Optimal Vehicle Scheduling in Public Transit

vorgelegt von
Diplom-Wirtschaftsmathematiker
Andreas Löbel
aus Berlin

Vom Fachbereich 3 Mathematik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
genehmigte Dissertation

Promotionsausschuß:

Vorsitzender:	Prof. Dr. Jean-Dominique Deuschel
Berichter:	Prof. Dr. Martin Grötschel
Berichter:	Prof. Dr. Günter M. Ziegler

Tag der wissenschaftlichen Aussprache: 13. November 1997

Berlin 1997
D 83

Zusammenfassung

Die vorliegende Dissertation beschäftigt sich mit der Optimierung der Fahrzeugeinsatzplanung im öffentlichen Personennahverkehr. Dieses Problem ist für die meisten praxisrelevanten Fälle schwierig ($\mathcal{NP}$-schwer). In dieser Arbeit präsentieren wir Methoden der ganzzahligen linearen Programmierung zur Lösung dieses Planungsproblems. "Vernünftige" mathematische Formulierungen des Fahrzeugeinsatzplanungsproblems basieren auf *Netzwerkfluß-Modellen* und entsprechenden ganzzahligen linearen Programmen (LP). Dies sind sogenannte *bogenorientierte Mehrgüterfluß-Modelle* bzw. *pfadorientierte Set-Partitioning-Modelle.* Wir beschäftigen uns mit beiden Ansätzen, der Schwerpunkt liegt aber auf dem bogenorientierte Mehrgüterfluß-Modell.

Mathematisch bearbeiten wir diese Modelle mit *Branch-und-Cut-* bzw. *Branch-und-Cut-und-Price-Methoden.* Reale Anwendungen führen zu riesigen LPs mit einigen Millionen ganzzahligen Variablen. Die Behandlung solcher LPs erfordert *Spalten-Erzeugungs-Verfahren* (auch *Column-Generation-Verfahren* genannt). Basierend auf Lagrange-Relaxationen entwickeln wir hierzu neue Verfahren zur Auswahl der zu erzeugenden Spalten, die wir *Lagrange-Pricing* nennen. Lagrange-Pricing-Techniken haben es erstmalig ermöglicht, LPs dieser Art mit rund 70 Millionen Variablen zu lösen.

Für den bogenorientierten (Mehrgüter-)Fluß-Zugang beschreiben wir ausführlich, wie Lagrange-Relaxationen sowie die LP-Relaxation effizient gelöst werden. Zusätzlich schlagen wir eine Heuristik vor, die schnell gute Lösungen erzeugt. Diese Heuristik beruht auf einem sog. Schedule-First-Cluster-Second-Ansatz. Eine zentrale Aufgabe bei der Lösung dieser primalen und dualen Probleme ist dabei die effiziente Behandlung von Problemen mit einem Depot. Wir zeigen, daß das bogenorientierte Mehrgüterfluß-Modell durch eine geeignete Anwendung der Dantzig-Wolfe-Dekomposition in ein pfadorientiertes Set-Partitioning-Modell überführt werden kann.

Der zweite Teil dieser Arbeit präsentiert die Rechenergebnisse zu den von uns entwickelten und implementierten Verfahren. Diese Untersuchungen basieren auf realen Testdaten von drei großen deutschen Nahverkehrsunternehmen.

Die implementierten Codes arbeiten zuverlässig und stabil. Die mit diesen Verfahren durchgeführten Testläufe lieferten hervorragende Ergebnisse: Bis auf ein Problem können alle Beispiele optimal gelöst werden. Die Lösungen des Branch-and-Cut-Verfahrens wurden auch mit den Planungsergebnissen der in der Praxis gegenwärtig eingesetzten Verfahren verglichen: Wir konnten zusätzlich mehrere Fahrzeuge einsparen sowie eine Kostenreduktion von bis zu 10 % aufzeigen.

Der mögliche Nutzen dieser Methoden ist enorm. Beispielsweise rechnet die BVG damit, den Planungsprozeß mit den von uns entwickelten Softwaretools deutlich straffen und jährlich Einsparungen in Höhe von rund 100 Millionen Mark erzielen zu können, siehe den Artikel *Auf Sparkurs zum Ziel* im Rheinischer Merkur, Nummer 39, von Schmidt [1997].

Teile der vorgestellten Methoden wurden bereits in die Planungssysteme BERTA (der Berliner Verkehrsbetriebe (BVG)) und MICROBUS II (der IVU Gesellschaft für Informatik, Verkehrs- und Umweltplanung mbH, Berlin) integriert. Darüber hinaus hat auch die Forschungsabteilung der SIEMENS AG in München dieses System erworben.

Mathematics Subject Classification (1991): 90B06, 90B10, 90C05. 90C06.

Abstract

This thesis deals with integer linear programming approaches for the $\mathcal{NP}$-hard *Multiple-Depot Vehicle Scheduling Problem* (MDVSP) in public mass transit. "Reasonable" formulations of the MDVSP are based on *network flow models* and their integer linear programming analogues. In particular, *arc-oriented multicommodity flow* models and *path-oriented set partitioning* models (derived by Dantzig-Wolfe decomposition) are two customary models for this kind of problem. In this thesis, we investigate both approaches. The main emphasis is on the solution of the multicommodity flow formulation.

The solution methods applied for the two models are *branch-and-cut* and *branch-and-cut-and-price*, respectively. Additionally, column generation techniques seem indispensable for both approaches. We have developed new column generation rules that make it possible to solve the huge linear programs with up to 70 million integer variables that come up here. These rules for selecting new columns are based on Lagrangean relaxations and, therefore, called *Lagrangean pricing*.

For the arc-oriented multicommodity flow approach, we describe the efficient solution of single-depot instances, Lagrangean relaxations, and the LP relaxation. In addition, we propose a heuristic that quickly generates good solutions. This heuristic is based on a schedule first – cluster second approach. For the path-oriented set partitioning approach, we describe its relation to the arc-oriented model.

The second part of this thesis presents the computational investigations of the solution methods that we have developed and implemented. These investigations are based on test data of three large German public transportation companies.

The computational results are excellent: Except for one problem, all encountered test instances can be solved to proven optimality. Compared with the results obtained with the best planning system currently available in practice, our test runs indicate savings of several vehicles and a cost reduction, on the average, of about 10 %.

The possible profit from using this optimization software is immense. For instance, BVG reckons on savings of about DM 100 million per year, see the newspaper article *Auf Sparkurs zum Ziel* in Schmidt [1997].

Parts of the presented methods are already integrated in the BERTA system of the Berliner Verkehrsbetriebe (BVG) and in the MICROBUS II system of the IVU Gesellschaft für Informatik, Verkehrs- und Umweltplanung mbH, Berlin. Moreover, this system has also been purchased by the research department of the SIEMENS AG, Munich.

Mathematics Subject Classification (1991): 90B06, 90B10, 90C05. 90C06.

Acknowledgements

There are good friends, persons, and institutions whose support and cooperation were essential for the successful work presented in my thesis. First of all, I wish to thank my supervisor Martin Grötschel for the opportunity to graduate. He has introduced me into the field of optimization and operations research and supported me during my time at the Konrad-Zuse-Zentrum für Informationstechnik (ZIB). I wish also to thank Günter M. Ziegler for his support. I wish to thank my friends and colleagues Ralf Borndörfer, Alexander Martin, and Rüdiger Schultz for their helpful discussions. The same thanks are devoted to Robert Bixby (Rice University, Houston, TX, and ILOG CPLEX Division). I enjoyed the times with him whenever he visited Berlin and gave me insight into many important numerical aspects of linear programming. My special thanks are due to our group's secretary, Sybille Mattrisch, for her patient proof-reading.

I am grateful to ILOG CPLEX Division (formerly: CPLEX Optimization, Inc.), for regularly providing access to the latest beta versions of CPLEX and the CPLEX Callable Library. I am also grateful to Beguelin, Dongarra, Geist, Jiang, Manchek, and Sunderam [1994] for providing PVM – Parallel Virtual Machine.

The industrial partners also play an important role: I am grateful to Manfred Völker and Anna Neufeld of HanseCom GmbH for their support to model the MDVSP and providing real-world data of the Hamburger Hochbahn AG and the Verkehrsbetriebe Hamburg-Holstein AG. I am grateful to Uwe Strubbe of IVU GmbH for providing real-world data of the Berliner Verkehrsbetriebe (BVG). I am also grateful to the Berliner Verkehrsbetriebe, the Hamburger Hochbahn AG, and the Verkehrsbetriebe Hamburg-Holstein AG for their kind permission to use and publish their data.

Last but not least, I am grateful to the German Federal Ministry of Education, Science, Research, and Technology for the financial support in the scientific program *Anwendungsorientierte Verbundprojekte auf dem Gebiet der Mathematik* (Application-Oriented Joint Projects in Mathematics), grant no. 03-GR7ZIB -7.

Berlin, November 1997 *Andreas Löbel*

Contents

Introduction

Solving transportation problems was and still is one of the driving forces behind the development of mathematical disciplines such as optimization and operations research, see Borndörfer, Grötschel, and Löbel [1995]. Truly large transportation problems have to be solved, for instance, in airline traffic (airline and crew scheduling) and public mass transit (vehicle and duty scheduling). In the past, the corresponding transportation markets have often been protected by monopolistic structures. However, deregulation of such monopolistic markets has led to world-wide competition. It is therefore obvious that competitive participants in these markets must use computer-aided tools for their operational planning process to employ their resources as efficiently as possible. Modern and sophisticated mathematical optimization techniques can help to solve such planning problems.

For instance, public transportation in the European Community is subject to such market deregulation. Monopolistic markets have become more liberal or will soon be broken up. In order to prevent their complete extinction from the market, monopolistic transportation companies will therefore have to change from deficit-oriented monopolies to competitive market players. One important factor in facing the challenges of a competitive market is, of course, cost reduction, which can be obtained by making intelligent use of the latest mathematical knowhow, see Böhmig and Wolter [1997] and Kretschmann and Lawerentz [1997].

Over the last few decades, planning large systems in public transit has been subdivided into a hierarchical process: line planning, timetable planning, vehicle scheduling, and duty scheduling and rostering. Solving each of these single steps is still a hard task. This thesis deals with one of these steps: vehicle scheduling.

The *Multiple-Depot Vehicle Scheduling Problem* (MDVSP) is to assign a fleet of vehicles, possibly stationed at several garages, to a given set of (timetabled or passenger) trips such that operational, company-specific, technical, and further side constraints are satisfied and the available resources are employed as efficiently as possible. In the last three decades, considerable research has gone into the development of academic as well as practice-oriented solution techniques for the $\mathcal{NP}$-hard MDVSP and special, often polynomially solvable, cases of it. Review articles on this topic are, for instance, Desrosiers, Dumas, Solomon, and Soumis [1995], Daduna and Paixão [1995], and Bussieck, Winter, and Zimmermann [1997].

The most successful solution approaches for the MDVSP are based on *network flow models* and their integer programming analogues. In the literature, there are two basic mathematical models of this type: First, a direct *arc-oriented model* leading to a *multicommodity flow problem* and, second, a *path-oriented model* leading to a *set partitioning problem*. The latter can also be derived from Dantzig-Wolfe (DW) decomposition applied to the first. Both approaches lead to large-scale integer programs, and *column generation techniques* are required to solve their LP relaxations. We shall explicitly discuss these two models in our literature overview. This thesis investigates both approaches. However, the main emphasis is on the solution of the multicommodity flow formulation.

To the best of our knowledge, only relatively small (artificially generated and real-world) MDVSPs have by now been *solved to proven optimality*: Forbes, Holt, and Watts [1994] report on numerical investigations with up to 3 depots. Mesquita and Paixão [1997], a recent publication on this topic, still solve problems just with up to 4 depots and 352 timetabled trips.

Practice-oriented heuristics are often based on a *single-commodity* network flow relaxation. We will show that solutions generated by methods based on network flow do not necessarily attain the optimum since, in general, they do not use all degrees of freedom. Moreover, these methods do not provide tight lower bounds or do not provide lower bounds at all.

The *contribution of this thesis* is the efficient and optimal solution of the integer linear program (ILP) derived from the multicommodity flow formulation. Column generation seems indispensable to solve large problems of this kind. In particular, we present a new technique, called *Lagrangean pricing*, that is based on two Lagrangean relaxations of the multicommodity flow model.

Our computational investigations are performed on large-scale data from three German public transportation companies: the Berliner Verkehrsbetriebe (BVG), the Hamburger Hochbahn AG (HHA), and the Verkehrsbetriebe Hamburg-Holstein AG (VHH). These instances involve problems with up to 49 depots, about 25 thousand timetabled trips, and about 70 million integer decision variables. These test instances have been provided by our partners HanseCom GmbH, Hamburg, and IVU Gesellschaft für Informatik, Verkehrs- und Umweltplanung mbH (IVU), Berlin. Test runs on this test set show that our method is able to solve problems of this size optimally. These problems are orders of magnitude larger than the instances successfully solved with other approaches, as far as we know.

The interested reader can find further information about the involved companies BVG, HanseCom, HHA, IVU, and VHH (in alphabetical order) via WWW at www.bvg.de, www.hansecom.com, www.hochbahn.com, www.ivu-berlin.de, and www.oepnv.de/vhh, respectively.

Our research has been supported by the German Federal Ministry of Education, Science, Research, and Technology in the program *Anwendungsorientierte Verbundprojekte auf dem Gebiet der Mathematik* (Application-Oriented Joint Projects in Mathematics). This program aimed at giving financial support to joint projects of academic institutions and partners from industry. It was the goal to improve available or to develop new mathe-

matical techniques and to transfer the resulting software tools into practice. And indeed, parts of our software have already been integrated in the planning systems BERTA of the Berliner Verkehrsbetriebe and MICROBUS II of the IVU GmbH. Additionally, the research department of the SIEMENS AG, Munich, has also purchased our system. With our software, the Berliner Verkehrsbetriebe expects to save about DM 100 million per year, see the article *Auf Sparkurs zum Ziel* by V. A. Schmidt [1997].

This thesis contains 12 chapters and is divided into five parts: *basics* (Chapter 1), *the problem* (Chapters 2–4), *lower and upper bounds* (Chapters 5–9), *exact methods* (Chapters 10–11), and *computations* (Chapter 12).

Basics: In Chapter 1, we give an introduction to some basics concerning notation, linear algebra, polyhedral theory, (integer) linear programming, graph theory, and network flows.

The Problem: This part is divided into three chapters: a problem description, complexity investigations, and a literature overview.

The MDVSP is introduced in Chapter 2, and we present a multicommodity flow formulation with its integer programming analogue. We also distinguish our multicommodity flow formulation from some other (arc-oriented) models that have been presented in the literature.

The complexity of the MDVSP is investigated in Chapter 3, which is divided into one section for the polynomially solvable single-depot case and another for the $\mathcal{NP}$-hard multiple-depot case.

Chapter 4 contains a detailed literature overview. In addition, we discuss the arc-oriented multicommodity flow and the path-oriented set partitioning formulations.

Lower and Upper Bounds: In this part, we present the algorithmic tools that we have used to solve large problem instances with our branch-and-cut method.

It starts with the solution technique for single-depot instances in Chapter 5. Such problems are solved with a network simplex algorithm combined with column generation.

Chapter 6 deals with two Lagrangean relaxations of the ILP and presents the subgradient methods which we apply for their solution. The occurring inner minimization problems of the Lagrangean duals are solved by the network simplex code presented in the chapter before. In particular, Lagrangean relaxations are used to compute fast and tight lower bounds.

In Chapter 7, we will describe in detail the basic ingredients of our LP method that are indispensable to solve the LP relaxation of large problems. In particular, we introduce Lagrangean pricing. We have also investigated certain recent approximation approaches for the solution of the LP relaxation. We do not believe that such algorithms can substantially help solving the LP relaxation that we investigate here.

Chapter 8 deals with some heuristics. It starts with two opening heuristics: the well-known cluster first – schedule second approach (CF-SS) and a schedule – cluster – reschedule algorithm, which is a composition of (the other well-known) schedule first – cluster second approach and CF-SS. The chapter continues with *LP-plunging*, which is an LP-based iteratively rounding heuristic and used within our branch-and-cut algorithm. Last but not least, we give a short introduction to the vehicle scheduling of HOT.

Chapter 9 presents some results of our polyhedral investigations of the MDVSP.

Exact Methods: This part presents exact branch-and-bound and branch-and-cut-and-price approaches for the multicommodity flow and the DW set partitioning formulations, respectively.

Chapter 10 give a composition of the concepts presented in the previous part resulting in an efficient method to solve MDVSP instances.

In Chapter 11, we show how the DW decomposition formulation is derived from the multicommodity ILP formulation, discuss the relation between the arc-oriented multicommodity flow and the path-oriented set partitioning formulations, and provide a branch-and-cut-and-price algorithm for the solution of the DW decomposition.

Computations: The methods of the Parts II and IV have been tested on real-world data of BVG, HHA, and VHH. In Chapter 12, we summarize the results of our computational tests for single-depot instances (solved with our network simplex implementation) and multiple-depot instances (solved with branch-and-cut and decomposition approaches, respectively).

Chapter 1

Notation and Preliminaries

This chapter introduces some basics concerning notation, linear algebra, polyhedral theory, (integer) linear programming, graph theory, and network flows. Readers who are familiar with the basics of these fields may wish to continue with Chap. 2. In what follows, the accent is rather on collecting prerequisites than on completeness. Parts of the exposition follow Grötschel, Lovász, and Schrijver [1988]. For more detailed information, we recommend

- Bazaraa, Jarvis, and Sherali [1990], Chvátal [1980], Grötschel, Lovász, and Schrijver [1988], Luenberger [1989], Nemhauser and Wolsey [1988], and Schrijver [1989] for polyhedral theory and (integer) linear programming,
- Berge [1973] and Grötschel, Lovász, and Schrijver [1988] for graph theory,
- Ahuja, Magnanti, and Orlin [1993,1995] and Bazaraa, Jarvis, and Sherali [1990] for network flows, and
- Garey and Johnson [1979] and Papadimitriou and Steiglitz [1982] for complexity theory.

1.1 Sets, Vector Spaces, and Matrices

By $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ we denote the sets of natural, integer, rational, and real numbers, respectively. $\mathbb{Z}_+$, $\mathbb{Q}_+$, and $\mathbb{R}_+$ are the subsets of nonnegative elements. We assume that $0 \notin \mathbb{N}$ and denote $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. The symbols $\mathbb{N}^n$, $\mathbb{Z}^n$, $\mathbb{Q}^n$, and $\mathbb{R}^n$ are used for the sets of vectors with n components and entries in N, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$. $\mathbb{R}^n$ is understood as the usual n-dimensional Euclidean vector space with an **inner product** $x^{\mathrm{T}}y := \sum_{i=1}^{n} x_i y_i$ for $x, y \in \mathbb{R}^n$. The superscript “T” denotes **transposition**, and, unless stated otherwise, $x \in \mathbb{R}^n$ is always a column vector. $\mathbb{1} \in \mathbb{R}^n$ denotes the n-vector with all entries equal to one, and $e^j \in \mathbb{R}^n$ is the j-th canonical unit vector.

A vector $x = \sum_{i=1}^{k} \lambda_i x_i$ is called **linear combination** of the vectors $x_1, x_2, \ldots, x_k \in \mathbb{R}^n$

if $\lambda_i \in \mathbb{R}$ for $i = 1, \dots k$. If, in addition,

$$\left.\begin{array}{l} \lambda_i \geqslant 0 \ \ \forall i \\ \qquad\qquad \lambda^{\mathrm{T}}\mathbb{1} = 1 \\ \lambda_i \geqslant 0 \ \ \forall i, \ \ \lambda^{\mathrm{T}}\mathbb{1} = 1 \end{array}\right\} \text{ we call } \ x \ \text{ a(n) } \left\{\begin{array}{c} \textbf{conic} \\ \textbf{affine} \\ \textbf{convex} \end{array}\right\} \textbf{ combination}$$

of the vectors $x_1, x_2, \dots, x_k$. These combinations are called **proper** if neither $\lambda = 0$ nor $\lambda = e^j$ for some $j \in \{1, 2, \dots, k\}$. For a nonempty subset $S \subseteq \mathbb{R}^n$, we denote by

$$\left.\begin{array}{l} \mathrm{lin}(S) \\ \mathrm{cone}(S) \\ \mathrm{aff}(S) \\ \mathrm{conv}(S) \end{array}\right\} \text{ the } \left\{\begin{array}{c} \textbf{linear} \\ \textbf{conic} \\ \textbf{affine} \\ \textbf{convex} \end{array}\right\} \textbf{hull} \text{ of the elements of } S,$$

that is the set of all vectors that are linear (conic, affine, convex) combinations of finitely many vectors of S. For the empty set, we define $\mathrm{lin}(\emptyset) := \mathrm{cone}(\emptyset) := \{0\}$ and $\mathrm{aff}(\emptyset) := \mathrm{conv}(\emptyset) := \emptyset$.

A subset $S \subseteq \mathbb{R}^n$ is called a(n)

$$\left\{\begin{array}{l} \textbf{linear subspace} \\ \textbf{cone} \\ \textbf{affine subspace} \\ \textbf{convex set} \end{array}\right\} \text{ if } \ S = \left\{\begin{array}{l} \mathrm{lin}(S) \\ \mathrm{cone}(S) \\ \mathrm{aff}(S) \\ \mathrm{conv}(S) \end{array}\right\}.$$

A subset $S \subseteq \mathbb{R}^n$ is called **linearly** (**affinely**) **independent** if none of its members is a proper linear (affine) combination of elements of S; otherwise S is called **linearly** (**affinely**) **dependent**. It is well known that a linearly (affinely) independent subset of $\mathbb{R}^n$ contains at most n elements ($n + 1$ elements). For any set $S \subseteq \mathbb{R}^n$, the **rank** of S (**affine rank** of S), denoted by $\mathrm{rank}(S)$ ($\mathrm{arank}(S)$), is the cardinality of the largest linearly (affinely) independent subset of S. For any subset $S \subseteq \mathbb{R}^n$, the **dimension** of S, denoted by $\dim(S)$, is the cardinality of the largest affinely independent subset of S minus one, i.e., $\dim(S) = \mathrm{arank}(S) - 1$. A set $S \subseteq \mathbb{R}^n$ with $dim(S) = n$ is called **full-dimensional**.

If E and R are sets, then R^E is the set of mappings from E to R. If E is finite, it is very convenient to consider the elements of R^E as $|E|$-dimensional vectors where each component of a vector $x \in R^E$ is indexed by an element of E, i.e., $x = (x_e)_{e \in E}$. For $F \subseteq E$, the vector $\chi^F \in R^E$ defined by $\chi^F_e = 1$ if $e \in F$ and $\chi^F_e = 0$ if $e \in E \setminus F$ is called the **incidence vector** of F. For a set G (not necessarily $G \subseteq E$) $x(G) \in R$ is defined as

$$x(G) := \sum_{e \in G \cap E} x_e.$$

For a real number α, $\lfloor \alpha \rfloor$ denotes the largest integer not larger than α (the **floor** or **lower integer part** of α), $\lceil \alpha \rceil$ denotes the smallest integer not smaller than α (the **ceiling** or **upper integer part** of α), and $\lceil \alpha \rfloor := \lceil \alpha - \frac{1}{2} \rceil$ denotes the integer nearest to α.

For any set R, $R^{m\times n}$ denotes the set of $m\times n$-matrices with entries in R. For a matrix $A \in R^{m\times n}$, we usually assume that the row index set of A is $\{1,\dots,m\}$ and that the column index set is $\{1,\dots,n\}$. Unless specified otherwise, the elements or entries of $A \in R^{m\times n}$ are denoted by a_{ij}, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$; we write $A = (a_{ij})$. Vectors with n components are also considered as $n\times 1$-matrices.

If I is a subset of the row index set M of a matrix A and J is a subset of the column index set N of A, then A_{IJ} denotes the submatrix of A induced by those rows and columns of A whose indices belong to I and J, respectively. Instead of A_{MJ} (A_{IN}, resp.) we frequently write $A_{.J}$ ($A_{I.}$, resp.). $A_{i.}$ is the i-th row of A (so it is a row vector), and $A_{.j}$ is the j-th column of A.

Whenever we operate with vectors and matrices, but do not explicitly specify their dimensions, we always assume that their dimensions are compatible.

The **identity matrix** is denoted by I or, if we want to stress its dimension, by I_n. The symbol 0 stands for any appropriately sized matrix, which has all entries equal to zero, and similarly for any zero vector. If $A \in R^{m\times p}$ and $B \in R^{m\times q}$ then (A, B) (or just (AB) if this does not lead to confusion) denotes the matrix in $R^{m\times(p+q)}$ whose first p columns are the columns of A and whose other q columns are those of B. Analogously, if $C \in R^{r\times n}$ and $D \in R^{s\times n}$ then $\begin{pmatrix} C \\ D \end{pmatrix}$ denotes the matrix in $R^{(r+s)\times n}$ whose first r rows are the rows of C and whose other s rows are those of D. The inverse matrix of a nonsingular $n\times n$-matrix A is denoted by A^{-1}.

The **rank of a matrix** A (notation: $\mathrm{rank}(A)$) is the rank of the set of its column vectors. This is known to be equal to the rank of the set of its row vectors. An $m\times n$-matrix A is said to have **full row rank** (**full column rank**) if $\mathrm{rank}(A) = m$ ($\mathrm{rank}(A) = n$).

1.2 Polyhedra

If A is a real $m\times n$-matrix and $b \in \mathbb{R}^m$, then $Ax \leqslant b$ is called a **system of (linear) inequalities** and $Ax = b$ a **system of (linear) equations**. The solution set $P := \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$ of a system of inequalities is called a **polyhedron**. A polyhedron P that is **bounded** is called a **polytope**. A **polyhedral cone** is a cone that is also a polyhedron.

If $a \in \mathbb{R}^n \setminus \{0\}$ and $a_0 \in \mathbb{R}$, then $\{x \in \mathbb{R}^n \mid a^{\mathrm{T}}x \leqslant a_0\}$ is called a **halfspace**, and $\{x \in \mathbb{R}^n \mid a^{\mathrm{T}}x = a_0\}$ a **hyperplane**. To shorten notation, we shall sometimes speak of the hyperplane $a^{\mathrm{T}}x = a_0$ and the halfspace $a^{\mathrm{T}}x \leqslant a_0$. Every polyhedron is the intersection of finitely many halfspaces.

We call an inequality $a^{\mathrm{T}}x \leqslant a_0$ **valid** with respect to P if $P \subseteq \{x \in \mathbb{R}^n \mid a^{\mathrm{T}}x \leqslant a_0\}$. A set $F \subseteq P$ is called a **face** of P if there exists a valid inequality $a^{\mathrm{T}}x \leqslant a_0$ for P such that $F = \{x \in P \mid a^{\mathrm{T}}x = a_0\}$. We say that F is the **face defined** (or **induced**) by $a^{\mathrm{T}}x \leqslant a_0$. The subset $\mathrm{eq}(F) := \{i \in M \mid A_{i.}x = b_i \;\; \forall\, x \in F\}$ of the row index set of A is called the

equality set of F.

If x is a point in a polyhedron P such that $\{x\}$ is a face of P, then x is called a **vertex** of P. A **facet** of P is a maximal face F of $P := \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$ with $\emptyset \neq F \neq P$. Facet defining conditions are given by the following theorem:

(1.1) Theorem. Let $P := \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$ be a polyhedron and F, induced by the inequality $a^{\mathrm{T}}x \leqslant a_0$, be a nontrivial face of P. Then the following are equivalent:

i. F is a facet of P.
ii. $\dim(F) = \dim(P) - 1$.
iii. For each valid inequality $d^{\mathrm{T}}x \leqslant d_0$ satisfying $F \subseteq \{x \in P \mid d^{\mathrm{T}}x = d_0\}$, there exists some vector $u \in \mathbb{R}^{\mathrm{eq}(P)}$ and $0 \leqslant v \in \mathbb{R}$ such that $d_0 = va_0 + u^{\mathrm{T}}b_{\mathrm{eq}(P)}$ and $d^{\mathrm{T}} = va^{\mathrm{T}} + u^{\mathrm{T}}A_{\mathrm{eq}(P)}$..

