ Search query(Query) }
{ Search next }
```

{ Search next } fordert den Solver auf, eine Lösung im Browser auszugeben. Dies ist in aller Regel die zuerst gefundene.
Ist man an der besten Lösung interessiert, kann auch dies dem Solve-Combinator mitgeteilt werden. Dazu muß diesem ein zweistelliges Prädikat {P P1 P2} übergeben werden, so daß eine Lösung eines Problems P2 dann besser ist als P1, falls das Prädikat gilt. Dies ließe sich durch folgenden Quellcode verwirklichen:

```
proc{ Order P1 P2 }
   Cost in Cost >=: 0
   { Analize P1 }>: Cost
   { Analize P2 }=<: Cost
end
{ Solve.best.bab Query Order }
```

Analize sei dabei eine Funktion, die die Lösung (eine neue Tour) bzgl. einer Kostenfunktion bewertet. Solve.best.bab teilt dem Modul mit, daß die beste Lösung mittels einem Branch & Bound Ansatz gesucht werden soll. Die Prozedur liefert solved(X), mit {X} als Lösung, falls eine solche existiert, sonst failed.
Als Ergebnis wird jedem Planschritt wiederum eine eindeutige cst zugewiesen, die implizit die Ordnung angibt. Die einzelnen Schritte müssen nun noch bzgl. demand und next aktualisiert werden.

Literaturverzeichnis

[Bachem et al., 1992] Bachem, A., Hochstättler, W., and Malich, M. (1992). A New Parallel Approach For Solving Vehicle Routing Problem. Technical Report 92.125, Mathematisches Institut, Universität zu Köln.

[Bachem et al., 1993] Bachem, A., Hochstättler, W., and Malich, M. (1993). The Simulated Trading Heuristic For Solving Vehicle Routing Problem. Technical Report 93.139, Mathematisches Institut, Universität zu Köln.

[Christofides et al., 1979] Christofides, N., Mingozzi, A., and Toth, P. (1979). The Vehicle Routing Problem. In *Combinatorial Optimizations*. John Wiley & Sons, New York.

[Christofides et al., 1981] Christofides, N., Mingozzi, A., and Toth, P. (1981). Exact Algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. In *Mathematical Programming*, volume 20, pages 255–282.

[Clarke and Wright, 1964] Clarke, G. and Wright, J. (1964). Scheduling of vehicles from a central depot to a number of delivery points. In *Operations Research*, volume 12, pages 568–581.

[Dantzig and Ramser, 1959] Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. In *Management science*, volume 6, pages 80–91.

[Davis and Smith, 1983] Davis, R. and Smith, R. G. (1983). Negotiation as a metaphor for distributed problem solving. In *Artificial Intelligence*, volume 20(1).

[Desrochers et al., 1992] Desrochers, M., Desrosiers, J., and Solomon, M. (1992). A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. In *Operations Research*, volume 40(2), pages 343–354.

[Desrochers et al., 1988] Desrochers, M., Lenstra, J., Savelsbergh, M., and Soumis, F. (1988). Vehicle Routing with Time Windows: Optimization and Approximation. In Golden, B. L. and Assad, A. A., editors, *Vehicle Routing: Methods and Studies*, volume 16, pages 65–85. Elsevier Science Publishers B.V., North-Holland.

[Desrosiers et al., 1986] Desrosiers, J., Dumas, Y., and Soumis, F. (1986). The Multiple Vehicle Many to Many Ruting and Scheduling Problem with Time Windows. Cahiers du GERAD G-84-13, Ecole des Hautes Etudes Commerciales de Montréal.

[Dumas, 1985] Dumas, Y. (1985). Confection d'itinéraires de véhicules en vue du transport de plusieurs origines a plusieures destinations. Publication 434, Centre de recherche sur les transports, Université de Montréal.

[Durfee, 1991] Durfee, E. H. (1991). The Distributed Artificial Intelligence Melting Pot. In *IEEE Transactions on Systems, Man and Cybernetics*, volume 21(6).

[Erman et al., 1980] Erman, L., Hayes-Rith, F., Lesser, V., and Reddy, R. (1980). The Hearsay-II Speech Understanding System: Integrating Knowledge to resolve Uncertainty. In *ACM Computing Surveys*, volume 12(2).

[Fischer, 1993] Fischer, K. (1993). The Rule-based Multi-Agent System MAGSY. In *Proceedings of the CKBS'92 Workshop*. Keele University.