1.3 Linear and Integer Linear Programs

1.3.1 Linear Programming and Duality

Given an $m \times n$-matrix A, a vector b, and a vector c. A **linear programming problem (LP)**, we also just say **linear program**, is the task to find a vector $x^* \in P = \{x \mid Ax = b,\ x \geqslant 0\}$ minimizing the linear function $c^{\mathrm{T}}x$ over P. We will usually write an LP in one of the following forms:

$$\min_{\substack{Ax = b \\ x \geqslant 0}} c^{\mathrm{T}}x,$$

$$\min\{c^{\mathrm{T}}x \mid Ax = b,\ x \geqslant 0\},$$
$$\min c^{\mathrm{T}}x,\ Ax = b,\ x \geqslant 0,$$
$$\text{or } \min c^{\mathrm{T}}x,\ x \in P.$$

A vector $x \in P$ is called a **feasible solution** of the LP, and a feasible solution x^* is called an **optimal solution** if $c^{\mathrm{T}}x^* \leqslant c^{\mathrm{T}}x$ for all feasible $x \in P$. The linear function $c^{\mathrm{T}}x$ is called the **objective function** of the LP. The same terminology applies if minimization is replaced by maximization.

With every LP $\min\{c^{\mathrm{T}}x \mid Ax = b,\ x \geqslant 0\}$, we associate the dual LP $\max\{y^{\mathrm{T}}b \mid y^{\mathrm{T}}A \leqslant c^{\mathrm{T}}\}$ referring to the original LP as the primal one. Both problems are interrelated by the following duality theorem of linear programming:

(1.2) Theorem. If both the linear program (P) $\min\{c^{\mathrm{T}}x \mid Ax = b,\ x \geqslant 0\}$ and its dual linear program (D) $\max\{u^{\mathrm{T}}b \mid u^{\mathrm{T}}A \leqslant c^{\mathrm{T}}\}$ have feasible solutions then both LPs have optimal solutions yielding the same (optimal) objective value. If one of (P) or (D) has no feasible solution then the other is either unbounded or has also no feasible solution, and if one of (P) or (D) is unbounded then the other has no feasible solution.

If the polyhedron P is given by an arbitrary collection of linear inequality and equality constraints, the following corollary displays the general scheme for forming the dual to a primal LP.

(1.3) Corollary. The primal LP

$$\min d^{\mathrm{T}}x + e^{\mathrm{T}}y \tag{1.4a}$$

subject to

$$\begin{array}{lrcrcl} (1.4\text{b}) & Ax & + & By & = & a, \\ (1.4\text{c}) & Cx & + & Dy & \geqslant & b, \\ (1.4\text{d}) & Ex & + & Fy & \leqslant & c, \\ (1.4\text{e}) & & & x & \geqslant & 0, \end{array}$$

and its dual LP

$$\max u^{\mathrm{T}}a + v^{\mathrm{T}}b + w^{\mathrm{T}}c \tag{1.5a}$$

subject to

$$\begin{array}{lrcrcrcl} (1.5\text{b}) & u^{\mathrm{T}}A & + & v^{\mathrm{T}}C & + & w^{\mathrm{T}}E & \leqslant & d^{\mathrm{T}}, \\ (1.5\text{c}) & u^{\mathrm{T}}B & + & v^{\mathrm{T}}D & + & w^{\mathrm{T}}F & = & e^{\mathrm{T}}, \\ (1.5\text{d}) & & & v \geqslant 0, & & w & \leqslant & 0, \end{array}$$

yield the same optimal value provided that both linear programs are nonempty.

The following results provide some useful optimality conditions for linear programs.

(1.6) Theorem. Weak complementary slackness theorem. Consider the primal LP (1.4) and its dual LP (1.5). Let $(x^{*\mathrm{T}}, y^{*\mathrm{T}})^{\mathrm{T}}$ and $(u^{*\mathrm{T}}, v^{*\mathrm{T}}, w^{*\mathrm{T}})^{\mathrm{T}}$ denote some optimal solutions for (1.4) and (1.5), respectively. Then

$$\begin{aligned} v_i^* > 0 &\implies C_{i\cdot}x^* + D_{i\cdot}y^* = b_i, \\ w_i^* < 0 &\implies E_{i\cdot}x^* + F_{i\cdot}y^* = a_i, \\ x_j^* > 0 &\implies u^{*\mathrm{T}}A_{\cdot j} + v^{*\mathrm{T}}C_{\cdot j} + w^{*\mathrm{T}}F_{\cdot j} = d_j. \end{aligned}$$

(1.7) Theorem. Strong complementary slackness theorem. If the primal LP (P) $\min\{c^{\mathrm{T}}x \mid Ax = b,\ x \geqslant 0\}$ and its dual LP (D) $\max\{u^{\mathrm{T}}b \mid u^{\mathrm{T}}A \leqslant c^{\mathrm{T}}\}$ have feasible solutions then there exists optimal solutions x^* and u^* such that for each column j of A holds

$$\begin{aligned} x_j^* > 0 &\iff u^{*\mathrm{T}}A_{\cdot j} = c_j, \\ u^{*\mathrm{T}}A_{\cdot j} < c_j &\iff x_j^* = 0. \end{aligned}$$

1.3.2 Integer Linear Programming

Given an $m{\times}n$-matrix A, a vector b, and a vector c. An **integer linear programming problem (ILP)**, we also just say **integer linear program**, is the task of finding a vector $x^* \in X = \{x \mid Ax = b,\, x \geqslant 0,\, x \text{ integral}\}$ minimizing the linear function $c^{\mathrm{T}}x$ over X.

Let $\tilde{X} := \{x \mid Ax = b,\, x \geqslant 0\}$ denote the continuous relaxation of X. Given an ILP $\min\{c^{\mathrm{T}}x \mid x \in X\}$, then $\min\{c^{\mathrm{T}}x \mid x \in \tilde{X}\}$ is called its **LP relaxation**. It is easy to see that the following inequalities hold provided that all LPs and ILPs are feasible:

$$\min_{\substack{Ax=b\\ x\geqslant 0\\ x \text{ integral}}} c^{\mathrm{T}}x \;\geqslant\; \min_{\substack{Ax=b\\ x\geqslant 0}} c^{\mathrm{T}}x \;\overset{\text{Theorem 1.2}}{=}\; \max_{u^{\mathrm{T}}A \leqslant c^{\mathrm{T}}} c^{\mathrm{T}}x \;\geqslant\; \max_{\substack{u^{\mathrm{T}}A \leqslant c^{\mathrm{T}}\\ y \text{ integral}}} c^{\mathrm{T}}x$$

1.3.3 Lagrangean Relaxation

Another possible relaxation of an ILP is Lagrangean relaxation. Its general approach is outlined in Schrijver [1989] and, especially for network flow problems, in Ahuja, Magnanti, and Orlin [1993]. The basic idea is the following. Let X denote some nonempty polytope; for simplicity and for the sake of illustration assume $X := \{x \mid Ax = b,\, x \geqslant 0,\, x \text{ integral}\}$. Let $\tilde{X} := \{x \mid Ax = b,\, x \geqslant 0\}$ denote the continuous relaxation of X. Suppose we consider the ILP

$$\min\{c^{\mathrm{T}}x \mid Dx = d,\, x \in X\} \tag{1.8}$$

and its LP relaxation

$$\min\{c^{\mathrm{T}}x \mid Dx = d,\, x \in \tilde{X}\}. \tag{1.9}$$

A **Lagrangean relaxation** of (1.8) with respect to $Dx = d$ is the optimization problem

$$\max_u \left[\min_{x\in X}\left\{c^{\mathrm{T}}x \;-\; u^{\mathrm{T}}\big(Dx - d\big)\right\}\right]. \tag{1.10}$$

The inner minimization problem leads to a concave and piecewise linear function

$$L(u, X) \;:=\; u^{\mathrm{T}}d \;+\; \min_{x\in X}\left\{\big(c^{\mathrm{T}} - u^{\mathrm{T}}D\big)\,x\right\} \tag{1.11}$$

whose maximization problem $\max_u L(u, X)$ is called the **Lagrangean dual**. The components of u are called the **Lagrange multipliers**.

If we apply Lagrangean relaxation to the LP relaxation (1.9), we end up with some counterpart $L(u, \tilde{X})$. All together, we have the following theorem.

(1.12) Theorem.

$$\min_{\substack{D\,x\,=\,d\\ x\,\in\,X}} c^{\mathrm T}x \;\geqslant\; \max_u L(u,X) \;\geqslant\; \max_u L(u,\tilde X) \;=\; \min_{\substack{D\,x\,=\,d\\ x\,\in\,\tilde X}} c^{\mathrm T}x$$

Proof: Let x^* and u^* denote some optimal solutions of (1.8) and (1.10), respectively. Then

$$\min_{\substack{D\,x\,=\,d\\ x\,\in\,X}} c^{\mathrm T}x \;\stackrel{x^*\text{ opt.}}{=}\; c^{\mathrm T}x^* \;=\; c^{\mathrm T}x^* + \left(u^{*\mathrm T}d - u^{*\mathrm T}d\right) \;\stackrel{Dx=d}{=}\; c^{\mathrm T}x^* + u^{*\mathrm T}d - u^{*\mathrm T}D\,x \;=$$
$$u^{*\mathrm T}d + (c^{\mathrm T} - u^{*\mathrm T}D)\,x \;\geqslant\; u^{*\mathrm T}d + \min_{x\in X}\,(c^{\mathrm T} - u^{*\mathrm T}D)\,x \;=\; L(u^*,X) \;\stackrel{u^*\text{ opt.}}{=}\; \max_u L(u,X)$$

proves the first inequality. The second inequality holds since for each u

$$L(u,X) - u^{\mathrm T}d = \min_{x\in X}(c^{\mathrm T} - u^{\mathrm T}D)\,x \;\stackrel{X\subseteq\tilde X}{\geqslant}\; \min_{x\in\tilde X}(c^{\mathrm T} - u^{\mathrm T}D)\,x = L(u,\tilde X) - u^{\mathrm T}d.$$

The third inequality is proved by

$$\max_u L(u,\tilde X) \;=\; \max_u\left[u^{\mathrm T}d \;+\; \min_{x\in\tilde X}\left\{\left(c^{\mathrm T} - u^{\mathrm T}D\right)x\right\}\right] \;\stackrel{\text{Theorem 1.2}}{=}$$
$$\max_u\left[u^{\mathrm T}d \;+\; \max_{v^{\mathrm T}A\leqslant c^{\mathrm T}-u^{\mathrm T}D} v^{\mathrm T}b\right] \;=\; \max_{u^{\mathrm T}D+v^{\mathrm T}A\leqslant c^{\mathrm T}} u^{\mathrm T}d + v^{\mathrm T}b \;\stackrel{\text{Theorem 1.2}}{=}\; \min_{\substack{D\,x\,=\,d\\ x\,\in\,\tilde X}} c^{\mathrm T}x$$

❑

(1.13) Remark. Lagrangean relaxation aims at problems $L(u,X)$ that are easier to solve than the original problem (1.8). In particular, subproblems may benefit from separability or the convex hull of the set X may coincide with its continuous relaxation $\tilde X$. The latter is often the case in network optimization. Then the Lagrangean relaxation (1.10) and the LP relaxation (1.9) yield the same optimal objective value, and direct LP techniques can be applied to solve the Lagrangean relaxation. Moreover, it follows directly from the proof of Theorem 1.12 that the Lagrange multipliers simply correspond to the dual multipliers of $D\,x = d$.

(1.14) Remark. Typically, X is chosen such that $L(u,X)$ (1.11) can be evaluated in (pseudo-)polynomial time for any given u. For those cases, the Lagrangean dual can be theoretically solved in (pseudo-)polynomial time, see Schrijver [1989].

(1.15) Remark. If there is no danger of confusion with the considered polytopes X and $\tilde X$, we shall also write $L(u)$ instead of $L(u,X)$ and $L(u,\tilde X)$, respectively.

Since the Lagrangean dual is a concave maximization problem, subgradient methods from non-smooth optimization are employed for its solution. An overview about such methods is given in Hiriart-Urruty and Lemaréchal [1993].

1.4 Graphs

Our terminology for graph theory is an extended version of the terminology introduced by Grötschel, Lovász, and Schrijver [1988]. They themselves use a mixture of Berge [1973], Bollobás [1978], Bondy and Murty [1976], and Lawler [1976].

A **directed graph** (or **digraph**) $D = (V, A)$ consists of a finite nonempty **node** set V and a finite **arc** set A. For each arc $a \in A$, we associate an ordered pair (i, j) of nodes, called **endnodes**, at which i is the **initial endnode** (or **tail**) and j is the **terminal endnode** (or **head**) of a. We say that an arc $a = (i, j)$ **goes from** i **to** j, that a is **incident from** i and **incident to** j, and that a **leaves** i and **enters** j. If there is an arc going from i to j, we say that i is a **predecessor** of j and that j is a **successor** of i.

An **isolated** node has no incident arc, and a **loop** is an arc (i, i). We consider only graphs without isolated nodes and without loops. **Parallel** arcs are possible, i. e., two different arcs a and a' can have the same tail i and the same head j. If there is no danger of confusion, (i, j) denotes the arc having tail i and head j. Whenever parallel arcs occur, we will handle them carefully.

For subsets $W \subseteq V$ and $B \subseteq A$, $V(B)$ denotes the set of nodes incident to some arc in B, and $A(W)$ is the set of arcs with head and tail in W.

If $i \in V$ then the set of arcs having i as initial (terminal) node is denoted by $\delta^+(i)$ ($\delta^-(i)$); we set $\delta(i) := \delta^+(i) \cup \delta^-(i)$. The numbers $|\delta^+(i)|$, $|\delta^-(i)|$, and $|\delta(i)|$ are called the **outdegree**, **indegree**, and **degree** of i, respectively. For any set $W \subseteq V$, we set $\delta^+(W) := \big\{(i, j) \in A |\ i \in W \text{ and } j \notin W\big\}$, $\delta^-(W) := \delta^+(V \setminus W)$, and $\delta(W) := \delta^+(W) \cup \delta^-(W)$. The symbols $\delta^+(\cdot)$, $\delta^-(\cdot)$, and $\delta(\cdot)$ always belong to the digraph represented by $D = (V, A)$. If there is a danger of confusion, we write $\delta^+_D(\cdot)$, $\delta^-_D(\cdot)$, and $\delta_D(\cdot)$. For two subsets $U, W \subset V$, we set $(U \to W) := \delta^+(U) \cap \delta^-(W)$.

A **path** in D is a finite sequence $P = i_0, a_1, i_1, a_2, i_2, \ldots, a_k, i_k$, $k \geqslant 0$, that begins and ends with a node and contains mutually distinct nodes (with a possible exception for i_0 and i_k). The nodes i_l and the arcs a_l of P appear alternately such that i_{l-1} and i_l are the endnodes of a_l, for $l = 1, 2, \ldots, k$. The nodes i_0 and i_k are called the **origin** and the **terminus**, respectively, or the **endnodes** of P. If a node s is the origin and a node t is the terminus of P, P is called an $[s, t]$-path. The nodes $i_1, \ldots, i_{k-1}$ are called the **internal nodes** of P. The number k is the **length** of the path. A **directed path** or **dipath** in D is a path in which all arcs $a_l = (i_{l-1}, i_l)$, for $l = 1, 2, \ldots, k$. The directed version of an $[s, t]$-path is denoted by (s, t)-dipath.

Two nodes s, t of a digraph D are said to be **connected** if D contains an $[s, t]$-path. D is called **connected** if every two nodes of D are connected. A path is called **closed** if it has nonzero length and its origin and terminus are identical. A closed path (dipath) in which the origin and all internal nodes are different is called a **circuit** (**dicycle** or **directed cycle**). An arc set $T \subseteq A$ is called a **tree** if T does not contain a circuit and if the digraph $(T(V), T)$ is connected. A tree is called a **spanning tree** if $T(V) = V$.

We shall also use the words "path", "circuit", or "tree" to denote the arc set of a path, circuit, or tree, e. g., the incidence vector of a path is the incidence vector of the path's arc set.

1.5 Network Flows

Network flow problems and algorithms have been profoundly investigated during the last decades. Veldhorst [1993] has compiled a bibliography containing 370 references to single-, multicommodity, and other classes of flow papers published by 1993.

1.5.1 The Minimum-Cost Flow Problem

One of the most extensively studied and best understood problems in operations research is the minimum-cost flow problem. To formulate this problem, we start from some connected digraph $D = (V, A)$ together with some linear cost function $c \in \mathbb{Q}^A$, upper bounds $u \in \mathbb{Q}^A_+$, and **node imbalances** $b \in \mathbb{Q}^V$ fulfilling $\mathbb{1}^{\mathrm{T}} b = 0$. The minimum-cost flow problem is to find a vector $x^* \in \mathbb{Q}^A$ such that x^* is an optimal solution to the linear program

$$\min \sum_{a \in A} c_a x_a \tag{1.16a}$$

subject to

$$x\big(\delta^+(i)\big) - x\big(\delta^-(i)\big) = b_i, \qquad \forall\, i \in V, \tag{1.16b}$$
$$0 \leqslant x_{ij} \leqslant u_{ij}, \qquad \forall\, (i,j) \in A. \tag{1.16c}$$

The equations (1.16b) are called **flow conservation constraints** and the inequalities (1.16c) are the **flow capacities** on x. A flow x is called **feasible** if it satisfies the flow conservation constraints and the flow capacities. A node i is called **supply**, **demand**, or **transshipment** node depending upon whether b_i is larger than, smaller than, or equal to zero. With $\mathcal{N}$ denoting the node-arc incidence matrix of D, (1.16) reads

$$\min\{c^{\mathrm{T}} x \mid \mathcal{N} x = b,\ 0 \leqslant x \leqslant u\}. \tag{1.17}$$

It is well known that $\mathcal{N}$ and, thus, the constraint matrix of (1.17) are totally unimodular. For integer vectors u and b, there always exists an integer optimal flow, see Grötschel, Lovász, and Schrijver [1988].

Let $\pi \in \mathbb{Q}^V$ (the **node potentials**) and $\eta \in \mathbb{Q}^A$ be the dual multipliers for the flow conservation constraints (1.16b) and the upper bounds in (1.16c). The dual problem of (1.17) is

$$\max\{\pi^{\mathrm{T}} b - \eta^{\mathrm{T}} u \mid \pi^{\mathrm{T}} \mathcal{N} - \eta^{\mathrm{T}} \leqslant c^{\mathrm{T}},\ \eta \geqslant 0\}, \tag{1.18}$$

which is

$$\max \sum_{i \in V} \pi_i b_i \quad - \sum_{(i,j) \in A} \eta_{ij} u_{ij} \tag{1.19a}$$

subject to

$$\begin{aligned} \pi_i - \pi_j - \eta_{ij} &\leqslant c_{ij}, \quad \forall\, (i,j) \in A, &(1.19b)\\ \eta_{ij} &\geqslant 0, \quad \forall\, (i,j) \in A. &(1.19c)\end{aligned}$$

Note that our model (1.17) imposes a zero lower bound for the flow on each arc $a \in A$. This is no loss of generality since lower bounds $l \in \mathbb{Q}^A$ (with $l \leqslant u$) can easily be transformed to 0 by substituting the flow vector x by $x' + l$, $x' \in \mathbb{Q}^A$. Then, the system $l \leqslant x \leqslant u$ transforms to $0 \leqslant x' \leqslant u - l$, and $\mathcal{N}x = b$ transforms to $\mathcal{N}x' = b - \mathcal{N}l$, which is equivalent to decrease b_i and to increase b_j by l_{ij} for all $(i,j) \in A$. The objective min $c^{\mathrm{T}}x$ transforms to $c^{\mathrm{T}}l +$ min $c^{\mathrm{T}}x'$. Figure 1.1, which is taken from Ahuja, Magnanti, and Orlin [1993], displays such a lower bound transformation.

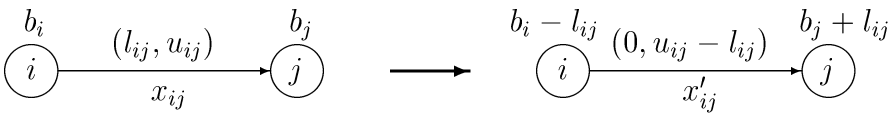


Figure 1.1: Transformation to zero lower bounds.

1.5.2 Flow Decomposition

Up to now, we have only considered flows that are defined on arcs. Whenever we talk about a flow, we mean such an **arc flow**. It is also possible, however, to define flows on dipaths and dicycles. For those cases, we explicitly refer to them by **dipath** or **dicycle flow**, and we sometimes write only **path** or **cycle flow**.

Let $\mathcal{P}$ denote all possible dipaths between any pair of nodes in D, and let $\mathcal{W}$ denote all dicycles in D. We set $P := \{\chi^R \mid R \in \mathcal{P}\}$ and $W := \{\chi^S \mid S \in \mathcal{W}\}$. For each $p \in P$ and each $w \in W$, let $f_p \geqslant 0$ and $f_w \geqslant 0$ denote the flow values on p and w, respectively, and let $x(p) := f_p\, p \in \mathbb{R}^A$ and $x(w) := f_w\, w \in \mathbb{R}^A$ denote the flow vectors corresponding to arc flows. Then every path and cycle flow uniquely determines an arc flow $x \in \mathbb{R}^A$ by

$$x := \sum_{p \in P} x(p) + \sum_{w \in W} x(w).$$

The following theorem shows how arc flows can be decomposed into path and cycle flows.

(1.20) Theorem. Every path and cycle flow has a unique representation as a nonnegative arc flow. Conversely, every nonnegative arc flow x can be represented as a path and cycle flow – though not necessarily uniquely – with the following two properties:

(a) Every dipath with positive flow connects a source node to a demand node.

(b) At most $|V| + |A|$ dipaths and dicycles have nonzero flow; out of these, at most $|A|$ dicycles have nonzero flow.

Proof: Flow Decomposition Theorem of Ahuja, Magnanti, and Orlin [1993], page 80, or Ahuja, Magnanti, and Orlin [1989], page 237. ❑

(1.21) Lemma. If $b_i = 0$, for all $i \in V$, each nonnegative flow x can be represented as a cycle flow along at most $|A|$ directed cycles.

1.5.3 The Multicommodity Minimum-Cost Flow Problem

Minimum-cost flows are always considered for a single commodity. Therefore, those problems are also called **single-commodity** flow problems. It is also possible to consider flow problems with different commodities, called **multicommodity** flow problems. For example, several vehicles, each defining a commodity, share the same or parts of the same network and each is governed by its own flow conservation constraints. If there is no interaction between the different commodities, the multicommodity flow problem decomposes into independent single-commodity flow problems. In general, however, the commodities share common resources and facilities (e. g., common flow capacities as in our case) and do interact such that a decomposition is impossible.

We shall give a formulation of the multicommodity flow problem that is tailor-made for our purposes. We are given a digraph $D = (V, A)$ and K commodities denoted by $1, 2, \ldots, K$. For each commodity d there is an arc set $A_d \subset A$ such that A is equal to the disjoint union $\dot{\bigcup}_d A_d$. Moreover, there are a linear cost function $c^d \in \mathbb{Q}^{A_d}$, upper bounds $u^d \in \mathbb{Q}_+^{A_d}$, and node imbalances $b^d \in \mathbb{Q}^V$ (such that $\mathbb{1}^{\mathrm{T}} b^d = 0$). A flow vector $x^d \in \mathbb{Q}^{A_d}$ is associated with each commodity $d \in \{1, \ldots, K\}$ such that each x^d is a feasible flow, i. e., x^d satisfies (1.16b) and (1.16c) with respect to b^d and u^d. The flow vector for A is denoted by $x := (x^d)_{d=1,\ldots,K}$. Individual lower bounds $l^d \in \mathbb{Q}^{A_d}$ can also be considered by a transformation for each commodity as described in Fig. 1.1.

There are common flow capacities defined as follows: For each two nodes $i, j \in V$, let l_{ij} and u_{ij} denote the common lower and upper bound for the total flow of all commodities from node i to node j, i. e.,

$$l_{ij} \leqslant x\big(\{i\} \to \{j\}\big) \leqslant u_{ij}. \tag{1.22}$$

l_{ij} and u_{ij} are set to zero if there exist no arc $(i,j) \in A$.

Let $D_d := (V, A_d)$ denote the digraph of commodity d. The multicommodity minimum-cost flow problem reads

$$\min \sum_{d=1}^{K} \sum_{a \in A_d} c_a^d x_a^d \tag{1.23a}$$

subject to

$$\begin{array}{lrclcl}
(1.23\text{b}) & l_{ij} & \leqslant & x\big(\{i\} \to \{j\}\big) & \leqslant & u_{ij}, \quad \forall\, i \in V \;\; \forall\, j \in V, \\
(1.23\text{c}) & x^d(\delta^+(i)) & - & x^d(\delta^-(i)) & = & b_i, \quad \forall\, i \in V, \\
(1.23\text{d}) & 0 & \leqslant & x_a^d & \leqslant & u_a^d, \quad \forall\, a \in A_d \;\; \forall\, d \in \{1, \ldots, K\}.
\end{array}$$

The integrality theorem for optimal minimum-cost flows does no longer hold for multicommodity flows: Consider three different commodities a, b, and c, each defined on a copy of the digraph from Fig. 1.2 consisting of 3 nodes and 6 arcs. For each commodity, the arc costs are set to 0 for the arcs $\big\{(1,3); (3,2); (2,1)\big\}$ and are set to $M > 0$ for the arcs $\big\{(1,2); (2,3); (3,1)\big\}$. The individual and common arc upper bounds are set to 1, i.e., u, u^a, u^b, and u^c are all set to $\mathbb{1}$. The task is to send for each commodity one unit of flow as follows: for commodity a from node 1 to 2, for commodity b from node 2 to 3, and for commodity c from node 3 to 1. It is easy to check that the optimal solution value is $\frac{3M}{2}$, and each commodity sends half a unit of flow on one of the arcs having cost M and half a unit of flow on the path that includes its transshipment node and has zero costs.

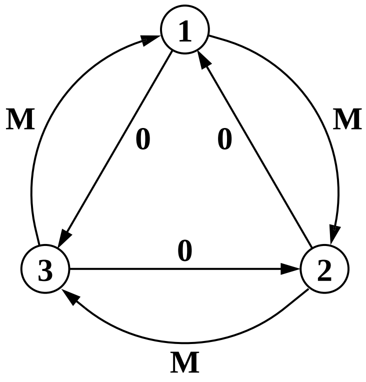


Figure 1.2: Multicommodity minimum-cost flow digraph that yields a fractional optimal solution.

Chapter 2

Multiple-Depot Vehicle Scheduling

2.1 Problem Description

We will now give a formal description of the Multiple-Depot Vehicle Scheduling Problem (MDVSP) in public mass transportation. Our terminology follows Hartley [1981] to a large extent.

A **garage** (or maintenance and storage facility) is a location where vehicles are parked and serviced. The term **fleet** denotes the set of vehicles of a transportation company. The fleet is divided among the garages, and it is known which vehicle types and how many vehicles of each type are stationed at each garage. A **depot** is a nonempty set of vehicles that need not be distinguished for the scheduling process. The set of all depots is denoted by $\mathcal{D}$. With each depot $d \in \mathcal{D}$, we associate a start point d^+ and an end point d^- where its vehicles begin and terminate their daily scheduled run or duty. Let

$$\mathcal{D}^- := \{d^- \mid d \in \mathcal{D}\} \quad \text{and} \quad \mathcal{D}^+ := \{d^+ \mid d \in \mathcal{D}\}$$

denote the set of all such start and end points, respectively. It is the task of the operator to subdivide the fleet into an appropriate set of depots. It is possible to define a depot for each individual vehicle or to define only one depot for all vehicles. In general, however, depots do not contain vehicles of different garages. Typically, all vehicles of the same type stationed at the same garage are combined in a depot.

We assume that the lines have been defined and their service frequencies have been chosen, i. e., a **timetable** has been determined. This timetable defines a set of so-called **timetabled trips** (or **passenger trips**), denoted by $\mathcal{T}$. We associate with each trip $t \in \mathcal{T}$ a first stop t^- together with a departure time s_t and a last stop t^+ together with an arrival time e_t. Let

$$\mathcal{T}^- := \{t^- \mid t \in \mathcal{T}\} \quad \text{and} \quad \mathcal{T}^+ := \{t^+ \mid t \in \mathcal{T}\}$$

denote the set of all first and last stops, respectively. For each individual trip $t \in \mathcal{T}$, there is given a nonempty set of *valid* depots, which we call **depot group** of t and denote by $G(t) \subseteq \mathcal{D}$. Only the vehicles of the trip's depot group are allowed to service a trip t. Such restrictions are necessary if, e. g., a double-decker bus would be too high for an underpass, an articulated bus would be too long for some narrow bend, the vehicle's capacity would not meet the trip's demand, etc. Normally, all timetabled trips of a line have the same basic depot group, but within the peak hours, depot groups are enlarged by further depots. Let

$$\mathcal{T}_d := \{t \in \mathcal{T} \mid d \in G(t)\}$$

denote all those timetabled trips that can be serviced by the vehicles of depot d and

$$\mathcal{T}_d^- := \{t^- \mid t \in \mathcal{T}_d\} \quad \text{and} \quad \mathcal{T}_d^+ := \{t^+ \mid t \in \mathcal{T}_d\}$$

the set of all first and last stops thereof.

Figure 2.1 shows three different lines in the plane. Each line is serviced in both directions. Figure 2.2 illustrates a small timetable with five chronologically ordered trips on these three lines. Each trip is uniquely assigned to some line as given in Fig. 2.3. In this figure, the lines and their service frequencies are combined to a set of timetabled trips. The depot group of each timetabled trip is shown by the different colours of the first and last stops.

There are further types of trips, which do not carry passengers. These trips are used to link timetabled trips: A **pull-out trip** connects a start point d^+ with a first stop t^-, a **pull-in trip** connects a last stop t^+ with an end point d^-, and a **dead-head trip** (or **dead running trip**) connects a last stop t^+ with a succeeding first stop t'^-. For notational simplicity, we call them all **unloaded trips**.

Many publications on vehicle scheduling abbreviate "timetabled trip" by "trip". We will not follow this use to avoid possible misunderstanding between timetabled trips and unloaded trips. Thus, whenever there is a danger of confusion, we explicitly use timetabled trips and unloaded trips, respectively.

For two timetabled trips t and $t' \in \mathcal{T}$, let $\Delta_{t,t'} \geqslant 0$ be given. In the literature, $\Delta_{t,t'}$ denotes the duration (travel plus layover time) from the location of the last stop of t to the location of the first stop of t', e. g., see Daduna and Paixão [1995], Dell'Amico, Fischetti, and Toth [1993], and Ribeiro and Soumis [1994]. However, our operating partners use such a definition of $\Delta_{t,t'}$ only for those dead-head trips for which the idle time or the difference $s_{t'} - e_t$ does not exceed a predefined maximum ranging from 40 to 120 minutes. Otherwise, $\Delta_{t,t'}$ is set to infinity. Dead-head trips exceeding this maximum are currently not considered for various reasons, e. g, the driver's idle time or break would become too long etc. We will show that such a restriction in the degree of freedom can lead to a higher vehicle demand and, therefore, to suboptimal solutions. To make it possible to use such links in spite of this, we set $\Delta_{t,t'} := s_{t'} - e_t$ whenever it is possible to park a vehicle between t and t' at the depot. We call these special dead-head trips also **pull-in-pull-out trips**. Our pull-in-pull-out trips are never given by the user and must be implicitly

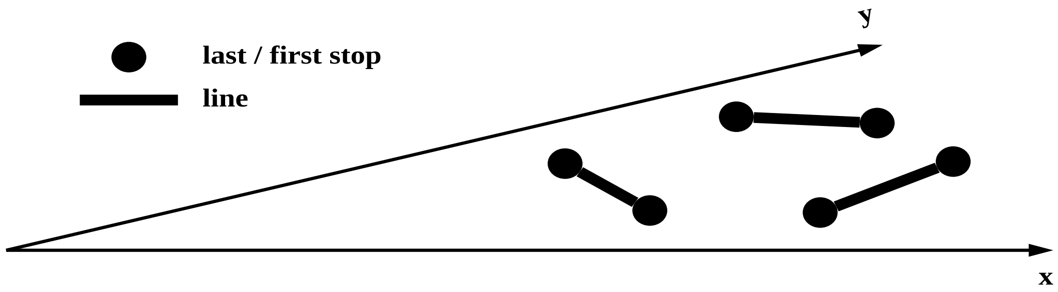


Figure 2.1: Lines.

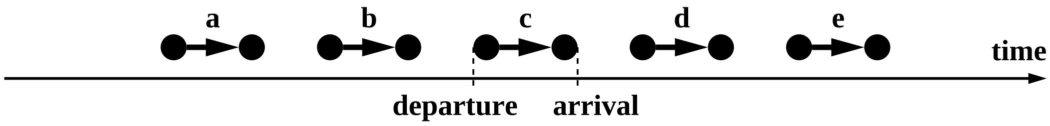


Figure 2.2: Timetable.

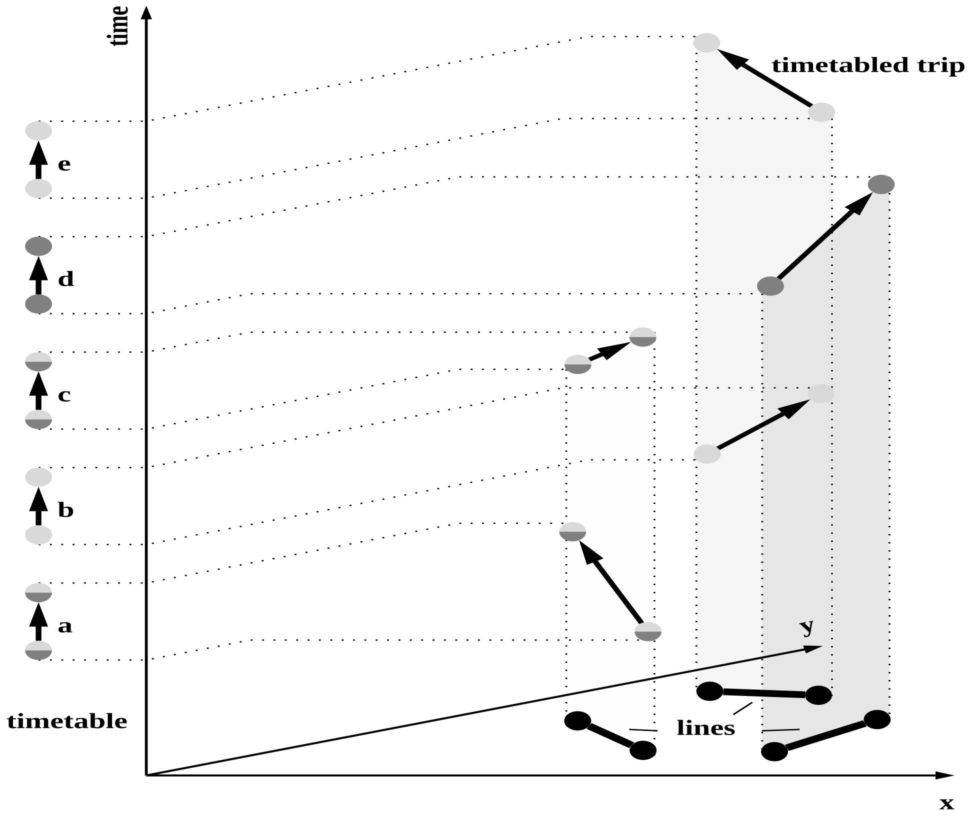


Figure 2.3: Timetabled trips for two depots.

defined by our system. We therefore call the *short* and explicitly defined dead-head trips also **user-defined** dead-head trips. When we talk about user-defined unloaded trips, we mean user-defined dead-head trips and pull-out and pull-in trips.

Whenever $e_t + \Delta_{t,t'} \leqslant s_{t'}$ is satisfied, the corresponding dead-head trip is called **compatible**. Pull-in and pull-out trips are always considered to be compatible, i. e., we assume that for each $d \in \mathcal{D}$ and each $t \in \mathcal{T}_d$ the pull-out and pull-in trip exist and can be used. Figure 2.4 illustrates possible unloaded trips for our small example.

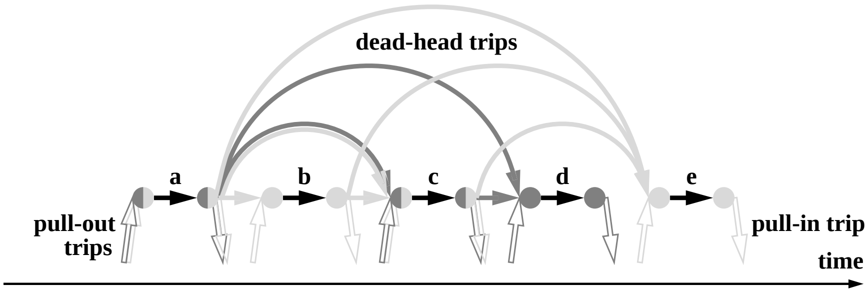


Figure 2.4: Unloaded trips.

The unloaded trips are used to interconnect the timetabled trips to **vehicle schedules**, each being a chain of trips such that the first trip is a pull-out trip, the last trip is a pull-in trip, and the timetabled and unloaded trips occur alternately. A vehicle schedule is feasible if there exists a depot that can service all its trips. All trips of a vehicle schedule have to be serviced by the same vehicle. If no dead-head trip is a pull-in-pull-out trip, the vehicle schedule is also called a **block** (or a **rotation**). A block that contains only one single timetabled trip is also called a **tripper**. Figure 2.5 shows a solution with two vehicle schedules for the small example presented in the last four figures.

For each trip we consider depot specific **weights** (or **costs**) depending on the trip's type, duration, distance, etc. The weight of an unloaded trip may in addition depend on the idle time of the vehicle and the driver etc. The exact definition of the weights must be quantified properly depending on operational interests. The weight of a vehicle schedule is the sum of the weights of all its trips.

The task of the MDVSP is to provide a set of vehicle schedules, which cover each trip $t \in \mathcal{T}$ exactly once and minimize a given linear objective function. There are different objectives possible, strongly depending on operational interests: The main objective – especially of large companies – is to use as few vehicles as possible and, subordinate, with minimum operational costs among all minimal fleet solutions. The motivation of this objective is to reduce the large capital costs for investment and maintenance of vehicles; a permanent and significant reduction in the number of vehicles does also reduce related fix costs (e. g., for garages, maintenance plants, or drivers). Another objective – especially of smaller transportation companies – is to schedule a given fleet with minimum operational costs. In the following, we concentrate our investigations on the first objective function.

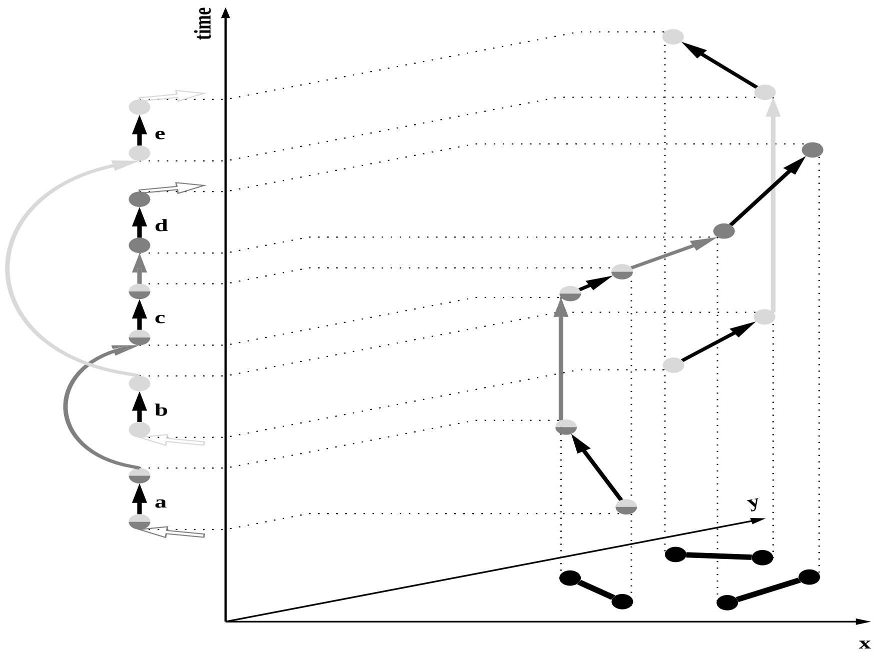


Figure 2.5: Two vehicle schedules.

2.2 Mathematical Models

We introduce a mathematical model that is based on a multicommodity flow formulation. For this formulation, we give an integer linear programming formulation.

An Integer Multicommodity Flow Model.

In the following, we define for each $d \in \mathcal{D}$ the following sets:

$$\begin{aligned}
A_d^{\text{t-trip}} &:= \{(t^-, t^+) |\; t \in \mathcal{T}_d\} && \text{(timetabled trips)},\\
A_d^{\text{pull-out}} &:= \{(d^+, t^-) |\; t \in \mathcal{T}_d\} && \text{(pull-out trips)},\\
A_d^{\text{pull-in}} &:= \{(t^+, d^-) |\; t \in \mathcal{T}_d\} && \text{(pull-in trips)},\\
A_d^{\text{d-trip}} &:= \{(p^+, q^-) |\; p, q \in \mathcal{T}_d \wedge e_p + \Delta_{p,q} \leqslant s_q\} && \text{(dead-head trips)},\\
A_d^{\text{u-trip}} &:= A_d^{\text{pull-out}} \cup A_d^{\text{pull-in}} \cup A_d^{\text{d-trip}} && \text{(unloaded trips)}.
\end{aligned}$$

Let (d^-, d^+) denote a **backward arc** from the depot's end to start point. Backward arcs are used for the vehicle return and to control depot capacities. Let

$$V_d' := \{d^+, d^-\} \cup \mathcal{T}_d^- \cup \mathcal{T}_d^+ \quad \text{and} \quad A_d' := A_d^{\text{t-trip}} \cup A_d^{\text{u-trip}} \cup \{(d^-, d^+)\}$$

denote the node and arc set, respectively, of the digraph $D'_d := (V'_d, A'_d)$. We define a large digraph

$$D' := (V', A') \tag{2.1}$$

with node set $V' := \mathcal{D}^+ \cup \mathcal{D}^- \cup \mathcal{T}^+ \cup \mathcal{T}^-$ and arc set $A' := \dot{\bigcup}_{d \in \mathcal{D}} A'_d$.

Figure 2.6 gives an illustration of D' for our small example from Sect. 2.1. The node set V' consists of the start and end points of all depots and the first and last stops of all timetabled trips. The arc set A' is the disjoint union of the arcs of all timetabled and unloaded trips. Note that the arc set A includes parallel arcs. Addressing an arc (t^-, t^+) without knowing its depot $d \in G(t)$ may lead to confusion. If necessary, we explicitly distinguish such arcs by their corresponding depots.

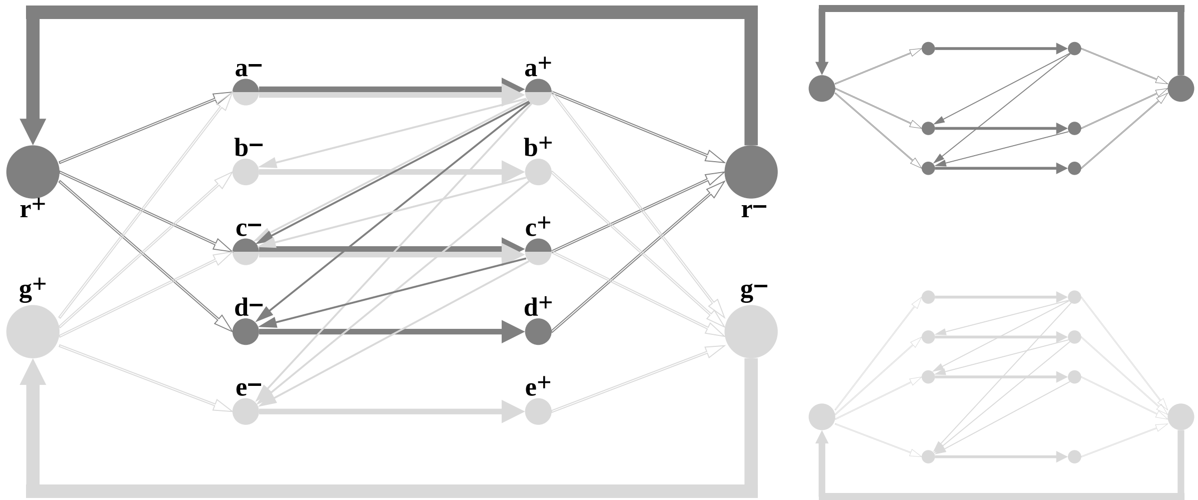


Figure 2.6: Digraphs (V', A') and (V'_d, A'_d), $d \in \mathcal{D}$, with $\mathcal{D} = \{$r,g$\}$ and $\mathcal{T} = \{$a,b,c,d,e$\}$.

An Integer Linear Programming Formulation.

We will now present an integer linear programming model for the MDVSP. For each $d \in \mathcal{D}$ and each $a \in A'_d$, we introduce an integer variable x^d_a that denotes a decision variable indicating whether a vehicle of depot d runs trip a or not, unless a denotes the backward arc. In this case, x^d_a counts all employed vehicles of the depot d. The variables x^d_a are combined into vectors $x^d := (x^d_a)_{a \in A'_d} \in \mathbb{R}^{A'_d}$, $d \in \mathcal{D}$, and these into $x := (x^d)_{d \in \mathcal{D}} \in \mathbb{R}^{A'}$.

Our two stage cost function is realized as follows: With each unloaded trip $a \in A^{\text{u-trip}}_d$, we associate a weight $c^d_a \in \mathbb{Q}$ representing its operational costs. In addition, we add to the weight of each pull-out trip a sufficiently large $\mathbf{M}$ representing the capital costs and being larger than the operational costs of any feasible solution. We deliberately do neither define weights for the timetabled trips nor for the backward arcs explicitly in our mathematical model, i. e., we assume them to be zero. If such weights are considered nevertheless, we apply the following simple transformation: Whenever a trip $t \in \mathcal{T}$ is serviced by the depot $d \in G(t)$, it is clear that exactly one unloaded trip of A_d incident from the last stop node t^+ must be used. Thus, we can easily shift the depot dependent costs of a trip $t \in \mathcal{T}_d$ to each arc of A_d that is incident from t^+. A similar transformation – shifting costs via the depot end point d^- – can be performed for nonzero costs of a backward arc.

For each $d \in \mathcal{D}$, let λ_d denote the depot lower and κ_d denote the depot upper capacity. These capacities define the (individual arc) lower and upper bounds of the backward arc. The (individual) lower and upper bound of all the other arcs is set to zero and one, respectively. For each $t \in \mathcal{T}$, the common lower and upper bound of the arcs going from the first stop node t^- to the last stop node t^+ are both set to 1. All the other arcs are not restricted by a common flow capacity, i. e., all these common lower and upper capacities are set to sufficiently small and large numbers, respectively, such that conditions (1.23b) are never tight and can therefore be neglected.

To describe a feasible vehicle schedule (and multicommodity flow vector, respectively), an integer vector $x \in \mathbb{R}^{A'}$ must satisfy the conditions from (1.23):

1. The common lower and upper capacities (1.23b), both together define the equations

$$\sum_{d \in G(t)} x^d_{(t^-,t^+)} = 1, \qquad \forall\, t \in \mathcal{T}. \tag{2.2}$$

2. The flow conservation constraints (1.23c) defining

$$x^d\big(\delta^+_{D'}(v)\big) - x^d\big(\delta^-_{D'}(v)\big) = 0, \qquad \forall\, v \in V'_d \;\; \forall\, d \in \mathcal{D}, \tag{2.3}$$

   which in detail read

$$\begin{aligned}
&(2.3a) && x^d\big(\delta^+_{D'}(t^+)\big) - x^d_{(t^-,t^+)} &&= 0, && \forall\, t \in \mathcal{T}_d \;\; \forall\, d \in \mathcal{D}.\\
&(2.3b) && x^d_{(t^-,t^+)} - x^d\big(\delta^-_{D'}(t^-)\big) &&= 0, && \forall\, t \in \mathcal{T}_d \;\; \forall\, d \in \mathcal{D},\\
&(2.3c) && x^d\big(\delta^+_{D'}(d^+)\big) - x^d_{(d^-,d^+)} &&= 0, && \forall\, d \in \mathcal{D},\\
&(2.3d) && x^d_{(d^-,d^+)} - x^d\big(\delta^-_{D'}(d^-)\big) &&= 0, && \forall\, d \in \mathcal{D}.
\end{aligned}$$

3. The individual flow capacities (1.23d) defining

$$\lambda_d \;\leqslant\; x^d_{(d^-,d^+)} \;\leqslant\; \kappa_d, \qquad \forall\, d \in \mathcal{D}, \tag{2.5}$$

   and

$$0 \;\leqslant\; x^d_a \;\leqslant\; 1, \qquad \forall\, a \in A^{\text{t-trip}}_d \cup A^{\text{u-trip}}_d \;\; \forall\, d \in \mathcal{D}. \tag{2.6}$$

The equalities (2.2), the so-called **flow conditions**, ensure that each timetabled trip t is serviced exactly once. With respect to our multicommodity flow model, the ILP reads

$$\min \sum_{d\in\mathcal{D}} \sum_{a\in A_d^{\text{u-trip}}} c_a^d x_a^d \tag{2.7a}$$

subject to

$$\begin{array}{lrcl}
(2.7\text{b}) & \sum_{d\in G(t)} x^d_{(t^-,t^+)} & = & 1, \qquad \forall\, t\in\mathcal{T},\\
(2.7\text{c}) & x^d\big(\delta^+_{D'}(v)\big) - x^d\big(\delta^-_{D'}(v)\big) & = & 0, \qquad \forall\, v\in V'_d \;\; \forall\, d\in\mathcal{D},\\
(2.7\text{d}) & \lambda_d \;\leqslant\; x^d_{(d^-,d^+)} & \leqslant & \kappa_d, \qquad \forall\, d\in\mathcal{D},\\
(2.7\text{e}) & 0 \;\leqslant\; x^d_a & \leqslant & 1, \qquad \forall\, a\in A_d^{\text{t-trip}}\cup A_d^{\text{u-trip}} \;\; \forall\, d\in\mathcal{D},\\
(2.7\text{f}) & & & x \text{ integral.}
\end{array}$$

The ILP (2.7) is a special integer multicommodity minimum-cost flow problem. Considering every depot d on its own, the solution vector x^d describes vehicle duties that depart from the depot's start point, flow through the digraph (V'_d, A'_d), arrive at the depot's end point, and return on the backward arc (d^-, d^+) to the start point. It is easy to see that x^d is a **circulation** of the "commodity" d through (V'_d, A'_d). The difficulty is that the circulations x^d are connected by the flow conditions (2.2): Exactly one of the $|G(t)|$ parallel arcs (t^-, t^+) must be used by some circulation x^d, $d\in G(t)$.

From a computational point of view, the ILP (2.7) includes many redundant constraints that can be eliminated by the following preprocessing steps:

1. If we consider a fixed $t\in\mathcal{T}$ and insert all flow conservation constraints from (2.3a) into the flow condition from (2.2), we get new (equivalent) flow conditions

$$\sum_{d\in G(t)} x^d\big(\delta^+_{D'}(t^+)\big) \quad = \quad 1, \qquad \forall\, t\in\mathcal{T},$$

   which, because $x^d\big(\delta^+_{D'}(t^+)\big) = x^d\big(\emptyset\big) = 0$ for all $d\notin G(t)$, is equivalent to

$$\sum_{d\in\mathcal{D}} x^d\big(\delta^+_{D'}(t^+)\big) \quad = \quad 1, \qquad \forall\, t\in\mathcal{T},$$

   or

$$x\big(\delta^+_{D'}(t^+)\big) \quad = \quad 1, \qquad \forall\, t\in\mathcal{T}. \tag{2.8}$$

2. Combining (2.3a) and (2.3b) for each fixed $d\in\mathcal{D}$ and each fixed $t\in\mathcal{T}_d$ yields new equations

$$x^d\big(\delta^+_{D'}(t^+)\big) - x^d\big(\delta^-_{D'}(t^-)\big) \quad = \quad 0, \qquad \forall\, t\in\mathcal{T}_d \;\; \forall\, d\in\mathcal{D}. \tag{2.9}$$

Substituting the flow conditions (2.2) by the new ones (2.8) and the flow conservations (2.3a) by (2.9) implies that the variables $x^d_{(t^-,t^+)}$ (and thus the equations (2.3b)) can be eliminated. In graph-theoretical terms, this variable elimination corresponds to a contraction of the two trip nodes t^- and t^+ to one single node, which we simply denote by t. Each arc of $A^{\text{t-trip}}_d$, for all $d \in \mathcal{D}$, will be removed; each arc incident to t^- gets the new head node t and, analogously, each arc incident from t^+ gets the new tail node t.

3. Consider for each fixed depot $d \in \mathcal{D}$ the flow conservation constraints (2.3c), (2.3d), and (2.9): The left-hand side of this system describes a node-arc incidence matrix of a network flow problem. It is well known (see, e. g., in Ahuja, Magnanti, and Orlin [1989], page 213) that each equation of such a system is linearly dependent on all the others. We can therefore eliminate the equation (2.3d) for each $d \in \mathcal{D}$. The variables $x^d_{(d^-,d^+)}$ are also unnecessary and can be eliminated by inserting the equation (2.3c) into the appropriate bound constraints (2.5) of the backward arc $(d^-, d^+) \in A'_d$. As in the contraction described above for the timetabled trips, we contract the two depot nodes d^- and d^+ to one new node d and remove the backward arc (d^-, d^+) from the arc set A'_d.

With these variable eliminations, we get reduced digraphs

$$D := (V, A) \tag{2.10}$$

and $D_d := (V_d, A^{\text{u-trip}}_d)$, $d \in \mathcal{D}$, with a new arc set $A := \dot{\bigcup}_{d\in\mathcal{D}} A^{\text{u-trip}}_d$ and new node sets

$$V_d := \{d\} \cup \mathcal{T}_d, \quad d \in \mathcal{D}, \qquad \text{and} \qquad V := \mathcal{D} \cup \mathcal{T}.$$

The contracted digraph (V, A) is displayed in Fig. 2.7. We rewrite (2.8) and (2.9) with respect to D as

$$x\big(\delta^+(t)\big) = 1, \qquad \forall\, t \in \mathcal{T}, \tag{2.11}$$

and

$$x^d\big(\delta^+(t)\big) - x^d\big(\delta^-(t)\big) = 0, \qquad \forall\, t \in \mathcal{T}_d \;\; \forall\, d \in \mathcal{D}. \tag{2.12}$$

The ILP (2.7) reduces with respect to $D = (V, A)$ to

$$\min \sum_{d\in\mathcal{D}} \sum_{a \in A^{\text{u-trip}}_d} c^d_a\, x^d_a \tag{2.13a}$$

subject to

$$
\begin{array}{lrcll}
(2.13\text{b}) & x\big(\delta^+(t)\big) & = & 1, & \forall\, t \in \mathcal{T},\\
(2.13\text{c}) & x^d\big(\delta^+(t)\big) - x^d\big(\delta^-(t)\big) & = & 0, & \forall\, t \in \mathcal{T}_d \;\; \forall\, d \in \mathcal{D},\\
(2.13\text{d}) & \lambda_d \;\leqslant\; x^d\big(\delta^+(d)\big) & \leqslant & \kappa_d, & \forall\, d \in \mathcal{D},\\
(2.13\text{e}) & 0 \;\leqslant\; x^d_a & \leqslant & 1, & \forall\, a \in A^{\text{u-trip}}_d \;\; \forall\, d \in \mathcal{D},\\
(2.13\text{f}) & x & \in & \{0,1\}^A. &
\end{array}
$$

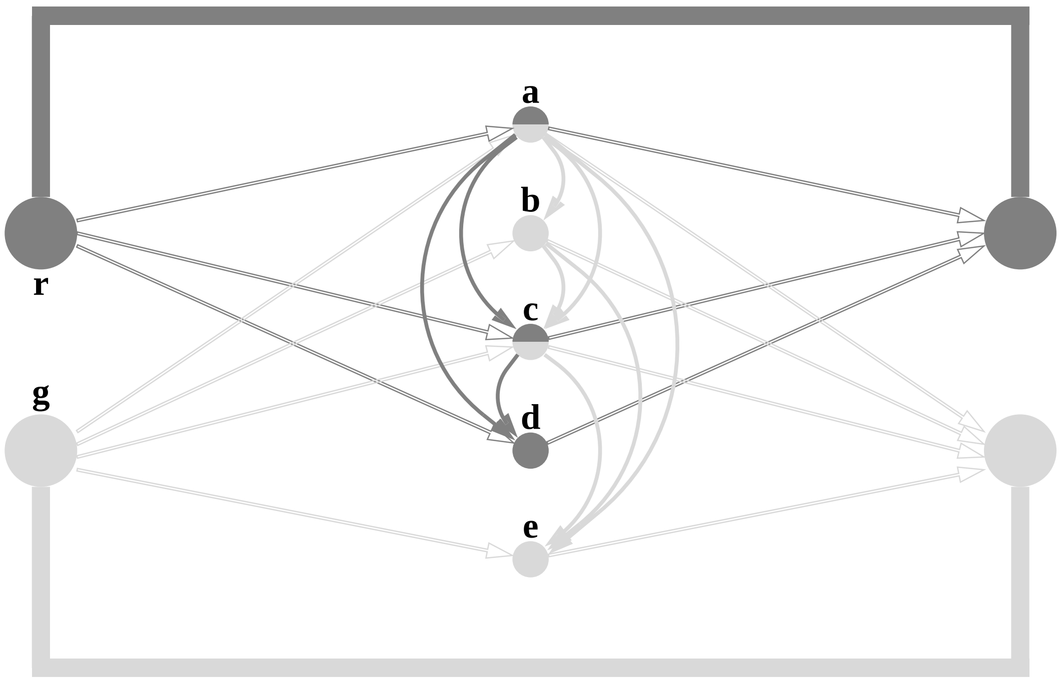


Figure 2.7: Contracted digraph $D = (V, A)$.