[Fischer and Kuhn, 1993] Fischer, K. and Kuhn, N. (1993). A DAI Approach to Modeling the Transportation Domain. Research Report 93-25, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Saarbrücken.

[Gillet and Miller, 1974] Gillet, B. and Miller, L. (1974). A Heuristic Algorithm for the Vehicle Dispatching Problem. In *Operations Research*, volume 22, pages 340–349.

[Golden and Assad, 1988] Golden, B. L. and Assad, A. A., editors (1988). *Vehicle Routing: Methods and Studies*, volume 16. Elsevier Science Publishers B.V., North-Holland.

[Henz et al., 1994] Henz, M., Müller, M., Schulte, C., and Würtz, J. (1994). The Oz Standard Modules. DFKI Oz documentation series, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Saarbrücken.

[Herrtwich, 1990] Herrtwich, R. G. (1990). An Introduction to Real-Time Scheduling. Technical Report 90-035, International Computer Science Institute, 1947 CenterStreet, Suite 600 Berkeley, California 947044-1105.

[Hopcroft and Ullman, 1993] Hopcroft, J. and Ullman, J. (1993). *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. Addison-Wesley.

[Köhler, 1925] Köhler, W. (1925). *The mentality of apes*. Routeledge&Kegan Paul, London.

[Köhler, 1947] Köhler, W. (1947). *Gestalt Psychology*. Liveright, NewYork.

[Kolen et al., 1987] Kolen, A., Rinnooy, A., and Trienekens, H. (1987). Vehicle Routing with Time Windows. In *Operations Research*, volume 35/2, pages 266–273.

[Lewis and Papademitriou, 1981] Lewis, H. R. and Papademitriou, C. H. (1981). *Elements of the Theory of Computation*. Prentice-Hall.

[Lin, 1965] Lin, S. (1965). Computer Solutions to the Traveling Salesman Problem. In *Bell System Tech. J. 44*, pages 2245–2269.

[Lin and Kerninghan, 1973] Lin, S. and Kerninghan, B. W. (1973). An Effective Heuristic Algorithm for the Traveling Salesman Problem. In *Operations Research*, volume 21/1, pages 498–516.

[Malich, 1991] Malich, M. (1991). Simulated-Trading als Parallelisierungsansatz zur Lösung von Vehicle Routing Problemen. Diplomarbeit, Universität zu Köln.

[Mille, 1991] Mille, Y. (1991). Just-In-Time 'entmystifiziert'. In *Tagungsband zum 7. Saarbrücker Logistikforum: Produktions- and Logistikstrategien für Europa.*

[Minsky, 1985] Minsky, M. (1985). *The society of mind.* Simon & Schuster, Inc., NewYork.

[Mole and Jameson, 1976] Mole, R. and Jameson, S. (1976). A Sequential Route-Building Algorithm Employing a Generalized Savings Criterion. In *Operations Research Quarterly*, volume 27, pages 503–511.

[Müller, 1993] Müller, H. J., editor (1993). *Verteilte Künstliche Intelligenz — Methoden und Anwendungen (Distributed Artificial Intelligence — Methods and Applications).* BI-Verlag.

[Müller and Pischel, 1993] Müller, J. P. and Pischel, M. (1993). The Agent Architecture InteRRaP: Concept and Application. Research Report 93-26, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Saarbrücken.

[Müller et al., 1994a] Müller, M., Müller, T., Schulte, C., Treinen, R., and Würtz, J. (1994a). DFKI Oz Demos. DFKI Oz documentation series, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Saarbrücken.

[Müller et al., 1995] Müller, M., Müller, T., and Van Roy, P. (1995). Multi-Paradigm Programming in Oz. In Smith, D., Ridoux, O., and Van Roy, P., editors, *Visions for the Future of Logic Programming: Laying the Foundations for a Modern Successor of Prolog*, Portland, Oregon. A Workshop in Association with ILPS'95.

[Müller et al., 1994b] Müller, T., Popow, K., Schulte, C., and Würtz, J. (1994b). Constraint Programming in Oz. DFKI Oz documentation series, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Saarbrücken.

[Papademitriou, 1994] Papademitriou, C. H. (1994). *Computational Complexity.* Addison-Wesley.

[Peterson and Silberschatz, 1994] Peterson, J. L. and Silberschatz, A. G. P. (1994). *Operating System Concepts.* Addison-Wesley.