A last simplification can be obtained from (2.11):

$$1 \quad = \quad x\big(\delta^+(t)\big) \quad \geqslant \quad x_a^d, \qquad \forall\, a \in \delta^+(t) \;\; \forall\, d \in \mathcal{D}.$$

A similar result holds for $x\big(\delta^-(t)\big) = 1$, which is a linear combination of equations from (2.11) and (2.12). Thus, the upper bound constraint from (2.6) is, for each arc $a \in A$, a linear combination of (2.11) and (2.12) plus an affine combination of $-x \leqslant 0$, i. e., the upper bounds are redundant and can be neglected. The final ILP formulation reads:

(2.14a) $$\min \sum_{d \in \mathcal{D}} \sum_{a \in A_d^{\text{u-trip}}} c_a^d\, x_a^d$$

subject to

(2.14b) $$x\big(\delta^+(t)\big) = 1, \qquad \forall\, t \in \mathcal{T},$$
(2.14c) $$x^d\big(\delta^+(t)\big) - x^d\big(\delta^-(t)\big) = 0, \qquad \forall\, t \in \mathcal{T}_d \;\; \forall\, d \in \mathcal{D},$$
(2.14d) $$\lambda_d \leqslant x^d\big(\delta^+(d)\big) \leqslant \kappa_d, \qquad \forall\, d \in \mathcal{D},$$
(2.14e) $$x \geqslant 0,$$
(2.14f) $$x \in \{0,1\}^A.$$

(2.15) Remark. A feasible vector x of (2.14) may be considered as the incidence vector of some feasible solution.

(2.16) Lemma. The linear system (2.11) and (2.12) is nonredundant.

Proof: We assume that $\mathcal{D} := \{1, \ldots, |\mathcal{D}|\}$. Since all pull-out and pull-in trips exist per definition, we can consider the submatrix that is defined by all columns of

$$\bigcup_{t \in \mathcal{T}} \{(t,d) \in A_d^{\text{pull-out}} | \, d = \min G(t)\} \cup \bigcup_{d \in \mathcal{D}} \{(d,t) \in A_d^{\text{pull-in}} | \, t \in \mathcal{T}_d\}.$$

It is easy to check that, arranging the columns in the right way, this submatrix is lower triangular and nonsingular, which implies the full rank of the system. ❏

2.3 Discussion of the Model

Applied to vehicle scheduling problems from practice, the integer linear programming formulation (2.14) leads to ILPs with up to 70 million integer variables and 125 thousand constraints. In this sections, we will distinguish our multicommodity flow formulation from some other models that have been presented in the literature.

Practice-oriented solution methods are in most cases based on a single-commodity minimum-cost flow relaxation within a **schedule first – cluster second** (SF-CS) approach, see Chap. 8 or Daduna and Paixão [1995] for a detailed description of this approach. This means that the multiple-depot formulation is reduced to a single-depot relaxation. Unlike multicommodity flow formulations, however, those single-depot relaxation approaches have two significant drawbacks:

Depot groups and flow conservation: It is only possible to consider a single (depot independent) dead-head trip – we better call it *link* – between two timetabled trips. Such a link (t, t') is considered to be feasible with respect to the depot groups if $G(t) \cap G(t') \neq \emptyset$. But if depot groups must only be satisfied locally between two trips, the intersection of the depot groups of a generated vehicle schedule (or block) may be empty. In other words, the solution would be infeasible, see, e. g., Fig. 2.8. Splitting such infeasible block into its feasible parts can lead to suboptimal solutions.

Figure 2.8: Invalid block.

To avoid falling in such traps, the MDVSP should be modelled as a (special) multicommodity flow problem. Many research groups have considered the MDVSP as a multicommodity flow problem long before we started our investigations. The requirement of many real-world applications to consider different depot groups, however, was just realized in the last years, e. g., by Lamatsch [1988] or Forbes, Holt, and Watts [1994]. Obviously, multicommodity flow formulations are natural for this kind of scheduling problems. This is also reflected by the fact that we independently came up with the same ILP model as Forbes, Holt, and Watts [1994].

Limited duration for dead-head trips: It is often the case that single-depot relaxations consider dead-head trips with a limited duration, see, e. g., Daduna and Mojsilovic [1988] and Daduna, Mojsilovic, and Schütze [1993]. It is therefore only possible to generate blocks that must be linked to vehicle schedules in a succeeding step. Based on heuristic ideas, the main objective is to use as many dead-head trips as possible and, subordinate, to minimize operational costs. Obviously, this objective function does indeed minimize the number of blocks if depot groups are handled correctly. At the same time, it is assumed that a block minimal solution provides also a minimal fleet solution. It can be shown, however, that this is not true in general, see Figs. 2.9 and 2.10. The blocks, which have been determined by this strategy are subdivided to the depots and depot-wise linked to vehicle schedules. These links correspond to pull-in-pull-out trips. It is clear that such a problem decomposition into two successive steps can lead to suboptimal solutions.

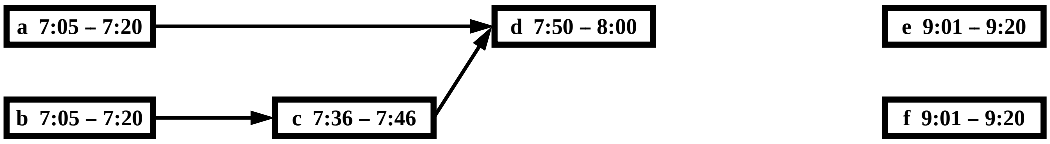


Figure 2.9: A single-depot instance (using limited durations for dead-head trips) for which a block minimal solution does not provide a minimal fleet solution.

Figure 2.9 displays such a single-depot instance: The maximum allowed duration of a dead-head trip is set to 60 minutes such that only the displayed dead-head trips (arrows) can be used. If we assume that the operational weight of (a, d) is significantly smaller than the weight of (c, d), it is easy to check that the blocks including the trips $\{a, d\}$, $\{b, c\}$, $\{e\}$, and $\{f\}$ define the unique block minimal solution with minimum operational weight. If we further assume that it is not possible to use a pull-in-pull-out trip to link "c" or "d" with "e" or "f", the block minimal solution requires four vehicle schedules. A fleet minimal solution, however, needs three vehicle schedules (e. g., including the timetabled trips $\{a, e\}$, $\{b, f\}$, and $\{c, d\}$), but has five blocks (including the timetabled trips $\{a\}$, $\{b\}$, $\{c, d\}$, $\{e\}$, and $\{f\}$).

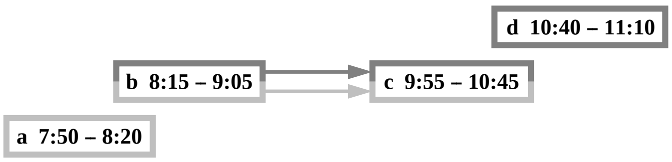


Figure 2.10: A multiple-depot instance (using limited durations for dead-head trips) for which a block minimal solution does not provide a minimal fleet solution.

Figure 2.10 displays a multiple-depot instance with two depots for which the minimal fleet solution cannot be obtained with a block minimal solution: The first depot can service trips "b, "c", and "d", and the second depot can service "a", "b", and "c". Two timetabled trips may be linked by a pull-in-pull-out trip if the depot groups are satisfied and if the two timetabled trips do not overlap. The maximum allowed duration of a dead-head trip is set to 60 minutes such that for both depots just the dead-head trip between "b" and "c" is possible. The block minimal number is three ("d" is assigned to the first depot, "a" is assigned to the second depot, and "b→c" is assigned either to the first or to the second depot) and requires three vehicles, but two vehicles are optimal ({a,c} and {b,d}) if each timetabled trip defines its own block.

We have shown in Figs. 2.9 and 2.10 that it is insufficient to generate a minimal fleet solution in such a two step approach. Linking blocks optimally and selecting user-defined unloaded trips must be done simultaneously. Pull-in-pull-out trips translate the decision of linking blocks into the terminology of dead-head trips. Therefore, using pull-in-pull-out trips makes it possible to generate a minimal fleet solution with minimum operational costs in one step.

Each pull-in-pull-out trip stands for a pull-in trip followed by a pull-out trip. The set of all pull-in-pull-out trips represents all feasible possibilities to link blocks to vehicle schedules. If we enlarge the user-defined unloaded trips by the pull-in-pull-out trips, the number of necessary vehicles is nothing but the number of used pull-out trips (or, equivalently, pull-in trips). Vice versa, if we replace each pull-in-pull-out trip of a vehicle schedule by the corresponding pull-in and pull-out trip, it is always possible to assign all resulting blocks of this vehicle schedule to a single vehicle.

The concept of pull-in-pull-out trips was first described by Bokinge and Hasselström [1980] and Desrochers, Desrosiers, and Soumis [1985]. Since the number of pull-in-pull-out trips grows in the order of $\#\mathcal{D} \times (\#\mathcal{T})^2$, they claimed that numerical investigations with pull-in-pull-out trips are unacceptable for the computers they used. Such a problem formulation needed too much main memory at this time. Therefore, Bokinge and Hasselström [1980] and Desrochers, Desrosiers, and Soumis [1985] give an alternative formulation for the single-depot case without pull-in-pull-out trips. Lamatsch [1988,1992] extends this formulation to the multiple-depot case. Compared to our formulation without pull-in-pull-out trips, his formulation results in a mathematical model with $\mathcal{O}(\#\mathcal{D} \times \#\mathcal{T})$ additional flow conservation constraints and $\mathcal{O}(\#\mathcal{D} \times \#\mathcal{T})$ additional variables. At first glance, such a problem formulation with a linear instead of a quadratic number of variables seems to be advantageous. Nevertheless, we have decided in favour of the problem formulation with pull-in-pull-out trips resulting in a mathematical model with a quadratic number of variables. The reasons for our decision are:

1. For the alternative formulation, it is not obvious to tell the beginning of a simple block from the beginning of a new vehicle schedule, i. e., it is impossible to count the number of necessary vehicles easily. In our formulation, each used pull-out trip defines a new vehicle schedule, i. e., a further vehicle must be provided.
2. The model with pull-in-pull-out trips is more flexible since, from an operational point of view, the weight of a pull-in-pull-out trip can differ from the sum of the

weights of its pull-in and pull-out trip.

3. Real-world problems with one depot can be solved efficiently with the computer generation of our days, even if we use million pull-in-pull-out trips. Truly large-scale instances, which are solved as subproblems within our solution process for MDVSPs, with up to 25 thousand timetabled trips and 70 million unloaded trips can be optimally solved within some minutes (see Chap. 5).

4. Although the use of pull-in-pull-out trips increases the problem size enormously, especially the number of variables of an integer linear programming formulation, the number of constraints remains constant. This does not hold for the formulation of Lamatsch, whose ILP includes $\mathcal{O}(\#\mathcal{D} \times \#\mathcal{T})$ more flow conservation constraints. For our largest problem instance, however, this may induce about 200 thousand flow conservation constraints that must be added to the 125 thousand original flow conditions and flow conservations. Currently, there is no LP solver available that could solve systems of this scale – even with a reduced variable set.

We have seen that arc-oriented multicommodity flow formulations and their integer programming analogues are proper models for the MDVSP. With such formulations, it is possible to consider most practical requirements. Only restrictions on the length or the durations of blocks or vehicle schedules, however, cannot be handled efficiently with such arc-oriented formulations. Even the SDVSP is $\mathcal{NP}$-hard if such restrictions must be considered, see, e. g., Freling and Paixão [1995]. For those problems, we suggest to use path-oriented set partitioning models. We will differentiate arc-oriented models from path-oriented models in the next chapter and describe a set partitioning approach for the MDVSP in Chap. 11.

Chapter 3

The Complexity of the MDVSP

The complexity of an MDVSP instance depends on $|\mathcal{D}|$. The **single-depot case** ($|\mathcal{D}| = 1$) with up to several thousand trips can be solved efficiently with polynomial time minimum-cost flow algorithms. For the other cases ($|\mathcal{D}| \geqslant 2$), even the uncapacitated MDVSP is $\mathcal{NP}$-hard. Bertossi, Carraresi, and Gallo [1987] proved that the multicommodity matching relaxation of the MDVSP is already $\mathcal{NP}$-hard for problems with two depots, no depot capacities, and no depot groups. Their proof is based on a special satisfiability problem, the ONE-IN-THREE 3SAT with unnegated literals.

We will give a new proof for their result. This new proof is based on our multicommodity flow formulation. In addition, we show that the MDVSP is $\mathcal{NP}$-hard in the strong sense. We will also prove the new result that the feasibility version of the MDVSP with depot capacities is already $\mathcal{NP}$-complete.

From now on, "MDVSP" always refers to the multiple-depot case $|\mathcal{D}| \geqslant 2$, whereas the Single-Depot Vehicle Scheduling Problem (SDVSP) refers to the single-depot case $|\mathcal{D}| = 1$. In the following of this thesis, we will consider the MDVSP and explicitly mention it when we consider the SDVSP.

For our complexity investigations in this chapter, we will use the terminology introduced in Garey and Johnson [1979].

3.1 The Single-Depot Case

Consider a SDVSP assuming $\mathcal{D} := \{d\}$ and $G(t) = \mathcal{D}$, for all $t \in \mathcal{T}$. Figure 3.1 illustrates a single-depot case digraph $D' = (V', A')$.

For the SDVSP we can neglect the index d. The ILP (2.7) of the "singlecommodity" flow model with a detailed list of its flow conservations (2.3) reads:

$$\min \sum_{a \in A^{\text{u-trip}}} c_a \, x_a \tag{3.1a}$$

subject to

$$
\begin{array}{llcll}
(3.1\text{b}) & x_{(t^-,t^+)} & = & 1, & \forall\, t \in \mathcal{T}, \\
(3.1\text{c}) & x\big(\delta^+(t^+)\big) - x_{(t^-,t^+)} & = & 0, & \forall\, t \in \mathcal{T}, \\
(3.1\text{d}) & x_{(t^-,t^+)} - x\big(\delta^-(t^-)\big) & = & 0, & \forall\, t \in \mathcal{T}, \\
(3.1\text{e}) & x\big(\delta^+(d^-)\big) - x_{(d^-,d^+)} & = & 0, & \\
(3.1\text{f}) & x_{(d^-,d^+)} - x\big(\delta^-(d^-)\big) & = & 0, & \\
(3.1\text{g}) & \lambda \;\leqslant\; x_{(d^-,d^+)} & \leqslant & \kappa, & \\
(3.1\text{h}) & 0 \;\leqslant\; x_a & \leqslant & 1, & \forall\, a \in A^{\text{t-trip}} \cup A^{\text{u-trip}}, \\
(3.1\text{i}) & & & x \text{ integral.} &
\end{array}
$$

As before, we apply to (3.1) some preprocessing steps to get a more concise ILP formulation:

- One of the flow conservations (3.1c)–(3.1f) is redundant; we eliminate (3.1e).
- Inserting (3.1b) into (3.1c) and (3.1d) yields new equations

$$
\begin{array}{llcll}
(3.2) & x\big(\delta^+(t^+)\big) & = & +1, & \forall\, t \in \mathcal{T}, \\
(3.3) & -x\big(\delta^-(t^-)\big) & = & -1, & \forall\, t \in \mathcal{T},
\end{array}
$$

  and we can eliminate the equations and variables of (3.1b). Figure 3.2 shows the transformed digraph for all these variable eliminations.
- Each inequality $x_a \leqslant 1$ is a linear combination of some equation (3.2) or (3.3), respectively, plus an affine combination of $-x \leqslant 0$. We can neglect them.
- It is easy to check that the left-hand side of the remaining equation system (3.1f), (3.3), and (3.2) describes a nonredundant node-arc incidence matrix of a network flow problem. Since all bounds and the right-hand side are integral, the integrality condition (3.1i) is automatically satisfied for each basic solution.

The reduced ILP (3.1) reads

$$
(3.4\text{a}) \qquad \min \sum_{a \in A^{\text{u-trip}}} c_a\, x_a
$$

subject to

$$
\begin{array}{llcll}
(3.4\text{b}) & x\big(\delta^+(t^+)\big) & = & 1, & \forall\, t \in \mathcal{T}, \\
(3.4\text{c}) & -x\big(\delta^-(t^-)\big) & = & -1, & \forall\, t \in \mathcal{T}, \\
(3.4\text{d}) & x_{(d^-,d^+)} - x\big(\delta^-(d^-)\big) & = & 0, & \\
(3.4\text{e}) & \lambda \;\leqslant\; x_{(d^-,d^+)} & \leqslant & \kappa, & \\
(3.4\text{f}) & x_a & \geqslant & 0, & \forall\, a \in A^{\text{u-trip}},
\end{array}
$$

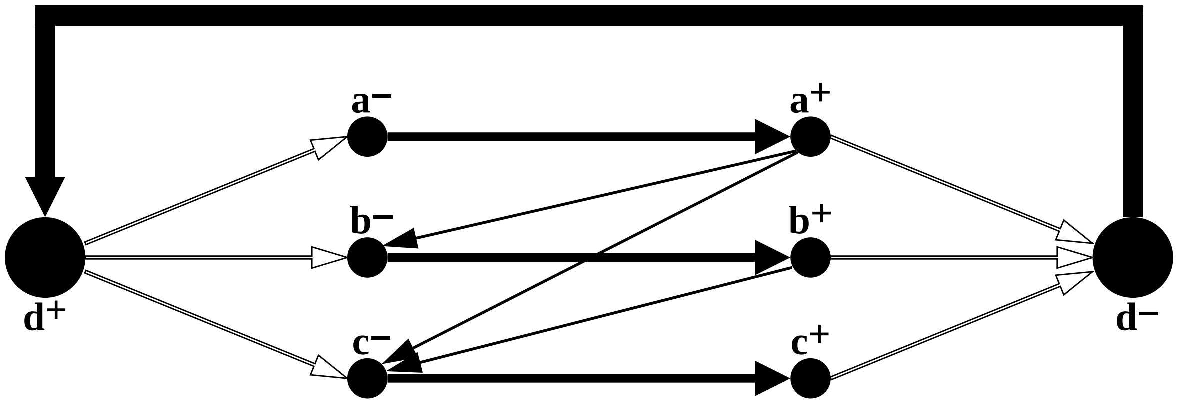


Figure 3.1: SDVSP digraph (V', A') with $\mathcal{T} = \{$a,b,c$\}$.

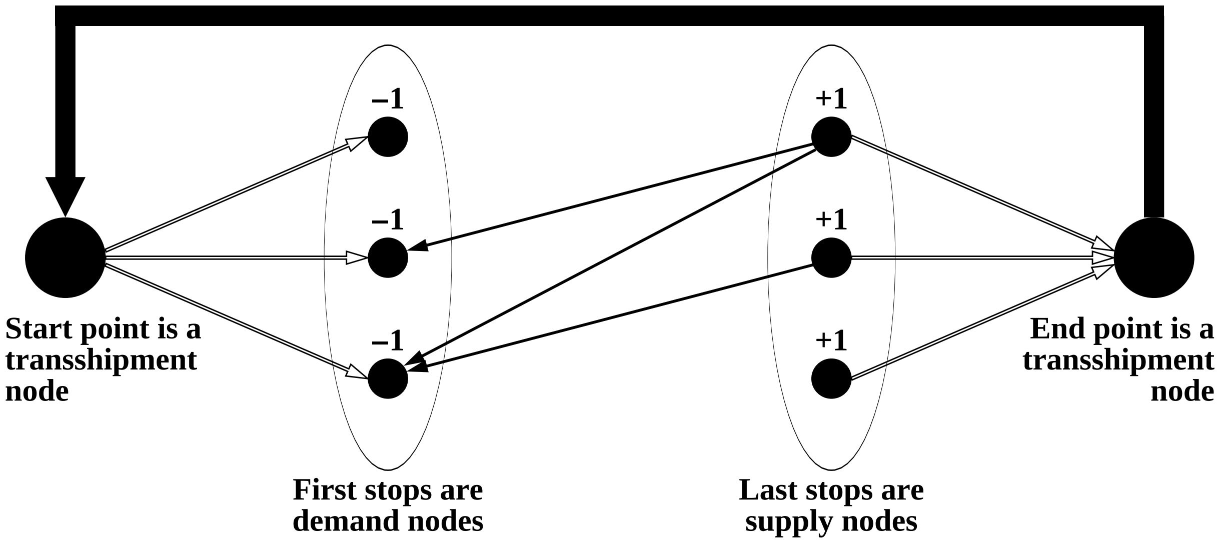


Figure 3.2: SDVSP digraph after variable elimination.

which is the LP formulation of a minimum-cost flow problem. The difference to the MDVSP is that the flow conditions (3.1b) in combination with the flow conservations (3.1c) and (3.1d) can be transformed to real node demands and supplies such that (3.1) reduces to a polynomially solvable minimum-cost flow problem.

(3.5) Theorem. The SDVSP can be solved in polynomial time.

Various polynomial and pseudo-polynomial time solution approaches for minimum-cost flow problems can be found, e. g., in Ahuja, Magnanti, and Orlin [1993]. We use a network simplex algorithm to solve the SDVSP. The implementation of this network simplex algorithm is described in Chap. 5.

3.2 The Multiple-Depot Case

To show the $\mathcal{NP}$-hardness of the MDVSP, we have to show the $\mathcal{NP}$-completeness of its **decision problem**, which is defined as follows: Given an instance (2.14) of the MDVSP and a number $L \in \mathbb{Q}$; is there a feasible solution x that satisfies all conditions of (2.14) and has an objective value not larger than L?

(3.6) Theorem. The MDVS-decision-P is $\mathcal{NP}$-complete in the strong sense.

Proof: See page 35. ❏

(3.7) Corollary. There exists no pseudo-polynomial time algorithm for the MDVSP unless $\mathcal{P} = \mathcal{NP}$.

Proof: Garey and Johnson [1979], Chap. 4, page 95. ❏

The answer to the question whether there exists a polynomial approximation scheme for the MDVSP is most probably "no":

(3.8) Theorem. The ε-approximation MDVSP is $\mathcal{NP}$-hard.

Proof: See page 39. ❏

(3.9) Remark. The objective function in the proof of Theorem 3.8 does not satisfy the triangle inequalities. It is still open whether there exists a polynomial time ε-approximation algorithm for the MDVSP when the triangle inequalities are satisfied. For practical applications, however, we do not believe that this has any relevance since the arc costs are seldom distances and usually do not satisfy the triangle inequalities.

Our above complexity investigations are focused on uncapacitated problem instances. For capacitated instances, however, it is not only $\mathcal{NP}$-hard to optimize, but already $\mathcal{NP}$-complete to find a feasible solution.

(3.10) Theorem. The feasibility problem of the capacitated MDVSP is $\mathcal{NP}$-complete.

Proof: See page 40. ❑

(3.11) Remark. Complexity theory tells us that it is $\mathcal{NP}$-hard to find a feasible solution if depot capacities are considered. However, it turned out that depot capacities are not always hard constraints and can thus sometimes be violated since vehicles can often be shifted from one garage (or depot) to another.

In the following, we give the proofs of the last three theorems. To prove the complexity results for the MDVSP, we use the **ONE-IN-THREE 3SAT** with unnegated literals, which we briefly write OIT-3SAT-UL. This special satisfiability problem is as follows: Given two integer numbers $q \geqslant 3$ and $p \geqslant 1$, a set $\mathfrak{U} = \{\mathfrak{u}_1, \ldots, \mathfrak{u}_q\}$ of Boolean variables, and a collection $\mathcal{C} = \{\mathfrak{C}_1, \ldots, \mathfrak{C}_p\}$ of clauses over $\mathfrak{U}$ such that each clause $\mathfrak{C} \in \mathcal{C}$ has $|\mathfrak{C}| = 3$ and that no $\mathfrak{C}$ contains a negated literal; is there a truth assignment $tr : \mathfrak{U} \longmapsto \{\mathrm{T}, \mathrm{F}\}$, T stands for "True" and F stands for "False", such that each clause $\mathfrak{C} \in \mathcal{C}$ has exactly one true literal? The OIT-3SAT-UL is known to be $\mathcal{NP}$-complete in the strong sense (see Garey and Johnson [1979]).

Proof of Theorem 3.6: We show that the theorem is already true for uncapacitated problems with two depots ($|\mathcal{D}| = 2$) and identical depot groups $G \equiv \mathcal{D}$ (the lower and the upper capacities are set to 0 and ∞, respectively). Given such a two depot instance of the MDVSP and a number $L \in \mathbb{Q}$. Obviously, the decision problem of the MDVSP is in $\mathcal{NP}$ since given a solution x, a nondeterministic algorithm can check all conditions of (2.14), compute the objective value $c^{\mathrm{T}}x$, and compare it with the given bound L in polynomial time. We transform the OIT-3SAT-UL to the decision problem of the MDVSP. Let $q \geqslant 3$, $p \geqslant 1$, $\mathfrak{U} = \{\mathfrak{u}_1, \ldots, \mathfrak{u}_q\}$, and $\mathcal{C} = \{\mathfrak{C}_1, \ldots, \mathfrak{C}_p\}$ make up an arbitrary instance of the OIT-3SAT-UL.

The construction of an MDVSP instance for the OIT-3SAT-UL instance will be based on a two depot problem: The first depot is denoted by **true** (T) and the second depot by **false** (F), i. e., $\mathcal{D} = \{\mathrm{T}, \mathrm{F}\}$. Let $\mathfrak{u}_0$ denote an artificial Boolean variable. Every Boolean variable and each literal of each clause define a timetabled trip, i. e.,

$$\mathcal{T}_{\mathrm{T}} := \mathcal{T}_{\mathrm{F}} := \mathcal{T} := \{\mathfrak{u}_0\} \cup \mathfrak{U} \cup \bigcup_{i=1}^{p} \mathfrak{C}_i.$$

The definition of the dead-head trips is slightly complicated. First, we introduce some help sets:

- $\mathcal{G}_0 := \mathfrak{U}$,
- $\mathcal{G}_i := \mathfrak{C}_i$, for all $i \in \{1, \ldots, p\}$, and
- $\mathcal{H}_j := \{\mathfrak{u}_j\} \cup \{\mathfrak{c} \in \bigcup_{i=1}^{p} \mathfrak{C}_i |\ \mathfrak{c}$ is a literal of $\mathfrak{u}_j\}$, for all $j \in \{1, \ldots, q\}$.

Let "$\prec_j$" denote an order on $\mathcal{H}_j$ such that for $\{x, y\} \subseteq \mathcal{H}_j$:

$$x \prec_j y \Longrightarrow \exists\, m < n : x \in \mathcal{G}_m \wedge y \in \mathcal{G}_n.$$

"$\prec_j$" is well-defined because each $\mathcal{H}_j$ contains at most one element from each $\mathcal{G}_i$. $y \in \mathcal{H}_j$ is called the successor "succ(x)" of $x \in \mathcal{H}_j$ if $x \prec_j y$ and if there exists no $z \in \mathcal{H}_j$ such that $x \prec_j z \prec_j y$. The arc sets of the dead-head trips are set to

$$A_{\mathrm{T}}^{\text{d-trip}} := \big(\{\mathfrak{u}_0\} \to \mathcal{G}_1\big) \cup \bigcup_{i=2}^{p} (\mathcal{G}_{i-1} \to \mathcal{G}_i)$$

and

$$A_{\mathrm{F}}^{\text{d-trip}} := \bigcup_{j=1}^{q} \Bigg(\bigcup_{\substack{t \in \mathcal{H}_j : \\ t \neq \max_{\prec_j} \mathcal{H}_j}} \big((t, \mathrm{succ}(t))\big) \Bigg).$$

The lower bounds λ_{T} and λ_{F} are set to 0, the upper bounds κ_{T} and κ_{F} are set to ∞. The costs of the following arcs are set to zero:

- all defined dead-head trips $A_{\mathrm{T}}^{\text{d-trip}}$ and $A_{\mathrm{F}}^{\text{d-trip}}$,
- the pull-out trips $\big(\{\mathrm{T}^+\} \to (\mathcal{G}_0 \cup \{\mathfrak{u}_0\})\big) \subset A_{\mathrm{T}}^{\text{pull-out}}$ and $\big(\{\mathrm{F}^+\} \to \mathcal{G}_0\big) \subset A_{\mathrm{F}}^{\text{pull-out}}$,
- the pull-in trips $\big((\mathcal{G}_0 \cup \mathcal{G}_p) \to \{\mathrm{T}^-\}\big) \subset A_{\mathrm{T}}^{\text{pull-in}}$ and $\big(\bigcup_{j=1}^{q} \{\max_{\prec_j} \mathcal{H}_j\} \to \{\mathrm{F}^-\}\big) \subset A_{\mathrm{F}}^{\text{pull-in}}$.

The costs of all the other arcs are set to 1. Finally, we set $L := 0$. The arcs, which can only be used by a feasible solution satisfying $c^{\mathrm{T}}x \leqslant 0 = L$, are those whose cost coefficients are 0, i. e.,

$$\text{if } (c^{\mathrm{T}}x = 0 \text{ and } x_a = 1) \text{ then } c_a = 0. \tag{3.12}$$

It is easy to see that the above transformation can be performed in $\mathcal{O}(p+q)$ time.

Figure 3.3 shows the digraph $D' = (V', A')$ of a transformed OIT-3SAT-UL problem instance with $\mathfrak{U} := \{\mathfrak{u}_1, \mathfrak{u}_2, \mathfrak{u}_3, \mathfrak{u}_4\}$ and $\mathfrak{C} := \big\{\{\mathfrak{u}_1, \mathfrak{u}_2, \mathfrak{u}_3\}, \{\mathfrak{u}_1, \mathfrak{u}_2, \mathfrak{u}_4\}\big\}$. Only the arcs satisfying $c_a = 0$ are displayed. The task is to find a truth assignment $tr : \mathfrak{U} \longmapsto \mathcal{D}$ satisfying $(\mathfrak{u}_1 \dot\vee \mathfrak{u}_2 \dot\vee \mathfrak{u}_3) \wedge (\mathfrak{u}_1 \dot\vee \mathfrak{u}_2 \dot\vee \mathfrak{u}_4)$.

To prove that the given transformation is indeed a transformation from OIT-3SAT-UL to MDVSP, we have to show that an instance of OIT-3SAT-UL is satisfiable if and only if the transformed problem is feasible.

Assume that we know a truth assignment satisfying the clauses of $\mathfrak{C}$. We will construct a feasible scheduling, whose incidence vector x is feasible. For each $\mathfrak{u}_j$ assigned to false, we define an F-vehicle schedule

$$\big(\mathrm{F}^+, \mathfrak{u}_j\big),\ \big(t, \mathrm{succ}(t)\big)\ \forall\, t \in \mathcal{H}_j : t \neq \max_{\prec_j} \mathcal{H}_j,\ \text{and } \big(\max_{\prec_j} \mathcal{H}_j, \mathrm{F}^-\big). \tag{3.13}$$

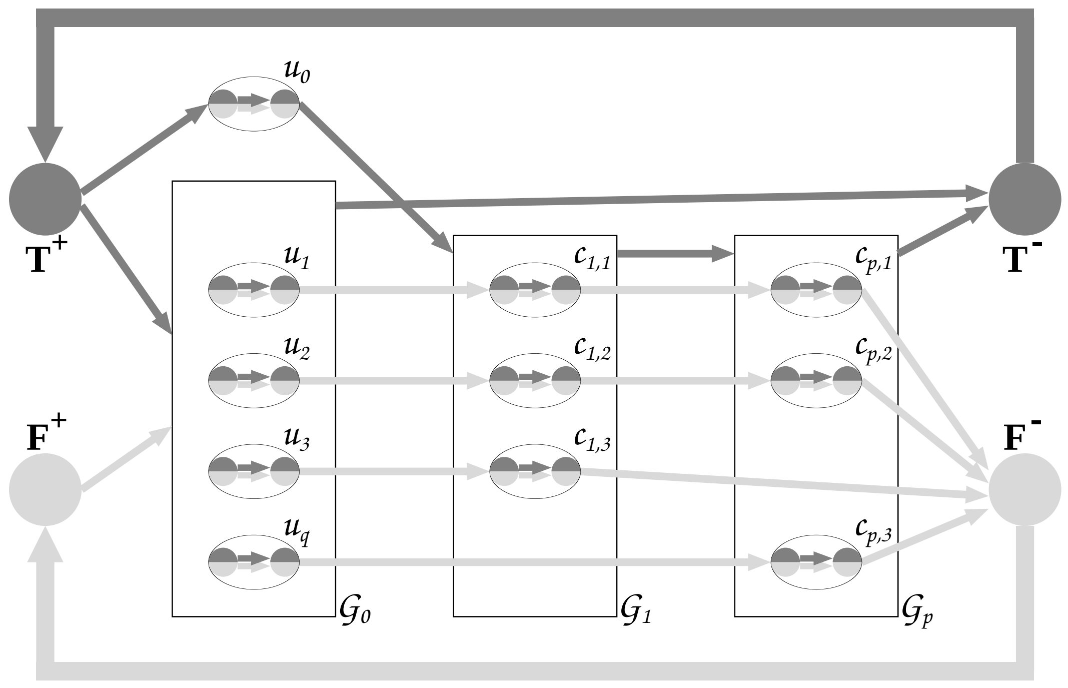


Figure 3.3: MDVSP digraph $D' = (V', A')$ of a OIT-3SAT-UL.

This covers all timetabled trips of $\bigcup_{j:\, tr(\mathfrak{u}_j)=\mathrm{F}} \mathcal{H}_j$. For each $\mathfrak{u}_j$ assigned to true, we define a T-vehicle schedule

$$(3.14) \qquad \big(\mathrm{T}^+, \mathfrak{u}_j\big) \quad \text{and} \quad \big(\mathfrak{u}_j, \mathrm{T}^-\big).$$

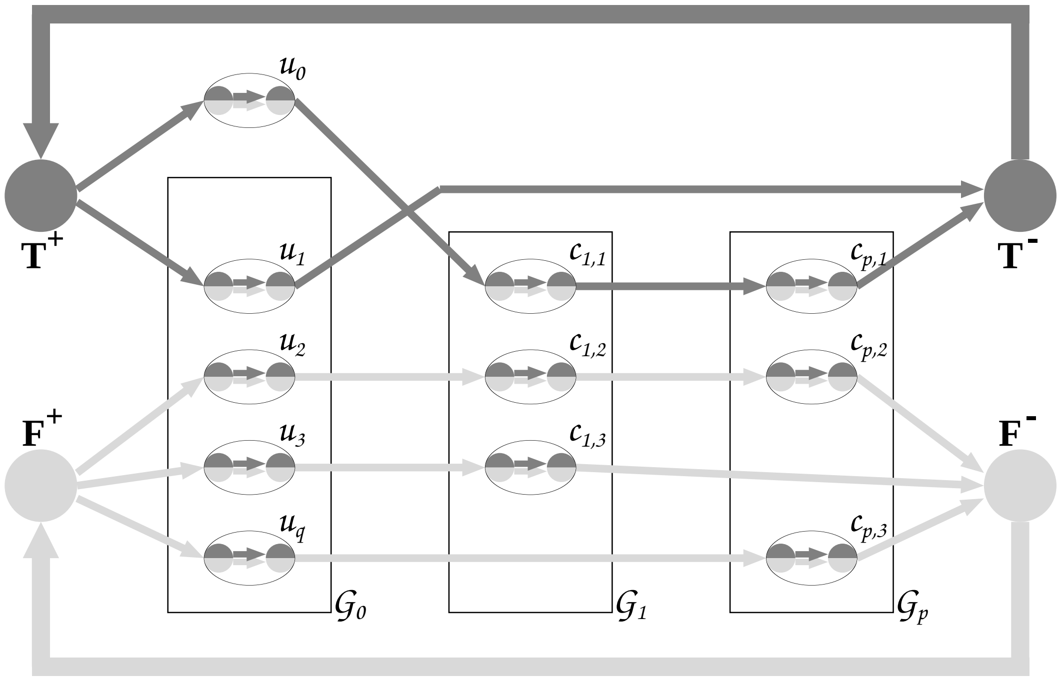
This covers all (by F-vehicle schedules uncovered) nodes of $\mathcal{G}_0$. The node $\mathfrak{u}_0$ and the "true" literal node of each clause group $\mathcal{G}_i$, $i = 1, \ldots, p$, are still uncovered. Let $\mathfrak{c}'_i \in \mathcal{G}_i$, $i = \{1, \ldots, p\}$, denote these uncovered literal nodes. To complete the scheduling, we define a last T-vehicle schedule

$$(3.15) \qquad \big(\mathrm{T}^+, \mathfrak{u}_0\big),\ \big(\mathfrak{u}_0, \mathfrak{c}'_1\big),\ \big(\mathfrak{c}'_1, \mathfrak{c}'_2\big),\ \ldots,\ \big(\mathfrak{c}'_{p-1}, \mathfrak{c}'_p\big),\ \text{and}\ \big(\mathfrak{c}'_p, \mathrm{T}^-\big).$$

Obviously, the constructed solution uses only defined dead-head trips and only unloaded trips with zero cost coefficients. It is easy to check that all conditions of (2.14) are satisfied and that the objective value does not exceed $L = 0$. Figure 3.4 shows the transformation of two feasible truth assignments for the example of Fig. 3.3; only arcs with $x_a = 1$, for all $a \in A^{\text{u-trip}}$, are displayed.

Conversely, consider a feasible solution x satisfying $c^{\mathrm{T}}x = 0$:

❶ Since x must follow (3.12), a feasible solution x can only use the pull-out trips $\big(\mathrm{T}^+ \to (\mathcal{G}_0 \cup \{\mathfrak{u}_0\})\big) \subset A^{\text{pull-out}}_{\mathrm{T}}$ and $(\mathrm{F}^+ \to \mathcal{G}_0, \mathrm{F}) \subset A^{\text{pull-out}}_{\mathrm{F}}$. Since $\delta^-(\mathfrak{u}_0) = \big(\mathrm{T}^+, \mathfrak{u}_0\big)$ and $\delta^-(t) = \big(\mathrm{T}^+, t\big) \cup \big(\mathrm{F}^+, t\big)$, for all $t \in \mathcal{G}_0$, each timetabled trip $t \in \mathcal{G}_0 \cup \{\mathfrak{u}_0\}$ must be



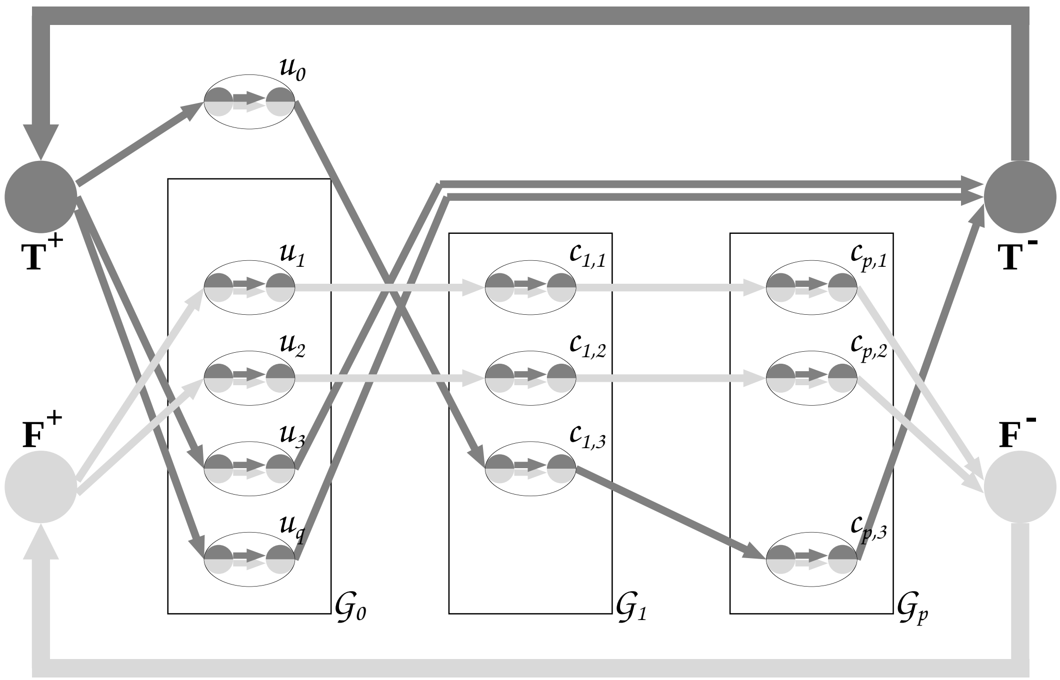


Figure 3.4: Two feasible truth assignments for the instance of Fig. 3.3.

serviced with some pull-out trip. Therefore, x must describe exactly $q+1$ vehicle schedules.

❷ In particular, $\mathfrak{u}_0$ can only be serviced by a vehicle schedule of T. Applying the flow conservation constraints (2.12), this T-vehicle schedule must also service exactly one literal node of each $\mathcal{G}_i$, $i \in \{1, \dots, p\}$, and is defined like (3.15). By the way, all these literal nodes serviced by this T-vehicle schedule will become the true literals of the truth assignment.

❸ $\delta^+(\mathfrak{u}_j) \cap A_{\mathrm{T}}^{\text{u-trip}} = \big(\mathfrak{u}_j, \mathrm{T}^-\big)$, for all $j \in \{1, \dots, q\}$, implies that the shape of all remaining T-vehicle schedules is like (3.14).

❹ All timetabled trips, which are not covered by one of the above described T-vehicle schedules, must be serviced by some F-vehicle schedule. Each set $\mathcal{G}_i$, $i \in \{1, \dots, p\}$, includes exactly two such nodes. Since $A(\mathcal{G}_i) = \emptyset$, for all $i \in \{0, \dots, p\}$, there exist at least two literal nodes (from $\mathcal{G}_1$) which are both covered by some F-vehicle schedule. Consider any literal node $t \in \bigcup_{i=1}^{p} \mathcal{G}_i$ that is serviced by some F-vehicle schedule: the flow condition (2.11), the flow conservation (2.12), and ❶ allow only F-vehicle schedules like (3.13), whereas j must be chosen such that $t \in \mathcal{H}_j$.

❺ We can conclude from ❹ that for a Boolean variable $\mathfrak{u}_j$ either all their nodes in $\mathcal{H}_j$ together build the nodes of some F-vehicle schedule or every node in $\mathcal{H}_j$ is a node of some T-vehicle schedule, and we can also conclude from ❷ that there exists at least one T-vehicle schedule like 3.14) as described in ❸.

❻ The fact that all variable nodes $\mathfrak{u}_j$, $j \in \{1, \dots, q\}$, are serviced follows from the flow condition (2.11).

The points ❶ – ❻ imply that the truth assignment

$$tr(\mathfrak{u}_j) := \begin{cases} \mathrm{T} & \text{if } x\big(\mathrm{T}^+, \mathfrak{u}_j\big) = 1,\ \big(\mathrm{T}^+, \mathfrak{u}_j\big) \in A_{\mathrm{T}}^{\text{pull-out}}, \\ \mathrm{F} & \text{if } x\big(\mathrm{F}^+, \mathfrak{u}_j\big) = 1,\ \big(\mathrm{F}^+, \mathfrak{u}_j\big) \in A_{\mathrm{F}}^{\text{pull-out}}, \end{cases} \tag{3.16}$$

for all $i \in \{1, \dots, q\}$, is satisfiable for the given OIT-3SAT-UL problem instance.

The $\mathcal{NP}$-completeness proof is done, the $\mathcal{NP}$-completeness in the strong sense immediately follows from the fact that the given transformation generates only the numbers 0 and 1 (see Garey and Johnson [1979], Chap. 4). ❑

Proof of Theorem 3.8: For an arbitrary problem instance of the MDVSP, let S_{opt} and $c(S_{\text{opt}})$ denote some optimal scheduling and its objective value, respectively. Given an $\varepsilon > 0$ and a polynomial time algorithm $\mathcal{A}$ that generates for every instance of the MDVSP a scheduling $S_{\mathcal{A}}$ satisfying $c(S_{\mathcal{A}}) \leqslant (1+\varepsilon)c(S_{\text{opt}})$. Similarly to the proof of Theorem 3.6, we transform the OIT-3SAT-UL to the decision problem of the ε-approximation MDVSP. Consider some $\delta > 0$. In difference to the proof of Theorem 3.6, L is set to $1+\varepsilon$, each arc's cost coefficient that was set to 1 is increased by $\varepsilon + \delta$, and $c_{(\mathrm{T}^+, \mathfrak{u}_0)}^{\mathrm{T}+}$ is set to 1. Everything else remains unchanged. We leave it to the reader to prove that an OIT-3SAT-UL instance is satisfiable if and only if the transformed problem has a feasible solution satisfying $c^{\mathrm{T}}x = 1 \leqslant 1+\varepsilon$ and that an OIT-3SAT-UL instance is unsatisfiable if and only if the objective of any feasible solution for the transformed problem satisfies $c^{\mathrm{T}}x > 1+\varepsilon$. ❑

Proof of Theorem 3.10: This proof is similar to the proof of Theorem 3.6. The theorem is already true for $|\mathcal{D}| = 2$ and $G \equiv \mathcal{D}$. Given such a capacitated instance of the MDVSP. The feasibility problem of the capacitated MDVSP is in $\mathcal{NP}$ since the additional work to the decision problem is to check the depot capacities, which can be done in polynomial time.

We transform the OIT-3SAT-UL to the capacitated feasibility problem of the MDVSP. Let $q \geqslant 3$, $p \geqslant 1$, $\mathfrak{U} = \{\mathfrak{u}_1, \dots, \mathfrak{u}_q\}$, and $\mathcal{C} = \{\mathfrak{C}_1, \dots, \mathfrak{C}_p\}$ make up an arbitrary instance of the OIT-3SAT-UL.

The depots are $\mathcal{D} = \{\mathrm{T}, \mathrm{F}\}$, the timetabled trips are

$$\mathcal{T} := \{\mathfrak{u}_0\} \cup \mathfrak{U} \cup \{\overline{\mathfrak{u}}_1, \dots, \overline{\mathfrak{u}}_{q-3}\} \cup \bigcup_{i=1}^{p} \mathfrak{C}_i \cup \{\mathfrak{u}'_0\} \cup \{\mathfrak{u}'_1, \dots, \mathfrak{u}'_q\}$$

($\{\overline{\mathfrak{u}}_1, \dots, \overline{\mathfrak{u}}_{q-3}\}$ are dummy nodes and $\{\mathfrak{u}'_0\} \cup \{\mathfrak{u}'_1, \dots, \mathfrak{u}'_q\}$ are copies of the nodes $\{\mathfrak{u}_0\} \cup \mathfrak{U}$), and $\mathcal{T}_{\mathrm{T}} = \mathcal{T}_{\mathrm{F}} = \mathcal{T}$.

For the definition of the dead-head trips, we first introduce some help sets:

- $\mathcal{G}_0 := \mathfrak{U}$,
- $\mathcal{G}_i := \mathfrak{C}_i$, for all $i \in \{1, \dots, p\}$,
- $\mathcal{G}_{p+1} := \{\mathfrak{u}'_1, \dots, \mathfrak{u}'_q\}$, and
- $\mathcal{H}_j := \{\mathfrak{u}_j\} \cup \{\mathfrak{c} \in \bigcup_{i=1}^{p} \mathfrak{C}_i : \mathfrak{c} \text{ is a literal of } \mathfrak{u}_j\} \cup \{\mathfrak{u}'_j\}$, for all $j \in \{1, \dots, q\}$.

Let "$\prec_j$" denote an order on $\mathcal{H}_j$ such that for $\{x, y\} \subseteq \mathcal{H}_j$:

$$x \prec_j y \Longrightarrow \exists\, m < n : x \in \mathcal{G}_m \wedge y \in \mathcal{G}_n.$$

$y \in \mathcal{H}_j$ is called the successor "succ(x)" of $x \in \mathcal{H}_j$ if $x \prec_j y$ and if there exists no element $z \in \mathcal{H}_j$ such that $x \prec_j z \prec_j y$. The arc set of the dead-head trips are set to

$$A_{\mathrm{T}}^{\text{d-trip}} := \big(\{\mathfrak{u}_0\} \to \mathcal{G}_1\big) \;\cup\; \bigcup_{i=2}^{p} \big(\mathcal{G}_{i-1} \to \mathcal{G}_i\big) \;\cup\; \big(\mathcal{G}_p \to \{\mathfrak{u}'_0\}\big) \;\cup\; \bigcup_{j=1}^{q} \big(\mathfrak{u}_j, \mathfrak{u}'_j\big)$$

and

$$A_{\mathrm{F}}^{\text{d-trip}} := \bigcup_{j=1}^{q} \Bigg(\bigcup_{\substack{t \in \mathcal{H}_j : \\ t \neq \max_{\prec_j} \mathcal{H}_j}} \big(t, \mathrm{succ}(t)\big) \Bigg).$$

For both depots, the lower bounds λ_{T} and λ_{F} are set to 2 and the upper bounds κ_{T} and κ_{F} are set to $q-1$, respectively. All arc costs are set to zero. It is easy to see that the above transformation can be performed in $\mathcal{O}(p+q)$ time.

Figure 3.5 shows the same OIT-3SAT-UL problem instance as Fig. 3.3: the Boolean variables are $\{\mathfrak{u}_1, \mathfrak{u}_2, \mathfrak{u}_3, \mathfrak{u}_4\}$ and the clauses are $\big\{\{\mathfrak{u}_1, \mathfrak{u}_2, \mathfrak{u}_3\}, \{\mathfrak{u}_1, \mathfrak{u}_2, \mathfrak{u}_4\}\big\}$, respectively. The task is to find a truth assignment $tr : \mathfrak{U} \longmapsto \mathcal{D}$ satisfying $(\mathfrak{u}_1 \dot\vee \mathfrak{u}_2 \dot\vee \mathfrak{u}_3) \wedge (\mathfrak{u}_1 \dot\vee \mathfrak{u}_2 \dot\vee \mathfrak{u}_4)$.

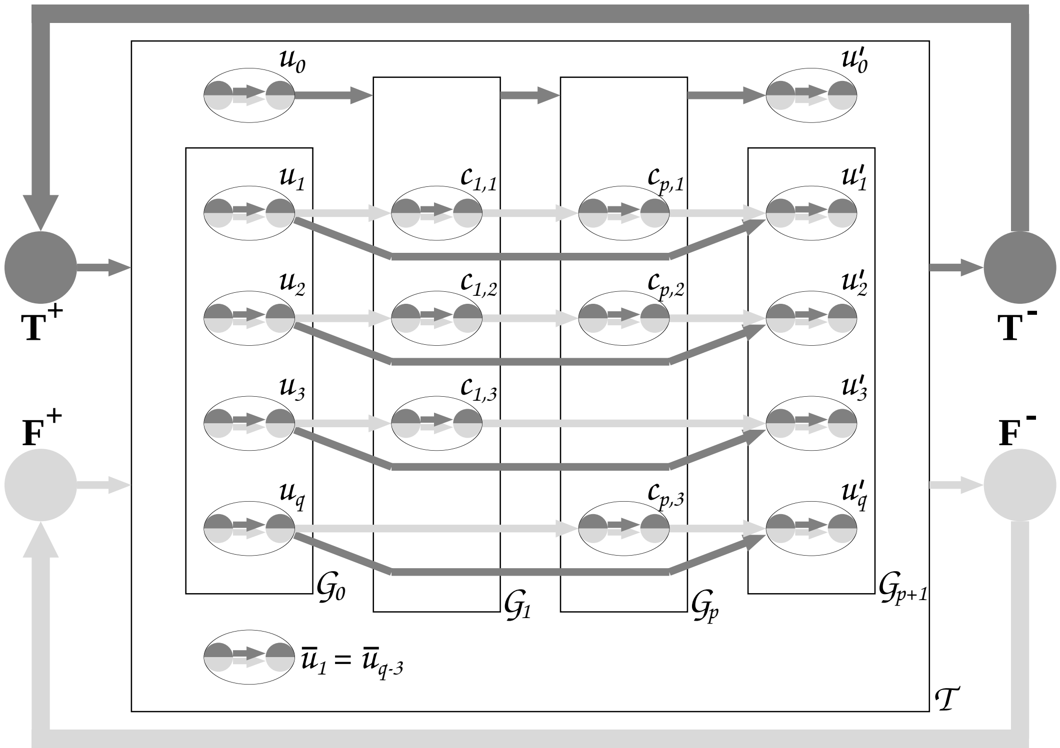


Figure 3.5: Capacitated MDVSP digraph $D' = (V', A')$ of a OIT-3SAT-UL.

Assume that we know a truth assignment satisfying the clauses of $\mathfrak{C}$. We will construct a feasible scheduling, whose incidence vector x is feasible. For each $\mathfrak{u}_j$ assigned to false, we define an F-vehicle schedule

(3.17)
$$\big(\mathrm{F}^+, \mathfrak{u}_j\big),\ \big(t, \operatorname{succ}(t)\big)\ \ \forall\, t \in \mathcal{H}_j : t \neq \max_{\prec_j} \mathcal{H}_j,\ \text{and}\ \big(\max_{\prec_j} \mathcal{H}_j, \mathrm{F}^-\big).$$

This covers all timetabled trips of $\bigcup_{j:\, tr(\mathfrak{u}_j)=\mathrm{F}} \mathcal{H}_j$. For each $\mathfrak{u}_j$ assigned to true, we define a T-vehicle schedule

$$\big(\mathrm{T}^+, \mathfrak{u}_j\big), \quad \big(\mathfrak{u}_j, \mathfrak{u}'_j\big), \quad \text{and} \quad \big(\mathfrak{u}'_j, \mathrm{T}^-\big). \tag{3.18}$$

This covers all (by F-vehicle schedules uncovered) nodes of $\mathcal{G}_0$ and $\mathcal{G}_{p+1}$. Let $\mathfrak{c}'_i \in \mathcal{G}_i$ denote the still uncovered literal node from each $\mathcal{G}_i$, for all $i \in \{1, \ldots, p\}$, (note: exactly two nodes of each $\mathcal{G}_i$ are covered by some F-vehicle schedule). To cover the nodes $\mathfrak{u}_0$, $\mathfrak{u}'_0$, and all these $\mathfrak{c}'_i$, we define a T-vehicle schedule

(3.19)
$$\big(\mathrm{T}^+, \mathfrak{u}_0\big),\ \big(\mathfrak{u}_0, \mathfrak{c}'_1\big),\ \big(\mathfrak{c}'_1, \mathfrak{c}'_2)\big),\ \ldots,\ \big(\mathfrak{c}'_{p-1}, \mathfrak{c}'_p\big),\ \big(\mathfrak{c}'_p, \mathfrak{u}'_0\big),\ \text{and}\ \big(\mathfrak{u}'_0, \mathrm{T}^-\big).$$

Until now, all nodes except $\{\overline{\mathfrak{u}}_1, \ldots, \overline{\mathfrak{u}}_{q-3}\}$ are covered. Since we defined exactly $q+1$ vehicle schedules, there is a total free capacity of $q-3$ vehicle schedules left for both depots together. For each node $\overline{\mathfrak{u}} \in \{\overline{\mathfrak{u}}_1, \ldots, \overline{\mathfrak{u}}_{q-3}\}$, we define either a T-vehicle schedule

$$(3.20) \qquad \big(\mathrm{T}^+, \overline{\mathfrak{u}}\big) \quad \text{and} \quad \big(\overline{\mathfrak{u}}, \mathrm{T}^-\big)$$

or an F-vehicle schedule

$$(3.21) \qquad \big(\mathrm{F}^+, \overline{\mathfrak{u}}\big) \quad \text{and} \quad \big(\overline{\mathfrak{u}}, \mathrm{F}^-\big)$$

depending on the free depot capacities. Figure 3.6 shows the transformation of two feasible truth assignments for the example of Fig. 3.5; only arcs with $x_a = 1$, for all $a \in A^{\text{u-trip}}$, are displayed.