[Rich and Knight, 1991] Rich, E. and Knight, K. (1991). *Artificial Intelligence.* McGraw-Hill.

[Savelsberg, 1990] Savelsberg, M. (1990). An efficient implementation of local search algorithms for constrained routing problems. In *European Journal of Operational Research*, volume 47, pages 75–85.

[Schier, 1994] Schier, D. (1994). Das neue Oz-Speditions-Szenario mit Zeitbeschränkungen. Fortgeschrittenenpraktikum, Universität zu Saarbrücken.

[Schulte et al., 1994] Schulte, C., Smolka, G., and Würtz, J. (1994). Encapsulated Search and Constraint Programming in Oz. In *Second Workshop on Principles and Practice of Constraint Programming*, pages 134–150. Springer-Verlag.

[Schupeta, 1992] Schupeta, A. (1992). Main Topics of DAI: A Review. Research Report 92-06, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Saarbrücken.

[Smolka, 1995] Smolka, G. (1995). An Oz Primer. DFKI Oz documentation series, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Saarbrücken.

[Solomon, 1983] Solomon, M. (1983). Vehicle Routing and Scheduling Problems with Time Window Constraints: Models and Algorithms. Report 83-02-01, Department of Decision Science, The Warton School, University of Pennsylvania, Philadelphia.

[Solomon, 1987] Solomon, M. (1987). Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. In *Operations Research*, volume 35/1, pages 254–265.

[Stankovic and Ramamrithan, 1988] Stankovic, J. A. and Ramamrithan, K., editors (1988). *Hard Real Time Systems.* IEEE Computer Society.

[Wertheimer, 1958] Wertheimer, M. (1958). *Productive Thinking.* Harper& Row, NewYork.

[Zlotkin and Rosenschein, 1993] Zlotkin, G. and Rosenschein, J. S. (1993). A Domain for Task Oriented Negotiation. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 416–422.

A. Ergebnistafeln

Problem	# Kunden	# Fahrzeuge	Distanz	Dauer	Wartezeit
r101	100	25	2327,7	4548,6	1220,9
r102	100	22	2114,1	4124,5	1010,4
r103	100	18	1776,3	3542,2	765,9
r104	100	15	1873,8	3223,9	350,1
r105	100	21	2392,9	3834,2	441,3
r106	100	20	2069,0	3616,5	547,5
r107	100	16	1950,0	3339,7	389,7
r108	100	14	1952,7	3111,7	159,0
r109	100	18	2126,1	3361,8	235,7
r110	100	19	2424,1	3680,5	256,4
r111	100	15	1947,1	3180,6	233,5
r112	100	15	2029,7	3152,0	122,3
r1	100	18,1667	2081,96	3559,68	477,725

Tabelle A.1.: Parallel-Insertion; unsortiert

Problem	# Kunden	# Fahrzeuge	Distanz	Dauer	Wartezeit
r101	100	21	1800,3	3886,6	1086,3
r102	100	20	1638,7	3669,1	1030,4
r103	100	14	1354,2	2808,8	454,6
r104	100	12	1556,5	2574,3	17,8
r105	100	18	1917,2	3311,3	394,1
r106	100	15	1625,2	2880,9	255,7
r107	100	13	1793,7	2842,6	48,9
r108	100	13	1837,5	2853,5	16,0
r109	100	17	2073,2	3233,1	159,9
r110	100	15	1939,0	3053,4	114,4
r111	100	14	1770,0	2901,4	131,4
r112	100	15	2130,3	3198,8	68,5
r1	100	15,5833	1786,32	3101,15	314,833

Tabelle A.2.: Parallel-Insertion; sortiert nach farthestCustomer, hardEstFirst

Problem	# Kunden	# Fahrzeuge	Distanz	Dauer	Wartezeit
r101	100	22	2087,6	3983,6	896,0
r102	100	20	1996,0	3823,6	827,6
r103	100	16	1760,5	3273,8	513,3
r104	100	12	1354,5	2587,0	232,5
r105	100	17	1743,1	3064,5	321,4
r106	100	14	1579,8	2808,0	228,2
r107	100	12	1420,7	2563,4	142,7
r108	100	11	1221,5	2300,1	78,6
r109	100	14	1537,0	2672,5	135,5
r110	100	13	1475,2	2645,7	170,5
r111	100	13	1465,5	2656,9	191,4
r112	100	11	1227,1	2368,0	140,9