Conversely, consider a feasible solution x:

① Since $\delta^-(t) = \big(\mathrm{T}^+, t\big) \cup \big(\mathrm{F}^+, t\big)$ for all $t \in \big(\{\mathfrak{u}_0\} \cup \mathcal{G}_0 \cup \{\overline{\mathfrak{u}}_1, \ldots, \overline{\mathfrak{u}}_{q-3}\}\big)$, x describes at least $|\{\mathfrak{u}_0\} \cup \mathcal{G}_0 \cup \{\overline{\mathfrak{u}}_1, \ldots, \overline{\mathfrak{u}}_{q-3}\}| = 1 + q + (q-3) = 2q-2$ vehicle schedules. This number of vehicle schedules is equal to the maximal capacity of both depots together. Therefore, some pull-out trip is used to service a timetabled trip node t if and only if $t \in \big(\{\mathfrak{u}_0\} \cup \mathcal{G}_0 \cup \{\overline{\mathfrak{u}}_1, \ldots, \overline{\mathfrak{u}}_{q-3}\}\big)$.

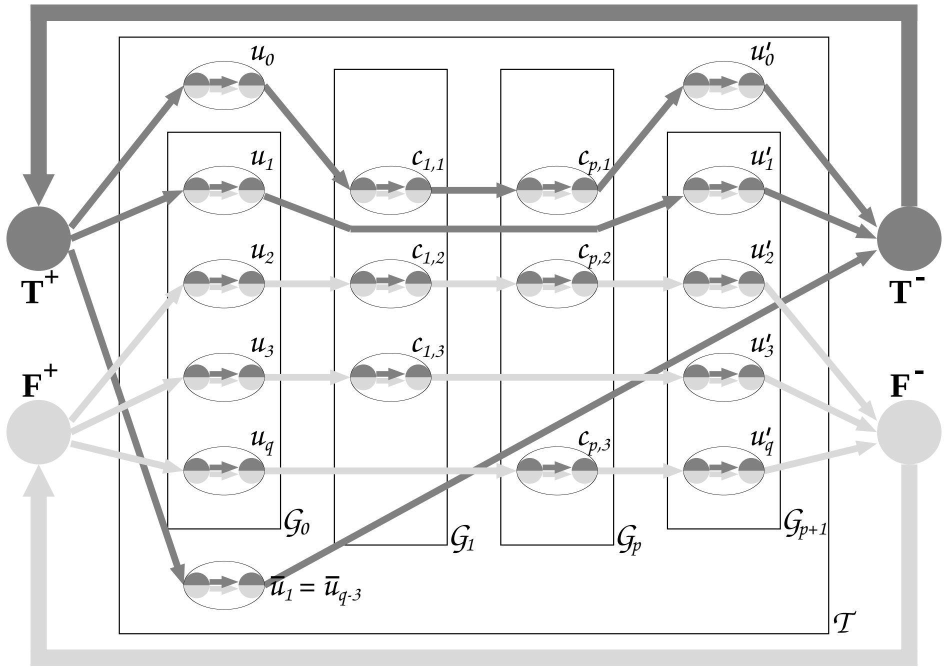
② The trip node $\mathfrak{u}'_0$ must be serviced with some T-vehicle schedule because ① and the flow condition (2.11) imply that $x\big(\mathrm{T}^+, \mathfrak{u}'_0\big) = x\big(\mathrm{F}^+, \mathfrak{u}'_0\big) = 0$ and, therefore, $x\big(\delta^-(\mathfrak{u}'_0) \cap A_{\mathrm{F}}^{\text{d-trip}}\big) = x\big(\emptyset\big) = 0$ or $x\big(\delta^-(\mathfrak{u}'_0) \cap A_{\mathrm{T}}^{\text{d-trip}}\big) = 1$. Tracing this vehicle schedule forward and tracing it backward with respect to the flow condition (2.11) and the flow conservation (2.12), the shape of the T-vehicle schedule is like (3.19).

③ For all $t \in \{\overline{\mathfrak{u}}_1, \ldots, \overline{\mathfrak{u}}_{q-3}\} : \delta(t) \cap \big(A_{\mathrm{T}}^{\text{d-trip}} \cup A_{\mathrm{F}}^{\text{d-trip}}\big) = \emptyset$. This implies that the vehicle schedule that services t looks like (3.20) or (3.21).

④ Consider the vehicle schedule of some timetabled trip node $\mathfrak{u}' \in \mathcal{G}_{p+1}$: tracing it forward and backward with respect to ①, to the flow conservation (2.12), and to $\delta^+(\mathfrak{u}') = \big(\mathfrak{u}', \mathrm{T}^-\big) \cup \big(\mathfrak{u}', \mathrm{F}^-\big)$ results either in a T-vehicle schedule like (3.18) or in an F-vehicle schedule like (3.17); these are precisely the vehicle schedules that also service all nodes from $\mathcal{G}_0$.

⑤ From ① follows that a feasible solution x must make use of the maximal depot capacities, which is $q-1$ in both cases. First, we can conclude that there exists at least one T-vehicle schedule of type (3.18) because one T-vehicle schedule is used as described in ② and there exist at most $q-3$ T-vehicle schedules of type (3.18). Second, there exist at least two F-vehicle schedules of type (3.17) because $\mathfrak{u}_0$ is serviced from depot T and there exist at most $q-3$ F-vehicle schedules of type (3.21).

⑥ Whenever a (literal) timetabled trip node $\mathfrak{c} \in \bigcup_{i=1}^{p} \mathcal{G}_i$ is serviced from a T-vehicle schedule, the flow conservation (2.12) implies that this is the T-vehicle schedule from ②. It is also known from ②, that for each $\mathcal{G}_i$, $i \in \{1, \ldots, p\}$, exactly two nodes



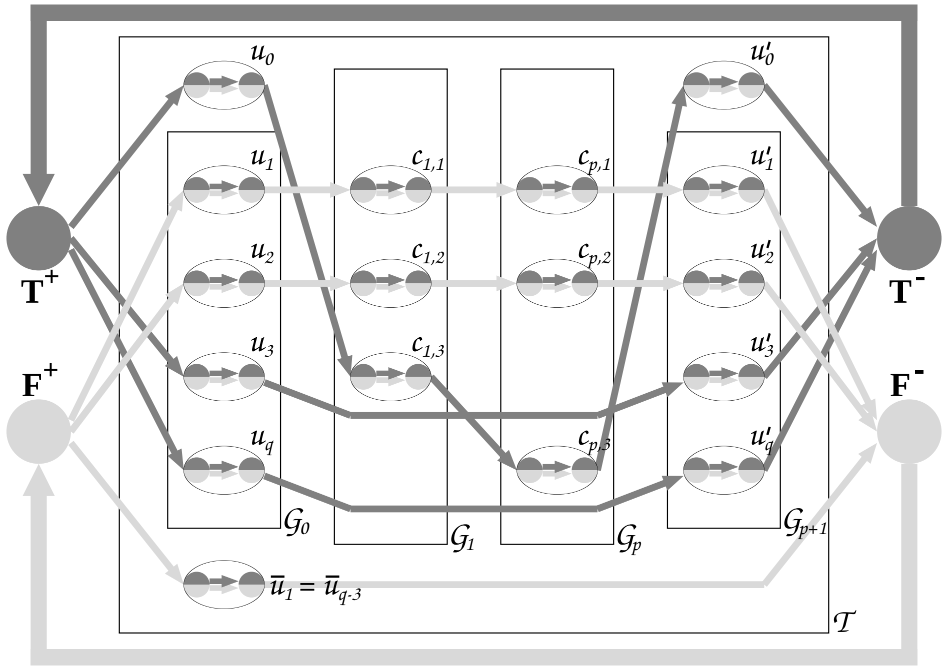


Figure 3.6: Two feasible truth assignments for the instance of Fig. 3.5.

cannot be serviced by this T-vehicle schedule. These two nodes can only be serviced by an F-vehicle schedule. But for the same reasons as for the nodes in $\mathcal{G}_{p+1}$, this F-vehicle schedule can only be of type (3.17).

⑦ The fact that all variable nodes $\mathfrak{u}_j$, $j \in \{1, \dots, q\}$, are serviced follows from the flow condition (2.11).

Points ① – ⑦ imply that the truth assignment

$$tr(\mathfrak{u}_j) := \begin{cases} \mathrm{T} & \text{if } x\big(\mathrm{T}^+, \mathfrak{u}_j\big) = 1, \ \big(\mathrm{T}^+, \mathfrak{u}_j\big) \in A_{\mathrm{T}}^{\text{pull-out}}, \\ \mathrm{F} & \text{if } x\big(\mathrm{F}^+, \mathfrak{u}_j\big) = 1, \ \big(\mathrm{F}^+, \mathfrak{u}_j\big) \in A_{\mathrm{F}}^{\text{pull-out}}, \end{cases} \tag{3.22}$$

for all $i \in \{1, \dots, q\}$, is satisfiable for the given OIT-3SAT-UL problem instance. ❑

Chapter 4

A Literature Overview

In this chapter, we give an overview on the literature dealing with the MDVSP and variants on it.

Arc-oriented multicommodity flow (as presented in Chap. 2) and path-oriented Dantzig-Wolfe (DW) set partitioning formulations (see Chap. 11) are usually used to model the MDVSP. Applied to vehicle scheduling problems from practice, their corresponding ILP formulations provide several million integer variables. Solving such large ILPs requires column generation techniques.

For the arc-oriented model, column generation can be seen as an implicit pricing technique (see Schrijver [1989]): one works on restricted subsets of active arcs that are generated and eliminated in a dynamic process. For the DW decomposition, column generation usually leads to pricing problems in the form of constraint shortest path problems. Many researchers automatically associate the term "column generation" with the solution process used in a DW decomposition (e. g., see Soumis [1997]). To distinguish this use of the term "column generation" from those as a general LP pricing technique in the sense of Schrijver, DW column generation is also called *delayed column generation* as proposed in Chvátal [1980]. To avoid misunderstandings, we will use in this paper the term "column generation" as a general LP pricing technique in the sense of Schrijver.

Dantzig-Wolfe decomposition models are needed for problems that involve path constraints. They apply not only to vehicle scheduling problems, but also to applications of similar flavour, e. g., to crew and airline scheduling. For a survey on set partitioning approaches to such problems, we refer the reader to Desrosiers, Dumas, Solomon, and Soumis [1995], Barnhart, Hane, and Vance [1996], Barnhart, Johnson, Nemhauser, and Vance [1997], and Soumis [1997].

Direct approaches to the multicommodity flow formulation can be used if all side constraints can be formulated solely in terms of the arcs of the network. This is the case for the MDVSP considered here.

All the models presented in the literature use *natural* flow formulation with their (self-suggesting) integer linear programming analogous. Depot groups and depot capacities are of course natural, but have often not been considered in the literature about the MDVSP. Conditions of these type have often been ignored to simplify the used notation, to receive nice results for special problem cases, or were not required by the real problem that has been investigated. In most cases, the only difference between the models published by now and our model presented in Chap. 2 is whether conditions of this kind have been considered or not.

In the following, we give a collection of references to computer-aided planning systems that are employed in practice. We continue with a survey on models and solution approaches in vehicle routing and scheduling. Afterwards, we give an overview on the SDVSP and the MDVSP.

4.1 Computer-Aided planning systems

From a practical point of view, there are several publications that report on experiences with computer-aided planning systems like

- BUSMAN, see Chamberlain and Wren [1988,1992];
- BERTA ("BEtRiebseinsaTzplanung und -Auswertung") of the Berliner Verkehrsbetriebe (BVG) and IVU GmbH, Berlin, see Löbel and Strubbe [1996] and Becker, Roß, and Schemczyk [1996];
- HASTUS, see Blais and Rousseau [1988] and Hamer and Séguin [1992];
- HOT II ("Hamburger Optimization Technique") of the Hamburger Hochbahn AG and HanseCom GmbH, Hamburg, see Hoffstadt [1981], Mojsilovic [1983], Daduna and Mojsilovic [1988], Daduna, Mojsilovic, and Schütze [1993], Schütze and Völker [1995], and Petzold and Schütze [1995].

4.2 Some Surveys

Bodin and Golden.

Bodin and Golden [1981] give a "Classification in Vehicle Routing and Scheduling". They define the term vehicle *route* as an ordered sequence of pickup or delivery points traversed by a vehicle, starting and ending at a depot. A vehicle *schedule* is a sequence of pickup or delivery points together with an associated set of arrival and departure times, which a vehicle traverses in the designated order at the specified times. When arrival times at nodes or arcs are fixed in advance, this is called a *scheduling* problem. Problems with unspecified arrival times are referred to as straightforward *routing* problems. When time windows or precedence relationships exist such that routing and scheduling functions need to be performed, the authors view such problems as combined vehicle routing and scheduling problems.

Bodin and Golden discuss several variations of vehicle routing and scheduling problems and provide a taxonomy for these problems, e. g., a classification in a homogeneous or a heterogeneous vehicle fleet, one or more than one depot, time restriction on particular arcs or nodes, etc. This taxonomy is followed by a classification of solution strategies for routing problems, e. g., cluster first – route second, route first – cluster second, saving approaches, etc. up to exact procedures. Finally, a short description of three combined routing and scheduling problems is given.

The interesting part of Bodin and Golden, from our point of view, is the hierarchy classification of vehicle scheduling problems: They start with a simple version of an SDVSP having the task to minimize the fleet size (or capital costs) only. Those problems are solved using flow algorithms. If, in addition, operational costs must be considered, they first solve the simple SDVSP, which computes the minimum fleet size, fix the minimum fleet size value, and minimize the total operational costs for all minimal fleet solutions. This problem is solved using a minimum-cost flow algorithm. We will show in Chap. 5 how this two stage approach can be composed to a single-stage approach. The next straightforward extension of the SDVSP is to allow multiple depots, which the authors solve either by a cluster first – schedule second or by a schedule first – cluster second approach. Depot groups are not considered, but the authors discuss restrictions on path lengths, time windows, etc.

Assad, Ball, Bodin, and Golden.

The survey of Assad, Ball, Bodin, and Golden [1983] gives a comprehensive overview about vehicle routing and scheduling approaches. The methods presented in Bodin and Golden [1981] seem to be a compact version of the techniques of Assad et al [1983]. Both articles give a taxonomy followed by some characteristics of routing, scheduling, and mixtures of both problems. Although Assad et al. devote a chapter to each of routing, scheduling, and combined routing and scheduling problems, we will concentrate our summary on their survey on scheduling problems.

The authors formulate SDVSPs as

$$\min \sum_{a \in A^{\text{u-trip}}} c_a\, x_a \tag{4.1a}$$

subject to

$$
\begin{aligned}
&(4.1\text{b}) & x\big(\delta^+(t)\big) &= 1, && \forall\, t \in \mathcal{T},\\
&(4.1\text{c}) & x\big(\delta^+(t)\big) - x\big(\delta^-(t)\big) &= 0, && \forall\, t \in \mathcal{T},\\
&(4.1\text{d}) & x_a &\geqslant 0, && \forall\, a \in A^{\text{u-trip}},
\end{aligned}
$$

which can be transformed to (3.4) without depot capacities and with an eliminated $x_{(d^-,d^+)}$ variable. The weights of the arcs are defined by capital costs (for a minimization of the fleet size) and operating cost. They describe two solution approaches. As in the case of Bodin and Golden [1981], they always consider either capital costs or operating costs (with a fixed fleet size) and apply a two stage approach to optimize such a problem. They also present a single stage approach as we will present in Chap. 5.

Although the authors are aware of the problem with limited durations of dead-head trips (see the model discussion in Chap. 2.3), they do not use trips like our pull-in-pull-out trips because there are $\mathcal{O}(|\text{timetabled trips}|^2)$ many of them.

SDVSPs with path restrictions are known to be $\mathcal{NP}$-hard (e. g., see Freling and Paixão [1995]). Assad et al. give an ILP formulation for such problems, but do not describe how such problems can be solved efficiently.

MDVSPs are formulated as multicommodity flow problems with the following ILP formulation, which is equivalent to (2.13) without depot groups:

$$\min \sum_{d \in \mathcal{D}} \sum_{a \in A_d^{\text{u-trip}}} c_a^d \, x_a^d \tag{4.2a}$$

subject to

$$
\begin{aligned}
&(4.2b) & x\big(\delta^+(t)\big) &= 1, && \forall\, t \in \mathcal{T},\\
&(4.2c) & x^d\big(\delta^+(t)\big) - x^d\big(\delta^-(t)\big) &= 0, && \forall\, t \in \mathcal{T} \;\; \forall\, d \in \mathcal{D},\\
&(4.2d) & \lambda_d \leqslant x^d\big(\delta^+(d)\big) &\leqslant \kappa_d, && \forall\, d \in \mathcal{D},\\
&(4.2e) & 0 \leqslant x_a^d &\leqslant 1, && \forall\, a \in A^{\text{u-trip}} \;\; \forall\, d \in \mathcal{D},\\
&(4.2f) & x &\in \{0,1\}^A. &&
\end{aligned}
$$

Assad et al. describe heuristics like a concurrent scheduler, an interchange method (similar to the 2-opt heuristic for travelling salesman problems), a cluster first – schedule second method, and a schedule first – cluster second method, but no exact method to solve the MDVSP.

Carraresi and Gallo.

Carraresi and Gallo [1984] give a review on solving vehicle and crew scheduling problems in public mass transit. They formulate the SDVSP as an assignment problem, which can be solved by assignment or general purpose minimum-cost flow algorithms, and discuss the different possibilities how the cost function can be used to minimize capital costs or operational costs or a mixture of both.

The MDVSP is considered without depot groups and without lower depot capacities. It is formulated as a multicommodity flow problem based on the assignment model for the single-depot case. In our notation, their formulation reads

$$\min \sum_{d \in \mathcal{D}} \sum_{a \in A'_d} c_a^d \, x_a^d \tag{4.3a}$$

subject to

$$\sum_{d\in\mathcal{D}} x^d_{(t^-,t^+)} = 1, \qquad \forall\, t \in \mathcal{T}, \tag{4.3b}$$

$$x^d\big(\delta^+(t^+)\big) - x^d_{(t^-,t^+)} = 0, \qquad \forall\, t \in \mathcal{T} \;\; \forall\, d \in \mathcal{D}, \tag{4.3c}$$

$$x^d_{(t^-,t^+)} - x^d\big(\delta^-(t^-)\big) = 0, \qquad \forall\, t \in \mathcal{T} \;\; \forall\, d \in \mathcal{D}, \tag{4.3d}$$

$$x^d\big(\delta^+(d^+)\big) \leqslant \kappa_d, \qquad \forall\, d \in \mathcal{D}, \tag{4.3e}$$

$$x^d_a \geqslant 0, \qquad \forall\, a \in A^{\text{u-trip}}_d \;\; \forall\, d \in \mathcal{D}, \tag{4.3f}$$

$$x^d_a \in \{0,1\}, \qquad \forall\, a \in A^{\text{t-trip}}_d \;\; \forall\, d \in \mathcal{D}. \tag{4.3g}$$

This formulation is similar to our multicommodity flow formulation (2.7). The authors propose the two heuristics cluster first – schedule second and schedule first – cluster second and give a note on a Lagrangean relaxation approach to solve the MDVSP.

Desrosiers et al.

Desrosiers, Dumas, Solomon, and Soumis [1995] give a survey on various vehicle scheduling and routing problems, we just summarize their "Fixed Schedule Problems".

Their MDVSP formulation does neither consider depot groups nor lower bounds for the depot capacities. The formulation is taken from Ribeiro and Soumis [1994] and reads

$$\min \sum_{d\in\mathcal{D}} \sum_{a\in A^{\text{u-trip}}_d} c_a\, x^d_a \tag{4.4a}$$

subject to

$$x\big(\delta^+(t)\big) = 1, \qquad \forall\, t \in \mathcal{T}, \tag{4.4b}$$

$$x^d\big(\delta^+(t)\big) - x^d\big(\delta^-(t)\big) = 0, \qquad \forall\, t \in \mathcal{T} \cup \{d\} \;\; \forall\, d \in \mathcal{D}, \tag{4.4c}$$

$$x^d\big(\delta^+(d)\big) \leqslant \kappa_d, \qquad \forall\, d \in \mathcal{D}, \tag{4.4d}$$

$$x \geqslant 0, \tag{4.4e}$$

$$x \in \{0,1\}^A. \tag{4.4f}$$

This problem formulation is very close to (4.2) of Assad et al. and to our formulation (2.14). For the single-depot case, (4.4) reads

$$\min \sum_{a\in A^{\text{u-trip}}} c_a\, x_a \tag{4.5a}$$

subject to

$$x\big(\delta^+(t)\big) = 1, \qquad \forall\, t \in \mathcal{T}, \tag{4.5b}$$

$$x\big(\delta^+(t)\big) - x\big(\delta^-(t)\big) = 0, \qquad \forall\, t \in \mathcal{T} \cup \{d\}, \tag{4.5c}$$

$$x\big(\delta^+(d)\big) \leqslant \kappa, \tag{4.5d}$$

$$x \geqslant 0. \tag{4.5e}$$

To apply some minimum-cost flow algorithm to (4.5), the authors transform the problem into an equivalent LP formulation of a minimum-cost flow problem similarly to (3.1).

In addition to the general minimum-cost flow formulation, the authors discuss how the SDVSP can be solved using a transportation or an assignment formulation. They also mention that the assignment formulation cannot handle depot capacities.

The authors are also aware of the problem with limited durations of dead-head trips. To avoid the use of a quadratic number of variables, they give references to the articles of Bokinge and Hasselström [1980], Desrochers, Desrosiers, and Soumis [1985], and Lamatsch [1988,1992] who give a problem formulation with a linear number of variables. We have discussed such approaches in Chap. 2.

Desrosiers et al. give a list of articles describing some heuristic algorithms for the MDVSP. The branch-and-bound approach of Carpaneto, Dell'Amico, Fischetti, and Toth [1989] and the Lagrangean relaxation approach of Lamatsch [1988,1992] are shortly described. A very detailed description is given for the Dantzig-Wolfe decomposition approach of Ribeiro and Soumis [1994].

Daduna and Paixão.

The problem classification and the SDVSP description in Daduna and Paixão [1995] closely follows Assad, Ball, Bodin, and Golden [1983] with some additional extensions to the SDVSP. Daduna and Paixão give illustrative description of

- assignment approaches (e. g., Mojsilovic [1983]),
- quasi-assignment approaches (e. g., Branco and Paixão (1987,1988)),
- matching approaches (e. g., Bertossi, Carraresi, and Gallo [1987]),
- transportation approaches (e. g., Gavish, Schweitzer, and Shlifer [1978] and Gavish and Shlifer [1978]),
- and minimum-cost flow approaches (e. g., Carraresi and Gallo [1984]).

The authors formulate the MDVSP as a multicommodity flow problem, but do not consider depot groups. Some practical requirements, however, as multiple vehicle types, path restrictions, timetable sensitivity analysis (e. g., see Ceder and Stern [1981], Fuchs [1992], or Daduna, Mojsilovic, and Schütze [1993]), etc. are discussed separately.

4.3 The Single Depot Case

We consider the single-depot case to be important since SDVSP instances occur not only as pure vehicle scheduling problems. They have to be solved repeatedly as subproblems of

- cluster first – schedule second approaches,
- schedule first – cluster second approaches,
- schedule – cluster – reschedule approaches,
- Lagrangean relaxations of MDVSPs resulting in SDVSP instances, and
- timetable sensitivity analysis, i. e., shifting some timetabled trips by few minutes may improve the vehicle schedule (see Schütze and Völker [1995], Daduna, Mojsilovic, and Schütze [1993], Daduna and Mojsilovic [1988], Mojsilovic [1983], Hoffstadt [1981], and Bokinge and Hasselström [1980]).

Löbel [1996a] describes a network simplex implementation in C, which solves very large real-world SDVSP instances from practice efficiently. With this code, it is possible to solve problem instances with up to 50 thousand of nodes and 70 million of unloaded trips, see also Chap. 5 and Chap. 12. This network simplex code, called MCF, is available for academic use free of charge via WWW at URL http://www.zib.de/Optimization, see Löbel [1997b].

For academic use, there are also other efficient codes available free of charge as, for instance, the relaxation code RELAX4 of Bertsekas and Tseng [1994] or the cost scaling code CS2 of Goldberg [1992].

Further articles reporting on the SDVSP are Fuchs [1992], Branco and Paixão [1987,1988], Desrochers, Desrosiers, and Soumis [1985], and Bokinge and Hasselström [1980]. The computational investigations of all these publications are on rather small instances and became therefore obsolete. Nevertheless, the current basic solution approaches are still the same.

4.4 The Multiple Depot Case

The list of publications about the MDVSP can be classified by the following attributes:

Size of the solved test instances. By now, only heuristics have been successfully applied to solve large instances. In most cases, exact methods and Lagrangean relaxation approaches have only been applied to rather small instances.

Multicommodity flow formulations. Are the used models based on multicommodity flow formulations or not? This question is important since depot groups cannot be considered exactly in single-commodity flow formulations. Only some special cases of the MDVSP reduce to the single-depot case, e. g., problems considering no depot groups and having depot-independent weights of the pull-out and pull-in trips, see Carpaneto, Dell'Amico, Fischetti, and Toth [1989].

Publications that use multicommodity flow formulations are, e. g., Grötschel, Löbel, and Völker [1997], Löbel [1997a,1997c,1996b], and Mesquita and Paixão [1997], Larsen and Madsen [1997], Kokott and Löbel [1996], Branco, Costa, and Paixão [1995], Forbes, Holt, and Watts [1994], Ribeiro and Soumis [1994], Dell'Amico, Fischetti, and Toth [1993], Mesquita and Paixão [1992], Lamatsch [1988,1992], Carpaneto, Dell'Amico, Fischetti, and Toth [1989], Bertossi, Carraresi, and Gallo [1987], Carraresi and Gallo [1984], Gavish, Schweitzer, and Shlifer [1978], Gavish and Shlifer [1978],

Publications that do not use multicommodity flow formulations are, for instance, Daduna and Völker [1997], Petzold and Schütze [1995], Schütze and Völker [1995], Daduna, Mojsilovic, and Schütze [1993], Daduna and Mojsilovic [1988], Mojsilovic [1983], and Hoffstadt [1981].

Consideration of depot groups. Depot groups are used in Larsen and Madsen [1997], Branco, Costa, and Paixão [1995], Forbes, Holt, and Watts [1994], and Lamatsch [1988,1992] as well as Grötschel, Löbel, and Völker [1997], Kokott and Löbel [1996], and Löbel [1997a,1997c,1996b]. In addition, Branco, Costa, and Paixão [1995] and Forbes, Holt, and Watts [1994] use the ILP formulation

$$\min \sum_{d \in \mathcal{D}} \sum_{a \in A_d^{\text{u-trip}}} c_a^d x_a^d \tag{4.6a}$$

subject to

$$\begin{aligned}
&(4.6\text{b}) & x\big(\delta^+(t)\big) &= 1, && \forall\, t \in \mathcal{T},\\
&(4.6\text{c}) & x^d\big(\delta^+(t)\big) - x^d\big(\delta^-(t)\big) &= 0, && \forall\, t \in \mathcal{T}_d \;\; \forall\, d \in \mathcal{D},\\
&(4.6\text{d}) & x^d\big(\delta^+(d)\big) &\leqslant \kappa_d, && \forall\, d \in \mathcal{D},\\
&(4.6\text{e}) & x &\geqslant 0, &&\\
&(4.6\text{f}) & x &\in \{0,1\}^A, &&
\end{aligned}$$

which is exactly the same formulation as our ILP (2.14) without depot lower capacities.

Lagrangean Relaxations.

Grötschel, Löbel, and Völker [1997], Larsen and Madsen [1997], Kokott and Löbel [1996], Branco, Costa, and Paixão [1995], Lamatsch [1988,1992], Mesquita and Paixão [1992]), and Bertossi, Carraresi, and Gallo [1987] discuss Lagrangean relaxation approaches for the MDVSP.