r1	100	14,5833	1572,38	2895,59	323,217

Tabelle A.3.: Sequentielles Insertion; unsortiert

Problem	# Kunden	# Fahrzeuge	Distanz	Dauer	Wartezeit
r101	100	21	1859,1	3744,1	885,0
r102	100	19	1736,4	3553,4	817,0
r103	100	14	1481,6	2805,2	323,6
r104	100	10	1137,8	2207,7	69,9
r105	100	15	1604,2	2853,8	249,6
r106	100	14	1448,8	2726,5	277,7
r107	100	12	1349,3	2507,5	158,2
r108	100	10	1140,3	2226,5	86,2
r109	100	14	1465,9	2582,1	116,2
r110	100	12	1336,5	2446,6	110,1
r111	100	12	1231,4	2424,9	193,5
r112	100	11	1223,2	2299,1	75,9
r1	100	13,6	1417,9	2698,1	280,2

Tabelle A.4.: Sequentielles Insertion; sortiert nach farthestCustomer, hardEstFirst

Problem	# Kunden	# Fahrzeuge	Distanz	Dauer	Wartezeit
r101	100	19	1711,6	3520,8	809,2
r102	100	19	1520,9	3615,9	1095,0
r103	100	14	1323,1	2735,8	412,7
r104	100	10	1075,5	2197,5	122,0
r105	100	15	1454,0	2812,5	358,5
r106	100	13	1334,8	2649,8	315,0
r107	100	12	1181,8	2485,8	304,0
r108	100	10	999,2	2165,4	166,2
r109	100	13	1296,1	2508,3	212,2
r110	100	12	1169,1	2331,3	162,2
r111	100	11	1160,4	2322,0	161,6
r112	100	11	1050,5	2172,6	122,1
r1	100	13,25	1273,1	2626,5	353,4

Tabelle A.5.: Sequentielles Insertion; sortiert nach farthestCustomer, hardEstFirst; nachoptimiert

Problem	# Kunden	# Fahrzeuge	Distanz	Dauer	Wartezeit
r101	25	8	696,2	1427,7	485,5
r102	25	7	605,1	1288,3	436,7
r103	25	6	554,6	1114,5	312,9
r104	25	5	502,0	955,6	206,1
r105	25	5	562,2	955,0	145,3
r106	25	5	538,9	979,8	193,4
r107	25	4	550,4	848,1	49,7
r108	25	4	509,0	819,7	62,7
r109	25	4	515,4	802,0	38,6
r110	25	5	507,3	854,6	99,8
r111	25	5	500,5	869,5	121,5
r112	25	4	540,6	840,1	51,5
r1	25	5,1	548,5	979,6	183,6

Tabelle A.6.: Initiallösung für Datensatz R1 mit 25 Kunden; sortiert nach farthestCustomer, hardEstFirst

Problem	# Kunden	# Fahrzeuge	Distanz	Dauer	Wartezeit
r101	25	8	618,1	1365,8	501,7
r102	25	7	566,1	1280,5	467,9
r103	25	5	485,3	962,5	227,2
r104	25	4	418,0	771,2	103,2
r105	25	5	559,6	955,0	147,9
r106	25	5	472,2	956,8	237,1
r107	25	4	445,5	789,8	96,3
r108	25	4	434,1	788,0	105,9
r109	25	4	460,5	763,3	54,8
r110	25	5	453,3	874,4	173,6
r111	25	5	431,1	845,0	163,9
r112	25	4	440,2	748,3	58,1
r1	25	5,0	482,0	925,1	194,8

Tabelle A.7.: R1 mit 25 Kunden; nachoptimiert

Problem	# Kunden	# Fahrzeuge	Distanz
r101	25	8	617,1
r102	25	7	546,4
r103	25	5	454,6
r104	25	4	416,9
r105	25	6	530,5
r106	25	5	467,4
r107	25	4	424,3
r108	25	4	397,2
r109	25	5	441,3
r110	25	4	429,5
r111	25	4	428,8
r112	25	4	393,0
r1	25	5,0	462,2

Tabelle A.8.: Optimale Ergebnisse nach [Desrochers et al., 1992]

Problem	# Fahrzeuge	Dauer
r101	19	2398,0
r102	17	1991,7
r103	14	1570,1
r104	10	1061,9
r105	14	1596,0
r106	12	1356,9
r107	11	1145,3
r108	10	989,1
r109	12	1262,1
r110	11	1162,9
r111	11	1164,4
r112	10	1005,2

Tabelle A.9.: Ergebnisse nach [Bachem et al., 1993]

B. Zeichen und Abkürzungen

a_i	(Auftrag) repräsentiert den Auftrag bzgl. Kunde i.	16,19
cst	(current start time) ist der Zeitpunkt, zu dem tatsächlich eine Aktion bei einem Kunden begonnen wird.	18
d_{ij}	(Distanz) ist die räumliche Entfernung zwischen Kunde i und j.	16
dda	(due date) ist der letztmögliche Zeitpunkt, zu dem bei einem Kunden mit einer Aktion begonnen worden sein muß. *dda_s* steht beim PDPTW für den Zeitpunkt beim Beladekunden, *dda_e* beim Entladekunden.	18,20
$DeadLine_i$	(DeadLine) repräsentiert den Zeitpunkt, zu dem das die Tour T_i ausführende Fahrzeug wieder im Depot sein muß.	17