Basically, the MDVSP is described by the flow condition and the flow conservation constraints. Hence, two Lagrangean relaxation approaches are reasonable: First, a relaxation with respect to the flow conditions resulting in subproblems that decompose into independently solvable minimum-cost flow problems, see Grötschel, Löbel, and Völker [1997], Larsen and Madsen [1997], Kokott and Löbel [1996], and Bertossi, Carraresi, and Gallo

[1987]. Second, a relaxation with respect to the flow conservations resulting in subproblems that are polynomially solvable SDVSP instances, see Grötschel, Löbel, and Völker [1997], Kokott and Löbel [1996], Branco, Costa, and Paixão [1995], Mesquita and Paixão [1992], and Lamatsch[1988,1992].

These two relaxation approaches have the following features:

1. In Kokott and Löbel [1996], we show that these two Lagrangean relaxations together with a subgradient method can be used for efficient computation of tight lower bounds for substantial problems from practice.
2. A popular method for finding feasible solutions of truly large-scale MDVSPs are certainly schedule first – cluster second approaches, which are sometimes followed by a reschedule method, see below and in Chap. 8. With respect to the flow conservations, the Lagrangean relaxation together with some primal (interchange and/or greedy) heuristic can be viewed as such a schedule first – cluster second approach. A similar idea as above can be applied to the Lagrangean relaxation with respect to the flow conditions resulting in a cluster first – schedule second approach. The latter is, however, less popular than the first approach.
3. In Grötschel, Löbel, and Völker [1997] and Löbel (1997d), we describe these Lagrangean relaxation techniques as new column generation strategies for the LP relaxation of the MDVSP, see Chap. 7.

Except of Grötschel, Löbel, and Völker [1997] and Kokott and Löbel [1996], the computational investigations of the mentioned articles on Lagrangean relaxation are based on test instances with at most 600 timetabled trips for a 10-depot-problem and 1791 timetabled tripsfor a 3-depot-problem.

Heuristics.

Bodin, Kydes, and Rosenfield [1978] propose a concurrent scheduler heuristic: Timetabled trips are considered according to their increasing starting times; let $t_j \in \mathcal{T}$ denote the j-th timetabled trip with respect to the starting time order. Let $H := \emptyset$ denote the set of vehicles that are currently in use. The following steps are performed for each $j = 1, \ldots, |\mathcal{T}|$: If there exists at least one vehicle $h \in H$ such that the last timetabled trip t_i assigned to h can be connected to t_j by an unloaded trip (t_i, t_j), select among them the cheapest unloaded trip and assign t_j to the corresponding vehicle. If this is not possible, a new vehicle h is used and inserted to H. This new vehicle is selected from some depot $d \in \mathcal{D}$ that still has a free capacity and minimizes the pullout costs.

Some other heuristics are the schedule first – cluster second approach (see Löbel [1997c], Grötschel, Löbel, and Völker [1997], Daduna, Mojsilovic, and Schütze [1993], Dell'Amico, Fischetti, and Toth [1993], Daduna and Mojsilovic [1988], Carraresi and Gallo [1984], Mojsilovic [1983], Assad, Ball, Bodin, and Golden [1983], Hoffstadt [1981], Bodin and Golden [1981], Gavish and Shlifer [1978], and Gavish, Schweitzer, and Shlifer [1978]), the cluster first – schedule second approach (see Löbel [1997c], Larsen and Madsen [1997], Grötschel, Löbel, and Völker [1997], Dell'Amico, Fischetti, and Toth [1993], Mesquita and Paixão [1992], Carraresi and Gallo [1984]), and reschedule procedures (see Löbel [1997c], Grötschel, Löbel, and Völker [1997], and Dell'Amico, Fischetti, and Toth [1993]). We shall describe this kind of approach in Chap. 8.

Branco, Costa, and Paixão [1995] propose an LP-based rounding heuristic: The optimal (nonintegral) solution of the ILP relaxation is rounded to some integral feasible solution. Afterwards, they apply a saving heuristic similar to the 2-opt heuristic for the travelling salesman problem. In Löbel [1997c], we also describe an LP based heuristic, which iteratively rounds components of a fractional LP solution to zero and one, respectively, reoptimize the enlarged LP, and proceeds until the problems becomes infeasible or an integer solution is found. This iterative rounding heuristic is called *LP-plunging*, for more details see Chap. 8.

Set Partitioning Approaches.

Löbel [1997a], Desrosiers, Dumas, Solomon, and Soumis [1995], Branco, Costa, and Paixão [1995], Bianco, Mingozzi, and Ricciardelli [1994], Ribeiro and Soumis [1994], and Lamatsch [1988] discuss set partitioning approaches (based on Dantzig-Wolfe decomposition) for the MDVSP. The rows and columns of the set partitioning problem correspond to the timetabled trips and to all possible vehicle schedules. A comprehensive description of the Dantzig-Wolfe decomposition scheme applied to the MDVSP is given in Desrosiers et al.

Lamatsch [1988] applies the Dantzig-Wolfe decomposition principle to his ILP formulation and gives a set partitioning formulation of the MDVSP without depot capacities. Since this approach was not suitable for the computers of that time, it is rejected by the author.

The first article, to the best of our knowledge, reporting about a decomposition approach and computational investigations (with up to 10 depots and 300 timetabled trips) is published by Ribeiro and Soumis [1994]. Applying the Dantzig-Wolfe decomposition principle to the MDVSP, they reformulate the MDVSP as a set partitioning problem with additional side constraints for the depot capacities.

A delayed column generation approach for the general pickup and delivery problem resulting in a set partitioning approach with additional side constraints can be found in the PhD thesis of Sol [1994]. Barnhart, Johnson, Nemhauser, Savelsbergh, and Vance [1994] discuss delayed column generation approaches for various problem classes and give an insight into branch-and-price approaches. Barnhart, Hane, and Vance [1996] report about computational investigations of a branch-and-price approach for the integer multicommodity flow problem. Although they claim that they solve *large* integer test instances, their two largest integer problems do not have more than 91 nodes, 203 arcs, and 18 commodities and 50 nodes, 130 arcs, and 585 commodities. Moreover, only the smallest integer problem with 50 nodes, 97 arcs, and 15 commodities could be solved to optimality;

for all the other integer test instances they stopped the run after one hour cpu time with a positive gap between the best integral solution and the branch-and-price lower bound.

Branch-and-Bound Approaches.

Up to now, only relatively small real-world MDVSPs have been solved to optimality: Forbes, Holt, and Watts [1994] solve their ILP formulation in three stages: First, they relax the problem resulting in a SDVSP that can be solved by a network simplex algorithm. Second, the optimal solution of the relaxed problem is used to construct a dual feasible basis for the LP relaxation. The corresponding basis solution satisfies the flow conditions that each timetabled trip has to be serviced exactly by one vehicle, but some flow conservations may be violated. The dual simplex algorithm is then applied to resolve the LP relaxation to optimality. Third, a branch-and-bound approach is used to find an optimal integral solution. The authors report on numerical investigations with up to 3 depots and 600 timetabled trips, some of these instances are taken from real-world problems.

Carpaneto, Dell'Amico, Fischetti, and Toth [1989] describe a different ILP formulation for the MDVSP based on an assignment formulation with additional cycle (or path) oriented flow conservation constraints. They apply a so-called "additive lower bounding" procedure proposed by Fischetti and Toth [1988] to receive a lower bound of their ILP formulation and apply a branch-and-bound approach to solve the MDVSP. Carpaneto et al. report on computational investigations for artificially generated problems with up to 3 depots and 70 timetabled trips. Ribeiro and Soumis [1994] show that this additive lower bounding is nothing but a special application of a Lagrangean relaxation and its corresponding subgradient method. Therefore, the lower bound of Carpaneto et al. is never better than the lower bound provided by the Lagrangean dual of the LP or the ILP formulation for a multicommodity flow model.

Computational Investigations.

Considering the presented computational investigations, most of the mentioned references in this literature overview put either *academic* or *practice-oriented* accents. The first describe mathematical background and methods, but do not tell how large-scale real-world problems can be solved. The computational results, if reported at all, are based on investigations with small, mostly randomly generated test instances having less than one thousand timetabled trips. This is different for practice-oriented publications:

- In Löbel [1997c], Grötschel, Löbel, and Völker [1997], Löbel (1997d), and Kokott and Löbel [1996], we have reported on solving practical MDVSPs to optimality.
- The heuristic vehicle scheduling approach of the HOT II system is reported by Schütze and Völker [1995], Daduna, Mojsilovic, and Schütze [1993], Mojsilovic [1983], Daduna and Mojsilovic [1988], and Hoffstadt [1981], all previously or currently employees of the Hamburger Hochbahn AG or its subsidiary HanseCom

GmbH. Although they do not explicitly report their computational results, their solutions are known to be of a high quality. Grötschel, Löbel, and Völker [1997] compared the optimal solutions of our exact branch-and-cut approach with the solutions of HOT II and find out that, on the average, a 3 % vehicle reduction and a 10 % cost saving are possible. From a practical point of view, the HOT system generates quite good solutions. The reason why the HOT system does not necessarily provide an optimal solution will be discussed in Chap 8.4.

- Forbes, Holt, and Watts [1994] report about real-world problems "of an actual bus operator". The full timetabled trip set of the bus operator consists of more than 6500 timetabled trips and three depots. On the average, 72 % of the trips can be serviced by only one depot, 25 % by two depots, and 3 % by all three depots. From this test set, which probably was too large for the used personal computer, several randomly extracted problems with up to 600 timetabled trips are considered. Since the average depot group size $\varnothing G = 1.31$, the considered test instances are not too difficult. This is also reflected by the fact that for 22 of all the 30 extracted test instances the optimal solution of the LP relaxation was already integral. For the other 8 test problems, the gap between the optimal value of the LP relaxation and the optimal value of a feasible integer solution is at most 0.003 %.
- Gavish, Schweitzer, and Shlifer [1978] report already in 1978 about a system that can solve problems with up to 2500 timetabled trips. Depot groups, however, are not considered such that MDVSPs can be reduced to SDVSPs with a schedule first – cluster second approach. With their system, parts of a manually generated set of vehicle schedules of a bus company operating more than 4500 timetabled trips, could be refined significantly.

Chapter 5

Solving the Single-Depot Case

SDVSPs occur not only in a pure form, but also as subproblems within heuristics or Lagrangean duals of MDVSPs. We will show in this thesis that solving the MDVSP requires at several steps the efficient and exact solution of SDVSPs. Although the SDVSP can be formulated as a polynomially solvable minimum-cost flow problem and it is well known how such flow problems can be solved efficiently, we devote an own chapter to the solution techniques for SDVSPs.

We present an efficient solution method based on the network simplex implementation MCF, see Löbel [1997b]. This code is able to solve SDVSPs of any relevant size. We start with a short summary of the network simplex algorithm and continue with some implementation details of our network simplex code. We assume the reader to be familiar with the network simplex algorithm. The last section describes how MCF is used to solve SDVSP instances.

5.1 The Primal Network Simplex Algorithm

The network simplex algorithm with upper bound technique is a specialized revised simplex algorithm, see Dantzig [1963] or Chvátal [1980], that exploits the structure of network flow problems. The linear algebra of the simplex algorithm is replaced by simple network operations. Helgason and Kennington [1995] and Ahuja, Magnanti, and Orlin [1989,1993] describe the (primal) network simplex algorithm and give pseudocodes, implementation hints, etc.

To apply the simplex algorithm to (1.17) $\min\{c^{\mathrm{T}}x \mid \mathcal{N}x = b,\ 0 \leqslant x \leqslant u\}$, we need a full rank constraint matrix. For a connected network D, the rank of the flow conservation constraints (1.16b) is equal to $|V| - 1$, and the flow conservation constraint for one designated node, the so-called **root node**, can be eliminated. We will assume that we have chosen such a root node and have eliminated its flow conservation constraint, i. e., the reduced node-arc incidence matrix has full rank. For notational simplification, we also denote the reduced node-arc incidence matrix by $\mathcal{N}$. It is well known that every nonsingular basis matrix B of $\mathcal{N}$ corresponds to a spanning tree of A in D and vice versa. Let $T \subseteq A$ be a spanning tree in D. The variables x_{ij}, $(i,j) \in T$, are called the **basic variables** corresponding to the basis matrix $B := \mathcal{N}_{\cdot,T}$. Let L and U denote the arcs that correspond to the **nonbasic variables** whose values are set to the lower and upper bound, respectively. The triple (T, L, U) is called a **basis structure**. For given nonbasic arc sets L and U, the right hand side b transforms to

$$b' := b - \sum_{(i,j)\in U} \mathcal{N}_{\cdot,ij} u_{ij}.$$

The associated basic solution is the solution of the system $Bx_T = b'$, the values of the node potentials are determined by the system $\pi^{\mathrm{T}} B = c_T^{\mathrm{T}}$. Let $\overline{c}_{ij} := c_{ij} - \pi_i + \pi_j$ denote the **reduced costs** of an arc $(i,j) \in A$. The dual multipliers η of the bounds $x \leqslant u$ are determined by

$$\eta_{ij} := \begin{cases} -\overline{c}_{ij} & \text{if } (i,j) \in U, \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

A basis structure (T, L, U) is called **primal feasible** if the associated basic solution x satisfies the flow bounds (1.16c) and is called **dual feasible** if for all $(i,j) \in A$:

$$\overline{c}_{ij} > 0 \quad \Rightarrow \quad (i,j) \in L, \tag{5.2a}$$

$$\overline{c}_{ij} < 0 \quad \Rightarrow \quad (i,j) \in U, \tag{5.2b}$$

A basis structure is called **optimal** if it is both primal and dual feasible. For further information about the primal network simplex, especially flow charts and pseudo codes, see Helgason and Kennington [1995].

5.2 Implementation Details

Many network flow textbooks contain (relatively similar) pseudocodes of network flow algorithms. We started our implementation with the pseudocode given by Ahuja, Magnanti, and Orlin [1989] and tried to improve important algorithmic details to make it more robust and efficient such that even truly large-scale problems can be solved routinely. We describe the key ingredients of our modifications. Most of our computational improvements result, in fact, from very efficient pricing strategies. The importance of pricing follows from our experimental observations that our implementation still spends, on the average, more than 80 percent of the cpu time on pricing.

In the description of the primal network simplex algorithm, we have assumed the input network to be connected, which we ensure by the following procedure: Given an arbitrary (not necessarily connected) network $D = (V, A)$, we add to V one artificial root node, denoted by "0". Each original node i of V is then connected to the root node 0 either by the artificially generated arc $(i, 0)$ if i is a supply or transshipment node or by the artificially generated arc $(0, i)$ if i is a demand node. Let $D' := (V \cup \{0\}, A')$ denote the network obtained by adding the artificial root node 0 to V and adding the artificial slack arcs $(i, 0)$, for each supply and transshipment node, and $(0, i)$, for each demand node, to A. Each artificial slack arc has a lower bound of 0, an upper bound of infinity, and a sufficiently large cost coefficient $\mathbf{M}$.

Computer Language and Data Structures.

Lustig [1990] investigates the influence of the computer languages C and FORTRAN on an implementation of a primal network simplex code. He finds out that, for both languages, an address-based implementation (linked lists and pointers) is more efficient than a cursor-based implementation (vectors and indices) and that the performances of cursor-based implementations in C and Fortran are essentially the same. Our MCF code is implemented in C with address-based data structures.

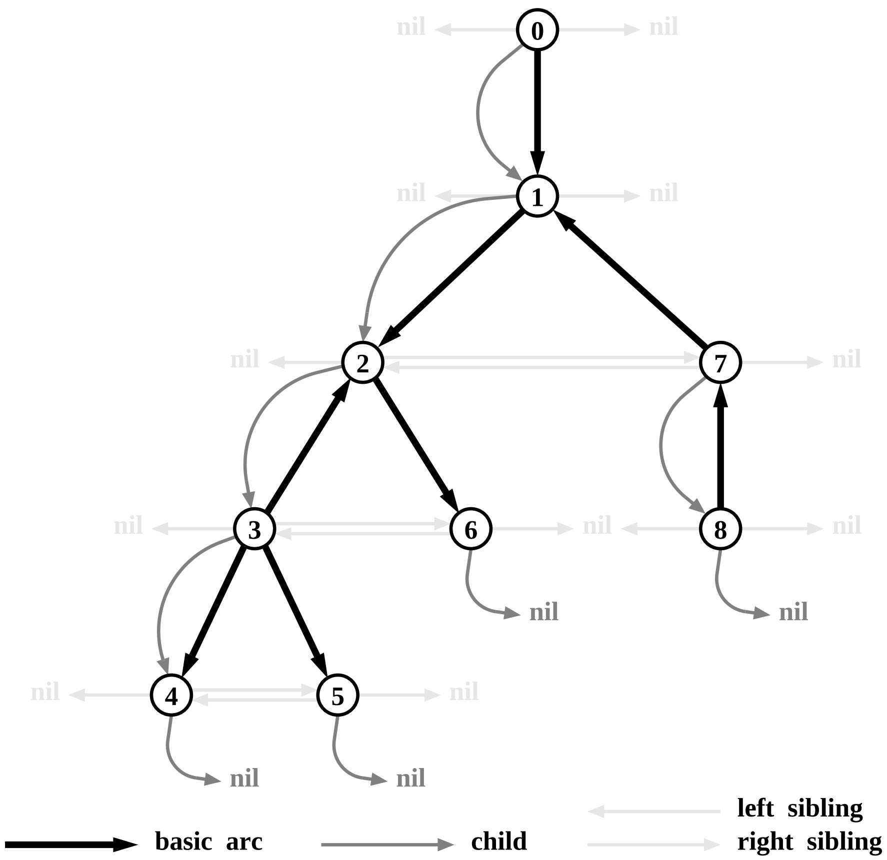
Over the last three decades, the basis tree representation and data structures for the network simplex algorithm have been investigated profoundly. Most of the network simplex implementations use similar data structures. We describe our version. All node and arc information, respectively, are stored in the following data structures:

Node information: Let $T \subseteq A$ be a spanning tree in D, and consider some node $v \in V \setminus \{0\}$. There is a unique (undirected) path in T, denoted by $P(v)$, leading from v to the root node 0. The arc in $P(v)$, which is incident to v, is called the **basic arc** of v. The other terminal node u of this basic arc is called the **predecessor** (**node**) of v. The basic arc of v is called **upward** (**downward**) oriented if v is the tail (head) node of its basic arc. If u is the predecessor of some other node v, we call v a **child** (**node**) of u. Suppose there is some order of the children of v, and let u and w be two different children of v. If u is smaller than w with respect to the given order, we call u the **left sibling** of w and w the **right sibling** of u. If there is no child u that is smaller (greater) than a given child w, then w has no left (right) sibling. Each node has at most one child reference, the other children of a node can be reached by traversing the sibling links. The number of nodes in $P(V)$ is called the **subtree size** of v.

The subtree size and predecessor variables are used by the ratio test. The orientation, child, and sibling variables are used for the computation of the node potentials.

Arc information: For each arc we store information about its tail and head node, its upper bound value, its costs, and whether it is a basic arc of T or a nonbasic arc of L or U.

Figure 5.1 shows a small example of a rooted basis tree for our data structures (the underlying network is a copy from Ahuja, Magnanti, and Orlin [1993]). Our technique to



node	0	1	2	3	4	5	6	7	8
subtree size	9	8	5	2	1	1	1	2	1
predecessor	nil	0	1	2	3	3	2	1	7
child	1	2	3	4	nil	nil	nil	8	nil
right sibling	nil	nil	7	6	5	nil	nil	nil	nil
left sibling	nil	nil	nil	nil	nil	4	3	2	nil
orientation	-	down	down	up	down	down	down	up	up

Figure 5.1: Rooted basis tree.

store the rooted basis tree results from personal discussions with R. E. Bixby.

Pricing.

Based on our computational experience, the pricing rule has the most significant influence on the performance of a network simplex implementation. Ahuja, Magnanti, and

Orlin [1993] describe some pricing rules such as Dantzig's rule, first eligible arc rule, and a candidate list rule. We have implemented and tested these pricing rules in slightly modified ways. It turned out that our by far fastest rules are special candidate list rules, called **multiple partial pricing** ,e. g., see Bixby[1992,1994].

Given two natural numbers K and J. The arc set A is divided into $\lceil \frac{|A|}{K} \rceil$ candidate lists, each of size at most K. If the arcs are indexed from 1 to $|A|$, the k^{th} candidate list includes all arcs i satisfying $(i-1)$ modulo $K = (k-1)$. There is a *hot-list* of at most $J+K$ arcs, which is initially empty. The candidate list number *next*, which defines the first to be examined candidate list in the initial pricing call, is set to 1. The candidate lists are always examined in a wraparound fashion. For a pricing call, the following steps are performed: First, the reduced costs of the arcs being currently in the hot-list are recomputed. If the new reduced costs of such an arc becomes nonnegative, this arc is immediately removed from the hot-list. Second, as long as the hot-list can be filled with at least K additional arcs and not all candidate lists have been examined in this pricing call, we price out all arcs of the *next* candidate list, add all nonbasic arcs of this list having negative reduced costs to the hot-list, and increment the *next* variable by 1 (if $next \leqslant K$, otherwise we reset *next* to 1). Third, if all candidate lists have been examined, but the hot-list is still empty, the current basis structure is optimal. Otherwise, some arc of the hot-list that most violates the reduced cost criterion is selected as the basis entering arc. The last step of a pricing call is the preparation of the hot-list for the next call: The basis entering arc leaves the hot-list. At most J arcs with most invalid reduced costs enter the new list.

Multiple partial pricing is very sensitive to the number of arcs which makes a fine tuning for every problem class necessary. We use the following default values for K and J depending on the number of arcs:

Number of arcs	K	J
$\|A\| < 10,000$	30	5
$10,000 \leqslant \|A\| \leqslant 100,000$	50	10
$\|A\| > 100,000$	200	20

Compared with the multiple partial pricing (with the default values of K and J as above), pricing rules such as first eligible arc rule or Dantzig's rule are about 14 to 75 times slower, see Löbel [1996a].

Initial Basis Structure.

The easiest way to find an initial primal feasible basis structure, is as follows: The initial basis tree consist of all artificial slack arcs and each original arc becomes nonbasic at its lower bound. Note, that no arc is at its upper bound. Such an initial basis structure is called **artificial basis structure**. Obviously, this artificial basis structure is primal feasible for D', and the original network D is feasible if the network D' has a feasible solution where no artificial arc has a positive flow value.

The use of an artificial basis tree has several advantages. First, it has a simple structure and can be generated quickly. Second, the ratio test and the basis update are quite fast for the first iterations. We have also tried to generate an initial basis structure using a crash procedure. The performance, however, was always slower than starting with an artificial basis tree. The only exceptions occur for special applications where a particular problem knowledge can be exploited, for instance, if we apply a column generation.

Sensitivity Analysis and Column Generation.

For large-scale networks, the performance often benefits from a column generation approach: In a first step, only a *restricted* subset $\tilde{A} \subset A'$ is considered, and the flow value of each arc $a \in A' \setminus \tilde{A}$ is fixed to zero. $\tilde{A}$ contains all artificial arcs to ensure the existence of a primal feasible basis structure.

When the restricted network $(V, \tilde{A})$ has been solved to optimality, all fixings of the ignored arcs are removed, and the reduced costs according to the last node potentials are computed for all arcs (**sensitivity analysis**). As long as there exist arcs that violate the optimality conditions, we add at least one and at most a (parameter controlled) maximum number of such arcs to the restricted arc set $\tilde{A}$ (**column generation**), reoptimize for the new enlarged arc set $\tilde{A}$, and iterate until optimality can be proved for the complete arc set A. Note that the original problem without artificial arcs has been solved to optimality if no artificial arc has a positive flow value. To avoid that $\tilde{A}$ grows too much, some nonbasic arcs of L, i. e., arcs whose flow values are set to 0, may also (parameter controlled) be removed from $\tilde{A}$, e. g., when the reduced costs are greater than some predefined threshold.

The initial subproblem $(V, \tilde{A})$ is optimized using an artificial basis structure. The new generated arcs become all nonbasic in L. Each subsequent restricted problem restarts with the optimal basis structure of its previous subnetwork.

Strongly feasible bases.

With the concept of strongly feasible bases, see Cunningham [1976], and a combinatorial version of perturbation, one can prove that the network simplex algorithm theoretically runs in pseudo-polynomial time, see Ahuja, Magnanti, and Orlin [1989], pages 305–310.

Our computational experiments show always a good "polynomial" behaviour of our network simplex implementation MCF. Even truly large-scale test instances are quickly solvable such that, on the average, a low-order polynomial in the number of arcs and nodes can be assumed for the complexity.

5.3 Applying the Network Simplex Code to SDVSP

For SDVSPs (3.4) we have designed the following algorithm that is based on the above described network simplex code in combination with sensitivity analysis and column generation.

The depot's starting point d^+ was chosen as the root node. A positive depot lower bound λ is transformed to zero (as shown in Fig. 1.1) by the variable substitution $x_{(d^-,d^+)} := x'_{(d^-,d^+)} + \lambda$, which results in

$$\text{(5.3a)} \qquad \min \sum_{a \in A^{\text{u-trip}}} c_a \, x_a$$

subject to

$$\begin{array}{lrcll}
\text{(5.3b)} & x\big(\delta^+(t^+)\big) & = & 1, & \forall\, t \in \mathcal{T}, \\
\text{(5.3c)} & -x\big(\delta^-(t^-)\big) & = & -1, & \forall\, t \in \mathcal{T}, \\
\text{(5.3d)} & x'_{(d^-,d^+)} - x\big(\delta^-(d^-)\big) & = & -\lambda, & \\
\text{(5.3e)} & 0 \;\leqslant\; x'_{(d^-,d^+)} & \leqslant & \kappa - \lambda, & \\
\text{(5.3f)} & x_a & \geqslant & 0, & \forall\, a \in A^{\text{u-trip}}.
\end{array}$$

Note that in the case of a fixed fleet size, i. e., $\kappa = \lambda$, the variable $x'_{(d^-,d^+)}$ can be eliminated (since it is set to zero), the depot's starting point and end point contract to one single depot node, and (5.3) reduces to

$$\text{(5.4a)} \qquad c_{(d^-,d^+)}\, \lambda \;+\; \min \sum_{a \in A^{\text{u-trip}}} c_a \, x_a$$

subject to

$$\begin{array}{lrcll}
\text{(5.4b)} & x\big(\delta^+(t^+)\big) & = & 1, & \forall\, t \in \mathcal{T}, \\
\text{(5.4c)} & -x\big(\delta^-(t^-)\big) & = & -1, & \forall\, t \in \mathcal{T}, \\
\text{(5.4d)} & -x\big(\delta^-(d^-)\big) & = & -\lambda, & \\
\text{(5.4e)} & x_a & \geqslant & 0, & \forall\, a \in A^{\text{u-trip}}.
\end{array}$$

All constructed artificial arcs together with the user-defined unloaded trips define the initial restricted arc set $\tilde{A}$. Between two subsequent subproblems, the column generation (of currently fixed pull-in-pull-out trips) is limited to a number ranging between 30 and 50 thousand arcs. To keep the restricted subnetworks small enough, it is also possible to remove "bad" arcs corresponding to a reduced cost criterion from $\tilde{A}$. The parameter control for the column generation and elimination depends on the problem type, the available main memory, etc.

The choice of the M for the artificial slack arcs has to be done very carefully since the objective coefficients of the pull-out trips can already contain a M value (see Chap. 2). Therefore, the cost coefficients of the artificial slack arcs are set to a sufficiently large **MM** that also dominates the M of the pull-out trips.

The column generation is nothing but a special pricing strategy. Our proposed method for solving SDVSPs runs in pseudo-polynomial time since we use strongly feasible bases. Of course, there exist several even strongly polynomial time algorithms for minimum-cost flow problems (e. g., see Ahuja, Magnanti, and Orlin [1993], page 395), but our network simplex code with column generation performs, on the average, always better than very efficient implementations (with default configuration) of such polynomial time algorithms for our test data (see Löbel [1996a]). Even our largest SDVSPs instances (derived by Lagrangean relaxation) with 70 million arcs can be solved to optimality within less than 15 minutes cpu time. Pure real-world SDVSP with up to several thousand timetabled trips and several million unloaded trips can be solved to optimality within few minutes.

Chapter 6

Solving Lagrangean relaxations

We describe in this chapter the two basic ideas how the Lagrangean relaxation approach can be applied to the MDVSP such that the resulting Lagrangean duals become efficiently solvable. The techniques presented here have already been presented in Kokott and Löbel [1996].