dur	(duration) ist die Zeitdauer, die eine Aktion bei einem Kunden benötigt. *dur_s* steht beim PDPTW für die Zeitdauer beim Beladekunden, *dur_e* beim Entladekunden.	18,20
est	(earliest start time) ist der Zeitpunkt, zu dem frühestens mit einer Aktion bei einem Kunden begonnen werden kann. *est_s* steht beim PDPTW für den Zeitpunkt beim Beladekunden, *est_e* beim Entladekunden.	18,20
G_R	(Routing-Graph) ist der dem VRP zugrundeliegende Graph, der Angaben bzgl. der Position und Entfernung zwischen Kunden enthält.	16
grant	definiert die Zusage an ein Fahrzeug bzw. Spedition,den Zuschlag für ein zu beförderndes Gut erhalten zu haben.	33
q_i	(Auftragsmenge) ist die Menge an Transportgütern, die ein Kunde i fordert.	16

Q_i	(Kapazität) ist die maximal mögliche Ladekapazität des Fahrzeugs, das zur Ausführung von T_i bestimmt ist.	17
reject	repräsentiert die Absage an ein Fahrzeug bzw. Fuhrunternehmen, bzgl. eines Auftrages.	33
t_{ij}	(Zeitraum) ist die zeitliche Entfernung zwischen zwei Kunden i und j.	16
T_i	(Tour) ist ein von einem Fahrzeug geplante und auszuführende Folge von Kunden.	17,19
$\mathcal{T}$	(Tourplan) ist eine Menge von Touren.	18
w_{ij}	(Kante) verbindet Knoten i mit j	16

Abbildungsverzeichnis

Tabellenverzeichnis

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

-Bibliothek, Information
und Dokumentation (BID)-
PF 2080
67608 Kaiserslautern
FRG

Telefon (0631) 205-3506
Telefax (0631) 205-3210
e-mail
dfkibib@dfki.uni-kl.de
WWW
http://www.dfki.uni-sb.de/dfkibib

Veröffentlichungen des DFKI

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder (so sie als per ftp erhaeltlich angemerkt sind) per anonymous ftp von ftp.dfki.uni-kl.de (131.246.241.100) im Verzeichnis pub/Publications bezogen werden. Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or (if they are marked as obtainable by ftp) by anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) in the directory pub/Publications.
The reports are distributed free of charge except where otherwise noted.

DFKI Research Reports

1996

RR-96-06
Claus Sengler
Case Studies of Non-Freely Generated Data Types
200 pages

RR-96-05
Stephan Busemann
Best-First Surface Realization
11 pages

RR-96-04
Christoph G. Jung, Klaus Fischer, Alastair Burt
Multi-Agent Planning
Using an *Abductive*
EVENT CALCULUS
114 pages

RR-96-03
Günter Neumann
Interleaving
Natural Language Parsing and Generation
Through Uniform Processing
51 pages

RR-96-02
E.André, J. Müller , T.Rist:
PPP-Persona: Ein objektorientierter Multimedia-Präsentationsagent
14 Seiten

RR-96-01
Claus Sengler
Induction on Non-Freely Generated Data Types
188 pages

1995

RR-95-20
Hans-Ulrich Krieger
Typed Feature Structures, Definite Equivalences, Greatest Model Semantics, and Nonmonotonicity
27 pages

RR-95-19
Abdel Kader Diagne, Walter Kasper, Hans-Ulrich Krieger
Distributed Parsing With HPSG Grammar
20 pages

RR-95-18
Hans-Ulrich Krieger, Ulrich Schäfer
Efficient Parameterizable Type Expansion for Typed Feature Formalisms
19 pages

RR-95-17
Hans-Ulrich Krieger
Classification and Representation of Types in TDL
17 pages

RR-95-16
Martin Müller, Tobias Van Roy
Title not set
0 pages

Note: The author(s) were unable to deliver this document for printing before the end of the year. It will be printed next year.

RR-95-15
Joachim Niehren, Tobias Van Roy
Title not set
0 pages

Note: The author(s) were unable to deliver this document for printing before the end of the year. It will be printed next year.

RR-95-14
Joachim Niehren
Functional Computation as Concurrent Computation
50 pages

RR-95-13
Werner Stephan, Susanne Biundo
Deduction-based Refinement Planning
14 pages

RR-95-12
Walter Hower, Winfried H. Graf
Research in Constraint-Based Layout, Visualization, CAD, and Related Topics: A Bibliographical Survey
33 pages

RR-95-11
Anne Kilger, Wolgang Finkler
Incremental Generation for Real-Time Applications
47 pages

RR-95-10
Gert Smolka
The Oz Programming Model
23 pages

RR-95-09
M. Buchheit, F. M. Donini, W. Nutt, A. Schaerf
A Refined Architecture for Terminological Systems: Terminology = Schema + Views
71 pages

RR-95-08
Michael Mehl, Ralf Scheidhauer, Christian Schulte
An Abstract Machine for Oz
23 pages

RR-95-07
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt
The Complexity of Concept Languages
57 pages

RR-95-06
Bernd Kiefer, Thomas Fettig
FEGRAMED
An interactive Graphics Editor for Feature Structures
37 pages

RR-95-05
Rolf Backofen, James Rogers, K. Vijay-Shanker
A First-Order Axiomatization of the Theory of Finite Trees
35 pages

RR-95-04
M. Buchheit, H.-J. Bürckert, B. Hollunder, A. Laux, W. Nutt,
M. Wójcik
Task Acquisition with a Description Logic Reasoner
17 pages

RR-95-03
Stephan Baumann, Michael Malburg, Hans-Guenther Hein, Rainer Hoch,
Thomas Kieninger, Norbert Kuhn
Document Analysis at DFKI
Part 2: Information Extraction
40 pages

RR-95-02
Majdi Ben Hadj Ali, Frank Fein, Frank Hoenes, Thorsten Jaeger,
Achim Weigel
Document Analysis at DFKI
Part 1: Image Analysis and Text Recognition
69 pages

RR-95-01
Klaus Fischer, Jörg P. Müller, Markus Pischel
Cooperative Transportation Scheduling
an application Domain for DAI
31 pages

1994

RR-94-39
Hans-Ulrich Krieger
Typed Feature Formalisms as a Common Basis for Linguistic Specification.
21 pages

RR-94-38
Hans Uszkoreit, Rolf Backofen, Stephan Busemann, Abdel Kader Diagne,
Elizabeth A. Hinkelman. Walter Kasper, Bernd Kiefer, Hans-Ulrich Krieger,
Klaus Netter, Günter Neumann, Stephan Oepen, Stephen P. Spackman.
DISCO–An HPSG-based NLP System and its Application for Appointment Scheduling.
13 pages

RR-94-37
Hans-Ulrich Krieger, Ulrich Schäfer
TDL - A Type Description Language for HPSG, Part 1: Overview.
54 pages

RR-94-36
Manfred Meyer
Issues in Concurrent Knowledge Engineering. Knowledge Base and Knowledge Share Evolution.
17 pages

RR-94-35
Rolf Backofen
A Complete Axiomatization of a Theory with Feature and Arity Constraints
49 pages

RR-94-34
Stephan Busemann, Stephan Oepen, Elizabeth A. Hinkelman,
Günter Neumann, Hans Uszkoreit
COSMA – Multi-Participant NL Interaction for Appointment Scheduling
80 pages

RR-94-33
Franz Baader, Armin Laux
Terminological Logics with Modal Operators
29 pages

RR-94-31
Otto Kühn, Volker Becker, Georg Lohse, Philipp Neumann
Integrated Knowledge Utilization and Evolution for the Conservation of Corporate Know-How
17 pages

RR-94-23
Gert Smolka
The Definition of Kernel Oz
53 pages

RR-94-20
Christian Schulte, Gert Smolka, Jörg Würtz
Encapsulated Search and Constraint Programming in Oz
21 pages

RR-94-19
Rainer Hoch
Using IR Techniques for Text Classification in Document Analysis
16 pages

RR-94-18
Rolf Backofen, Ralf Treinen
How to Win a Game with Features
18 pages

RR-94-17
Georg Struth
Philosophical Logics—A Survey and a Bibliography
58 pages

RR-94-16
Gert Smolka
A Foundation for Higher-order Concurrent Constraint Programming
26 pages

RR-94-15
Winfried H. Graf, Stefan Neurohr
Using Graphical Style and Visibility Constraints for a Meaningful Layout in Visual Programming Interfaces
20 pages

RR-94-14
Harold Boley, Ulrich Buhrmann, Christof Kremer
Towards a Sharable Knowledge Base on Recyclable Plastics
14 pages

RR-94-13
Jana Koehler
Planning from Second Principles—A Logic-based Approach
49 pages

RR-94-12
Hubert Comon, Ralf Treinen
Ordering Constraints on Trees
34 pages

RR-94-11
Knut Hinkelmann
A Consequence Finding Approach for Feature Recognition in CAPP
18 pages

RR-94-10
Knut Hinkelmann, Helge Hintze
Computing Cost Estimates for Proof Strategies
22 pages

RR-94-08
Otto Kühn, Björn Höfling
Conserving Corporate Knowledge for Crankshaft Design
17 pages

RR-94-07
Harold Boley
Finite Domains and Exclusions as First-Class Citizens
25 pages

RR-94-06
Dietmar Dengler
An Adaptive Deductive Planning System
17 pages

RR-94-05
Franz Schmalhofer, J. Stuart Aitken, Lyle E. Bourne jr.
Beyond the Knowledge Level: Descriptions of Rational Behavior for Sharing and Reuse
81 pages

RR-94-03
Gert Smolka
A Calculus for Higher-Order Concurrent Constraint Programming with Deep Guards
34 pages

RR-94-02
Elisabeth André, Thomas Rist
Von Textgeneratoren zu Intellimedia-Präsentationssystemen
22 Seiten

RR-94-01
Elisabeth André, Thomas Rist
Multimedia Presentations: The Support of Passive and Active Viewing
15 pages

DFKI Technical Memos

1996

TM-96-02
Harold Boley
Knowledge Bases in the World Wide Web:
A Challenge for Logic Programming
8 pages

TM-96-01
Gerd Kamp, Holger Wache
CTL — a description Logic with expressive concrete domains
19 pages

1995

TM-95-04
Klaus Schmid
Creative Problem Solving
and
Automated Discovery
— An Analysis of Psychological and AI Research —
152 pages

TM-95-03
Andreas Abecker, Harold Boley, Knut Hinkelmann, Holger Wache,
Franz Schmalhofer
An Environment for Exploring and Validating Declarative Knowledge
11 pages

TM-95-02
Michael Sintek
FLIP: Functional-plus-Logic Programming
on an Integrated Platform
106 pages

TM-95-01
Martin Buchheit, Rüdiger Klein, Werner Nutt
Constructive Problem Solving: A Model Construction Approach towards Configuration
34 pages

1994

TM-94-05
Klaus Fischer, Jörg P. Müller, Markus Pischel
Unifying Control in a Layered Agent Architecture
27 pages

TM-94-04
Cornelia Fischer
PAntUDE - An Anti-Unification Algorithm for Expressing Refined Generalizations
22 pages

TM-94-03
Victoria Hall
Uncertainty-Valued Horn Clauses
31 pages

TM-94-02
Rainer Bleisinger, Berthold Kröll
Representation of Non-Convex Time Intervals and Propagation of Non-Convex Relations
11 pages

TM-94-01
Rainer Bleisinger, Klaus-Peter Gores
Text Skimming as a Part in Paper Document Understanding
14 pages

DFKI Documents

1996

D-96-07
Technical Staff
DFKI Jahresbericht 1995
55 Seiten

Note: This document is no longer available in printed form.

D-96-05
Martin Schaaf
Ein Framework zur Erstellung verteilter Anwendungen
94 pages

D-96-04
Franz Baader, Hans-Jürgen Bürckert, Andreas Günter, Werner Nutt (Hrsg.)
Proceedings of the Workshop on Knowledge Representation and Configuration WRKP'96
83 pages

D-96-03
Winfried Tautges
Der DESIGN-ANALYZER - Decision Support im Designprozess
75 Seiten

1995

D-95-12
F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)
Working Notes of the KI'95 Workshop:
KRDB-95 - Reasoning about Structured Objects: Knowledge Representation Meets Databases
61 pages

D-95-11
Stephan Busemann, Iris Merget
Eine Untersuchung kommerzieller Terminverwaltungssoftware im Hinblick auf die Kopplung mit natürlichsprachlichen Systemen
32 Seiten

D-95-10
Volker Ehresmann
Integration ressourcen-orientierter Techniken in das wissensbasierte Konfigurierungssystem TOOCON
108 Seiten

D-95-09
Antonio Krüger
PROXIMA: Ein System zur Generierung graphischer Abstraktionen
120 Seiten

D-95-08
Technical Staff
DFKI Jahresbericht 1994
63 Seiten

Note: This document is no longer available in printed form.

D-95-07
Ottmar Lutzy
Morphic - Plus
Ein morphologisches Analyseprogramm für die deutsche Flexionsmorphologie und Komposita-Analyse
74 pages

D-95-06
Markus Steffens, Ansgar Bernardi
Integriertes Produktmodell für Behälter aus Faserverbundwerkstoffen
48 Seiten

D-95-05
Georg Schneider
Eine Werkbank zur Erzeugung von 3D-Illustrationen
157 Seiten

D-95-04
Victoria Hall
Integration von Sorten als ausgezeichnete taxonomische Prädikate in eine relational-funktionale Sprache
56 Seiten

D-95-03
Christoph Endres, Lars Klein, Markus Meyer
Implementierung und Erweiterung der Sprache *ACCP*
110 Seiten

D-95-02
Andreas Butz
BETTY
Ein System zur Planung und Generierung informativer Animationssequenzen
95 Seiten

D-95-01
Susanne Biundo, Wolfgang Tank (Hrsg.)
PuK-95, Beiträge zum 9. Workshop „Planen und Konfigurieren", Februar 1995
169 Seiten

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

1994

D-94-15
Stephan Oepen
German Nominal Syntax in HPSG
— On Syntactic Categories and Syntagmatic Relations —
80 pages

D-94-14
Hans-Ulrich Krieger, Ulrich Schäfer
TDL - A Type Description Language for HPSG, Part 2: User Guide.
72 pages

D-94-12
Arthur Sehn, Serge Autexier (Hrsg.)
Proceedings des Studentenprogramms der 18. Deutschen Jahrestagung für Künstliche Intelligenz KI-94
69 Seiten

D-94-11
F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)
Working Notes of the KI'94 Workshop: KRDB'94 - Reasoning about Structured Objects: Knowledge Representation Meets Databases
65 pages

Note: This document is no longer available in printed form.

D-94-10
F. Baader, M. Lenzerini, W. Nutt, P. F. Patel-Schneider (Eds.)
Working Notes of the 1994 International Workshop on Description Logics
118 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-94-09
Technical Staff
DFKI Wissenschaftlich-Technischer Jahresbericht 1993
145 Seiten

D-94-08
Harald Feibel
IGLOO 1.0 - Eine grafikunterstützte Beweisentwicklungsumgebung
58 Seiten

D-94-07
Claudia Wenzel, Rainer Hoch
Eine Übersicht über Information Retrieval (IR) und NLP-Verfahren zur Klassifikation von Texten
25 Seiten

D-94-06
Ulrich Buhrmann
Erstellung einer deklarativen Wissensbasis über recyclingrelevante Materialien
117 Seiten

D-94-04
Franz Schmalhofer, Ludger van Elst
Entwicklung von Expertensystemen: Prototypen, Tiefenmodellierung und kooperative Wissensevolution
22 Seiten

D-94-03
Franz Schmalhofer
Maschinelles Lernen: Eine kognitionswissenschaftliche Betrachtung
54 Seiten

Note: This document is no longer available in printed form.

D-94-02
Markus Steffens
Wissenserhebung und Analyse zum Entwicklungsprozeß eines Druckbehälters aus Faserverbundstoff
90 pages

D-94-01
Josua Boon (Ed.)
DFKI-Publications: The First Four Years
1990 - 1993
75 pages

Ein Multiagentenansatz zum Lösen von Fleet-Scheduling-Problemen

Darius Schier und Klaus Fischer

D-96-01
Document