In this chapter, the considered model are based on the digraph D' (2.1). Based thereon, we give a somewhat blown up ILP formulation of the MDVSP including redundant constraints:

$$\min \sum_{d \in \mathcal{D}} \sum_{a \in A_d^{\text{u-trip}}} c_a^d \, x_a^d \tag{6.1a}$$

subject to

$$
\begin{array}{llcll}
(6.1\text{b}) & \displaystyle\sum_{d \in G(t)} x^d_{(t^-,t^+)} & = & 1, & \forall\, t \in \mathcal{T}, \\
(6.1\text{c}) & x\big(\delta^+(t^+)\big) & = & 1, & \forall\, t \in \mathcal{T}, \\
(6.1\text{d}) & -x\big(\delta^-(t^-)\big) & = & -1, & \forall\, t \in \mathcal{T}, \\
(6.1\text{e}) & x^d\big(\delta^+(t^+)\big) - x^d\big(\delta^-(t^-)\big) & = & 0, & \forall\, t \in \mathcal{T}_d \ \forall\, d \in \mathcal{D}, \\
(6.1\text{f}) & x^d\big(\delta^+(t^+)\big) - x^d_{(t^-,t^+)} & = & 0, & \forall\, t \in \mathcal{T}_d \ \forall\, d \in \mathcal{D}, \\
(6.1\text{g}) & x^d_{(t^-,t^+)} - x^d\big(\delta^-(t^-)\big) & = & 0, & \forall\, t \in \mathcal{T}_d \ \forall\, d \in \mathcal{D}, \\
(6.1\text{h}) & x^d_{(d^-,d^+)} - x^d\big(\delta^-(d^-)\big) & = & 0, & \forall\, d \in \mathcal{D}, \\
(6.1\text{i}) & \lambda_d \ \leqslant \ x^d_{(d^-,d^+)} & \leqslant & \kappa_d, & \forall\, d \in \mathcal{D}, \\
(6.1\text{j}) & x^d_{(t^-,t^+)} & \leqslant & 1, & \forall\, (t^-,t^+) \in A_d^{\text{t-trip}} \ \forall\, d \in \mathcal{D}, \\
(6.1\text{k}) & x^d_a & \geqslant & 0, & \forall\, a \in A_d^{\text{t-trip}} \cup A_d^{\text{u-trip}} \ \forall\, d \in \mathcal{D}, \\
(6.1\text{l}) & & x \text{ integral.} & &
\end{array}
$$

The flow conditions are given three times by the equivalent constraints (6.1b), (6.1c), and (6.1d). Moreover, (6.1) includes the additional (redundant) flow conservations (6.1e). For each of the two Lagrangean relaxation approaches presented below, we shall select those constraints of (6.1) that give a sufficient problem description, but that are necessary to formulate each relaxation properly.

6.1 Relaxation of the Flow Conservations

The first relaxation is to put the flow conservations into the objective function. As we will see, the resulting inner minimization problem of the Lagrangean dual is a large SDVSP problem. However, to receive such a nice problem structure, we have to neglect the individual depot lower and upper capacities for this relaxation. This does not matter since depot capacities are often soft constraints, see Rem. 3.11, and we consider here a relaxation anyway (for most of our problems, we are unable to solve the following Lagrangean dual to optimality – with or without depot capacities). Hence, the variables $x^d_{(d^-,d^+)}$ and the constraints (6.1h) and (6.1 i) are ignored. The underlying ILP formulation for the first Lagrangean relaxation reads

$$\min \sum_{d\in\mathcal{D}} \sum_{a\in A_d^{\text{u-trip}}} c_a^d\, x_a^d \tag{6.2a}$$

subject to

$$\begin{aligned}
x\big(\delta^+(t^+)\big) &= 1, && \forall\, t \in \mathcal{T}, && \text{(6.2b)}\\
-x\big(\delta^-(t^-)\big) &= -1, && \forall\, t \in \mathcal{T}, && \text{(6.2c)}\\
x^d\big(\delta^+(t^+)\big) - x^d\big(\delta^-(t^-)\big) &= 0, && \forall\, t \in \mathcal{T}_d\ \ \forall\, d \in \mathcal{D}, && \text{(6.2d)}\\
x_a^d &\geqslant 0, && \forall\, a \in A_d^{\text{u-trip}}\ \ \forall\, d \in \mathcal{D}, && \text{(6.2e)}\\
& x \text{ integral.} && && \text{(6.2f)}
\end{aligned}$$

Note that the formulation (6.2) does not contain variables for the timetabled trips and the backward arcs. Since (6.2c) is a linear combination of (6.2b) and (6.2d), it is easy to see that (6.2) is equivalent to (2.14) without depot capacities.

Let $\pi := (\pi^d \in \mathbb{R}^{\mathcal{T}_d})_{d\in\mathcal{D}}$ denote the Lagrange multipliers associated with the flow conservations (6.2d). Let the subscript "fcs" of L_{fcs} and LR_{fcs} (defined below) be an abbreviation for **F**low-**C**on**S**ervation. Let LR_{fcs} denote the following Lagrangean relaxation: With respect to (6.2d), the Lagrangean dual of (6.2) is

$$\max_{\pi} L_{\text{fcs}}(\pi). \tag{6.3}$$

with inner minimization problem

(6.4)
$$L_{\text{fcs}}(\pi) := \min_{\substack{x\,\geqslant\,0 \text{ integral}\\ \text{satisfying}\\ (6.2\text{b}) \text{ and } (6.2\text{c})}} \sum_{d\in\mathcal{D}} \Bigg(\sum_{a\in A_d^{\text{u-trip}}} c_a^d\, x_a^d \;-\; \sum_{t\in\mathcal{T}_d} \pi_t^d \Big(x^d\big(\delta^+(t^+)\big) - x^d\big(\delta^-(t^-)\big)\Big)\Bigg).$$

The equations (6.2b) and (6.2c) are exactly the side constraints of the ILP (3.4) of a SDVSP without depot capacities. Hence, $L_{\text{fcs}}(\pi)$ corresponds to a large minimum-cost flow problem, and it follows from Remark 1.13 that LR_{fcs} and the LP relaxation of (6.1) (without depot capacities) yield the same optimal value.

Any optimal solution $x := x(\pi)$ attaining the value of $L_{\text{fcs}}(\pi)$ describes a set of vehicle schedules covering each timetabled trip exactly once. Some of these vehicle schedule, however, may violate flow conservations (6.2d) if its trips belong to different depots.

6.2 Relaxation of the Flow Conditions

The second relaxation is obtained by putting the flow conditions into the objective function. Its underlying ILP formulation reads

$$\text{(6.5a)} \qquad \min \sum_{d \in \mathcal{D}} \sum_{a \in A_d^{\text{u-trip}}} c_a^d x_a^d$$

subject to

$$\begin{array}{llcll}
\text{(6.5b)} & \displaystyle\sum_{d \in G(t)} x^d_{(t^-,t^+)} & = & 1, & \forall\, t \in \mathcal{T}, \\
\text{(6.5c)} & x^d\big(\delta^+(t^+)\big) - x^d_{(t^-,t^+)} & = & 0, & \forall\, t \in \mathcal{T}_d \ \ \forall\, d \in \mathcal{D}, \\
\text{(6.5d)} & x^d_{(t^-,t^+)} - x^d\big(\delta^-(t^-)\big) & = & 0, & \forall\, t \in \mathcal{T}_d \ \ \forall\, d \in \mathcal{D}, \\
\text{(6.5e)} & x^d\big(\delta^+(d^+)\big) - x^d_{(d^-,d^+)} & = & 0, & \forall\, d \in \mathcal{D}, \\
\text{(6.5f)} & x^d_{(d^-,d^+)} - x^d\big(\delta^-(d^-)\big) & = & 0, & \forall\, d \in \mathcal{D}, \\
\text{(6.5g)} & \lambda_d \ \leqslant \ x^d_{(d^-,d^+)} & \leqslant & \kappa_d, & \\
\text{(6.5h)} & x^d_{(t^-,t^+)} & \leqslant & 1, & \forall\, (t^-,t^+) \in A_d^{\text{t-trip}} \ \ \forall\, d \in \mathcal{D}, \\
\text{(6.5i)} & x^d_a & \geqslant & 0, & \forall\, a \in A_d^{\text{t-trip}} \cup A_d^{\text{u-trip}} \ \ \forall\, d \in \mathcal{D}, \\
\text{(6.5j)} & & & x \text{ integral.} &
\end{array}$$

Let $\nu := (\nu_t)_{t \in \mathcal{T}} \in \mathbb{R}^{\mathcal{T}}$ denote the Lagrange multipliers associated with the flow conditions (6.5b). Let the subscript "fcd" of L_{fcd} and LR_{fcd} be an abbreviation for **F**low-**C**on**D**ition. Let LR_{fcd} denote the following Lagrangean relaxation: With respect to (6.5b), the Lagrangean dual of (6.5) is

$$\text{(6.6)} \qquad \max_{\nu} L_{\text{fcd}}(\nu).$$

with inner minimization problem

(6.7)
$$L_{\mathrm{fcd}}(\nu) := \min_{\substack{x \text{ satisfies} \\ (6.5\,\mathrm{c}\,)\text{–}(6.5\,\mathrm{j}\,)}} \Big(\sum_{d \in \mathcal{D}} \sum_{a \in A_d^{\text{u-trip}}} c_a^d \, x_a^d \;-\; \sum_{t \in \mathcal{T}} \nu_t \Big(\sum_{d \in G(t)} x^d_{(t^-,t^+)} \;-\; 1 \Big) \Big).$$

Since L_{fcd} is equivalent to

$$\nu^{\mathrm{T}} \mathbb{1} \;+\; \sum_{d \in \mathcal{D}} \min_{\substack{x^d \text{ satisfies} \\ (6.5\,\mathrm{c}\,)\text{–}(6.5\,\mathrm{j}\,)}} \Big(\sum_{a \in A_d^{\text{u-trip}}} c_a^d \, x_a^d \;-\; \sum_{t \in \mathcal{T}_d} \nu_t \, x^d_{(t^-,t^+)} \Big), \tag{6.8}$$

it decomposes into a constant part $\nu^{\mathrm{T}}\mathbb{1}$ and into $|\mathcal{D}|$ independently solvable minimum-cost flow problems. Note that each of these minimum-cost flow subproblem is equivalent to its continuous relaxation. It follows from Remark 1.13 that $\mathrm{LR}_{\mathrm{fcd}}$ and the LP relaxation of (6.1) yield the same optimal value.

It is easy to see that a feasible solution of each subproblem

$$\min_{\substack{x^d \text{ satisfies} \\ (6.5\,\mathrm{c}\,)\text{–}(6.5\,\mathrm{j}\,)}} \Big(\sum_{a \in A_d^{\text{u-trip}}} c_a^d \, x_a^d \;-\; \sum_{t \in \mathcal{T}_d} \nu_t \, x^d_{(t^-,t^+)} \Big), \tag{6.9}$$

corresponds to a set of vehicle schedules satisfying the depot specific capacities. The additional inequalities (6.5h) ensure that the vehicle schedules of a depot cover each timetabled trip at most once. The Lagrange multipliers correspond to the shadow prices of the timetabled trips and measure their *attractivity*. It is easy to see that the objective of (6.8) is to find those vehicle schedules that give the most objective progress for the given shadow prices. Considering all vehicle schedules of all depots together, some timetabled trip may not be serviced at all or may be serviced by more than one depot, i. e., some flow conditions may be violated. In those cases, the associated Lagrange multipliers are increased or decreased to make the timetabled trips *cheaper* or *more expensive*.

6.3 Subgradient Methods

We solve the Lagrangean relaxations $\mathrm{LR}_{\mathrm{fcs}}$ and $\mathrm{LR}_{\mathrm{fcd}}$ using a subgradient method. The general idea is the following: Let L denote one of the two concave functions L_{fcs} and L_{fcd}. A **subgradient** for L in u is a vector g that satisfies $L(u+z) \leqslant L(u) + g^{\mathrm{T}} z$ for each z. For a given $u^{(0)}$, let $x^{(0)}$ denote some optimal solution of $L(u^{(0)})$. This solution can be used to easily generate a subgradient by the following well known lemma.

(6.10) Lemma. $g^{(0)} := d - D\,x^{(0)}$ is a subgradient for L in $u^{(0)}$.

It follows from Lemma (6.10) that any u^* attaining the maximum value of L yields $(u^* - u^{(0)})^{\mathrm{T}} g^{(0)} \geqslant L(u^*) - L(u^{(0)}) \geqslant 0$. The optimal value of the Lagrangean dual may be approximated by the following procedure: Start with some arbitrary $u^{(0)}$ and recursively create a sequence $u^{(0)}, u^{(1)}, u^{(2)}, \ldots$ of new points, each depending on its previous points and the computed subgradients hitherto. Note that each $L(u^{(k)})$ is a lower bound on the maximum value of L, i. e., it is also a lower bound for the value of the ILP (2.14) and (6.1), respectively. Our basic method is as follows:

(6.11) Algorithm. Basic Subgradient Method for the MDVSP.
Input: L.
Output: Lower bound for ILP (2.14) and (6.1), respectively.
1. Choose initial Lagrange multiplier $u^{(0)}$ and set $k := 0$.
2. Evaluate $L(u^{(k)})$.
3. Compute subgradient $g^{(k)} := g^{(k)}(u^{(k)})$.
4. If the iteration limit N_1 is reached or $g^{(k)} = 0$ then STOP; otherwise continue.
5. Compute new step length $\sigma^{(k)}$.
6. Compute new step direction $\tilde{g}^{(k)} \in \operatorname{conv}\{g^{(0)}, \ldots, g^{(k)}\}$.
7. Set $u^{(k+1)} := u^{(k)} + \sigma^{(k)}\tilde{g}^{(k)}$.
8. Increment k and go to 2.

Here follow the details of our subgradient algorithm (6.11):

Step 1. Choose initial Lagrange multiplier.

For $\mathrm{LR}_{\mathrm{fcs}}$, the initial $\pi^{(0)}$ is set to zero. Initializing $\nu^{(0)}$ for $\mathrm{LR}_{\mathrm{fcd}}$: It turned out from computational investigations that the following initialization of the Lagrange multipliers $\nu^{(0)}$ performs best concerning the maximal achieved objective value L_{fcd} and concerning the number of iterations of the subgradient method. Solve $L_{\mathrm{fcs}}(0)$ of the Lagrangean relaxation $\mathrm{LR}_{\mathrm{fcs}}$. Let ν^+ and ν^- denote the optimal dual multipliers of the constraints (6.2b) and (6.2c), respectively. The estimation of $\nu^{(0)}$ by $\nu^+ - \nu^-$ defines a good starting point since $L_{\mathrm{fcd}}(\nu^{(0)})$ and $L_{\mathrm{fcs}}(0)$ yield the same value.

Here is the proof of this statement: For each $d \in \mathcal{D}$, let $\nu_d^+ := \nu_d^- := 0$ denote the node potential of the contracted depot node. With respect to $L_{\mathrm{fcs}}(0)$, we know that the reduced costs $\overline{c}_{ij}^d := c_{ij}^d - \nu_i^+ + \nu_j^-$ are nonnegative for each unloaded trip. Since $x^d_{(t^-,t^+)} = x^d\big(\delta^-(t^-)\big) = x^d\big(\delta^+(t^+)\big)$, it follows straightforwardly by inserting ν^+ and ν^- that (6.9) is equivalent to

$$\min_{\substack{x^d \text{ satisfies} \\ (6.5\,\mathrm{c})-(6.5\,\mathrm{j})}} \sum_{(i,j)\in A_d^{\text{u-trip}}} c_{ij}^d - \nu_i^+ + \nu_j^- .$$

Since this objective function is nonnegative for each d and x^d describes a circulation in this case, it is easy to see that $x^d = 0$ is optimal. Thus, $L_{\mathrm{fcd}}(\nu^+ - \nu^-) = \mathbb{1}^{\mathrm{T}}\nu^+ - \mathbb{1}^{\mathrm{T}}\nu^- = L_{\mathrm{fcs}}(0)$.

Step 2. Evaluate L.

The core of our Lagrangean relaxation codes is the network simplex implementation MCF, described in Chap. 5, together with a column generation for the single-depot case and, therefore, also for $\mathrm{LR}_{\mathrm{fcs}}$ (6.3). We use a modified version of this implementation to solve the minimum-cost flow problems L_{fcd} (6.6). The initial restricted arc set consists for both relaxations of the user-defined unloaded trips.

Step 3. Compute subgradient.

First, let $\pi^{(k)}$ denote the k^{th} Lagrange multiplier for $\mathrm{LR}_{\mathrm{fcs}}$, and let $x^{(k)}$ denote some optimal solution attaining the value of $L_{\mathrm{fcs}}\big(\pi^{(k)}\big)$. If follows from Theorem 6.10 that

$$g^{(k)} = g^{(k)}(\pi^{(k)}) := \left(\Big(\big(x^d\big)^{(k)}\big(\delta^-(t^-)\big) \;-\; \big(x^d\big)^{(k)}\big(\delta^+(t^+)\big)\Big)_{t\in\mathcal{T}_d}\right)_{d\in\mathcal{D}}$$

is a subgradient for L_{fcs} at $\pi^{(k)}$. It easy to see that $\big(g_t^d\big)^{(k)} \in \{-1, 0, 1\}$. The interpretation of $g^{(k)}$ is as follows:

- If $\big(g_t^d\big)^{(k)} = 0$, $x^{(k)}$ is valid for the corresponding flow conservation.
- If $\big(g_t^d\big)^{(k)} = +1$, some vehicle of depot d enters the timetabled trip $t \in \mathcal{T}_d$, but none leaves it.
- If $\big(g_t^d\big)^{(k)} = -1$, some vehicle of depot d leaves the timetabled trip $t \in \mathcal{T}_d$, but none enters it.

If $g^{(k)}$ is used as the step direction then

- if $\big(g_t^d\big)^{(k)} = 0$, the Lagrange multiplier π_t^d remains unchanged, i. e., $\big(\pi_t^d\big)^{(k+1)} := \big(\pi_t^d\big)^{(k)}$;
- if $\big(g_t^d\big)^{(k)} = +1$, the Lagrange multiplier π_t^d is increased, the objective coefficients of each arc entering t^- is increased, and the objective coefficient of each arc leaving t^+ is decreased;
- if $\big(g_t^d\big)^{(k)} = -1$, the Lagrange multiplier π_t^d is decreased, the objective coefficients of each arc entering t^- is decreased, and the objective coefficient of each arc leaving t^+ is increased.

Second, let $\nu^{(k)}$ denote the k^{th} Lagrange multiplier for $\mathrm{LR}_{\mathrm{fcd}}$, and let $x^{(k)}$ denote some optimal solution attaining the value of $L_{\mathrm{fcd}}(\nu^{(k)})$. It also follows from Theorem 6.10 that

$$g^{(k)} = g^{(k)}(\nu^{(k)}) := \mathbb{1} - \Big(x^{(k)}\big(\delta^+(t^-)\big)\Big)_{t\in\mathcal{T}}$$

is a subgradient for L_{fcd} at $\nu^{(k)}$. It is also easy to see that $g_t^{(k)} \in \{\ldots, -2, -1, 0, 1\}$. The interpretation of $g^{(k)}$ is as follows:

- If $g_t^{(k)} = 0$, the timetabled trip t is serviced exactly once and $x^{(k)}$ is valid for the corresponding flow condition.
- If $g_t^{(k)} = 1$, the timetabled trip t is not serviced at all.
- If $g_t^{(k)} < 0$, the timetabled trip t is serviced by exactly $g_t^{(k)}$ vehicle schedules.

If $g^{(k)}$ is used as the step direction then

- if $g_t^{(k)} = 0$, the Lagrange multiplier ν_t and, thus, the objective coefficient of each arc in $(t^- \to t^+)$ remains unchanged;
- if $g_t^{(k)} = 1$, the Lagrange multiplier ν_t is increased and the objective coefficient of each arc in $(t^- \to t^+)$ is decreased;
- if $g_t^{(k)} < 0$, the Lagrange multiplier ν_t is decreased and the objective coefficient of each arc in $(t^- \to t^+)$ is increased.

Step 4. Stopping criteria.

First, we check whether $g^{(k)} = 0$. If this condition is satisfied, the corresponding $x^{(k)}$ is feasible as well as optimal, and we can STOP. Second, we check our iteration limit, denoted by N_1, i. e., if $k > N_1$ we STOP. Our standard value for N_1 is 100.

Step 5. Compute new step length.

Polyak [1967] shows how the step length can be chosen such that the sequence $L(u^{(0)})$, $L(u^{(1)})$, $\ldots$ converges to the optimum of the Lagrangean dual. However, our goal was rather to compute good lower bounds quickly than on satisfying some convergence criterion. Therefore, we focused our efforts only on performance improvements within our given iteration limit. The step length and the step direction play a key role here. Based on the parameter setting, we use one of the following step length rules:

A)

$$\sigma^{(k)} := \begin{cases} \frac{\sigma^{(k-1)}}{2}, & \text{if } L \text{ declines for } N_2 \text{ consecutive iteration(s)}, \\ \sigma^{(k-1)}, & \text{else} \end{cases}$$

with $\sigma^{(0)} := 10$ and a maximum failure parameter $N_2 := 2$.

B)

$$\sigma^{(k)} := \frac{\alpha^{(k)}\left(\overline{L} - L(u^{(k)})\right)}{\|g^{(k)}\|^2} \quad \text{with} \quad \alpha^{(k)} := \begin{cases} \frac{\alpha^{(k-1)}}{2}, & \begin{array}{l}\text{if } L \text{ declines for } N_2 \\ \text{consecutive iteration(s)},\end{array} \\ \alpha^{(k-1)}, & \text{else,} \end{cases}$$

with $\alpha^{(0)}$ such that $\sigma^{(0)}$ becomes 10, a maximum failure parameter $N_2 := 2$, and an upper bound $\overline{L}$ for L provided our schedule – cluster – reschedule heuristic, see Chap. 8.

Remember our two-stage objective: minimize the fleet size and, subordinate, the operational costs. For almost all of our test instances, the initial Lagrangean function $L_{\text{fcs}}(0)$ and $L_{\text{fcd}}(\nu^{(0)})$ provide already the exact value for the fleet size. Therefore, we concentrate our efforts to find an initial $\sigma^{(0)}$ that improves the operational costs.

It turned out that the initial step length is the only sensitive parameter of our subgradient methods. For each of our test instances, we have made several test runs to find out good starting values for $\sigma^{(0)}$. We are, however, not able to provide any reasonable rule for a good starting configuration. Thus, we have decided to use $\sigma^{(0)} = 10$ as default for our presentation. For our complete test set, this value was one of the best among all that we have tried out. We refer the reader to Kokott [1996] for a detailed description of these tests.

Step 6. Compute new step direction.

Following the key idea of bundle methods, e. g., see Crowder [1976], we use the following step direction:

$$\tilde{g}^{(k)} := 0.6g^{(k)} + 0.2g^{(k-1)} + 0.1g^{(k-2)} + 0.1g^{(k-3)}$$

(we set $g^{(-1)} := g^{(-2)} := g^{(-3)} := g^{(0)}$). This turned out to be a robust choice.

Chapter 7

Solving the LP Relaxation

We will describe in this chapter how the LP relaxation of (2.14) can be solved to optimality by means of column generation techniques. The standard column generation approach in the literature is based on generating and eliminating columns based on the reduced cost criterion. We propose here a new technique that is based on Lagrangean relaxations of the multicommodity flow model. The method, which we call *Lagrangean pricing*, activates the arcs of complete paths and not only individual arcs. In particular, it is not only possible, but *essential* that columns with positive reduced costs are generated. Lagrangean pricing has been developed independently at the same time by Fischetti and Toth [1996] and Fischetti and Vigo [1996] for solving the Asymmetric Traveling Salesman Problem and the Resource-Constrained Arborescence Problem, respectively.

Solving an MDVSP instance to optimality using LP based approaches requires to solve the LP relaxation to optimality. With Lagrangean pricing, it becomes possible to solve the huge linear programs that come up here. Therefore, we propose Lagrangean pricing as one of the basic ingredients of an effective method to solve this kind of problems to *proven* optimality.

In this chapter, the underlying ILP formulation is based on the contracted digraph D (2.10). Remember, the considered LP relaxation reads

$$\min \sum_{d \in \mathcal{D}} \sum_{a \in A_d^{\text{u-trip}}} c_a^d x_a^d \tag{7.1a}$$

subject to

$$\begin{aligned}
&(7.1\text{b}) & x\big(\delta^+(t)\big) &= 1, && \forall\, t \in \mathcal{T},\\
&(7.1\text{c}) & x^d\big(\delta^+(t)\big) - x^d\big(\delta^-(t)\big) &= 0, && \forall\, t \in \mathcal{T}_d \;\; \forall\, d \in \mathcal{D},\\
&(7.1\text{d}) & x^d\big(\delta^+(d)\big) &\geqslant \lambda_d, && \forall\, d \in \mathcal{D},\\
&(7.1\text{e}) & x^d\big(\delta^+(d)\big) &\leqslant \kappa_d, && \forall\, d \in \mathcal{D},\\
&(7.1\text{f}) & x &\geqslant 0.
\end{aligned}$$

Let $\nu \in \mathbb{R}^{\mathcal{T}}$, $\pi := (\pi^d \in \mathbb{R}^{\mathcal{T}_d})_{d \in \mathcal{D}}$, $0 \leqslant \beta \in \mathbb{R}^{\mathcal{D}}$, and $0 \leqslant \gamma \in \mathbb{R}^{\mathcal{D}}$ denote the dual multipliers associated with the constraints (7.1b), (7.1c), (7.1d), and (7.1e), respectively.

In the recent years, considerable research has gone into the design of pseudo-polynomial time approximation algorithms for multicommodity flow *feasibility* problems. Several papers have been written about this topic as, for instance, Leighton, Makedon, Plotkin, Stein, Tardos, and Tragoudas [1991], Plotkin, Shmoys, and Tardos [1991], and Klein, Plotkin, Stein, and Tardos [1994]. However, there are too few papers describing implementations of such algorithms and reporting about computational results thereof. Moreover, the results reported in Leong, Shor, and Stein [1993] and Borger, Kang, and Klein [1993] on rather small problem instances do not look encouraging from a computational point of view. It is therefore not clear whether approximation algorithms could substantially help solving the *optimization* problems that we investigate here. We will discuss such approximation algorithms in the last section.

7.1 Implementation Details

The instances of the MDVSP we encountered in practice have up to 70 million variables and 125 thousand equations. Ignoring the integrality stipulation, we obtain linear programs, which are way out of reach for even the best LP codes currently available, not to mention the fact that it is impossible to explicitly store such a large LP in todays computers.

We will show in this section how the LP relaxation (7.1) can be solved to optimality using Lagrangean pricing techniques. In particular, our implementation combines robust LP software, a minimum-cost flow code, and parts of the Lagrangean relaxations codes for the MDVSP. In our case, we use the CPLEX Callable Library (CPLEX [1995]) and our network simplex code MCF.

In a first try, we have tried to apply a standard column generation and elimination technique based on the reduced cost criterion, see Sect. 7.1.1. With such a standard approach, however, only rather small instances have been solved successfully. *Stalling* in the objective value occurred for larger instances. Within the column generation process, many new columns have been generated, but none of them could help to improve the objective value. Moreover, almost all active columns have reduced costs near to zero and, therefore, none of them could be eliminated resulting in too large RLPs.

The new Lagrangean pricing techniques can help to improve the column generation process. We will describe Lagrangean pricing in Sect. 7.1.2. The right composition of all employed ingredients is given in Sect. 7.1.3.

7.1.1 Column Generation

The basic idea of a column generation is to provide only a relatively small subset of the columns, which includes some optimal basis, and to ignore all the other ones. One starts with a subset of columns that, in addition, includes at least some primal feasible basis. The reduced LP, defined by this subset of columns, is called *restricted* LP (RLP). It is the task to solve a sequence of RLPs until it is proved that the last RLP contains the columns of some basis, which is optimal for the complete LP. The global optimality condition of an RLP is described below.

An exact description of the column generation is as follows. Assume that we have already determined a subset $\tilde{A} \subset A$ such that $\tilde{A}$ includes some (primal) feasible solution. Consider the RLP including only the columns according to this subset $\tilde{A}$. In addition, assume that a primal feasible starting basis is determined. In general, the RLP is resolved to optimality, but it is sufficient to perform only some (primal) simplex iterations. Let $\tilde{\nu}$, $\tilde{\pi}$, and $\tilde{\gamma}$ denote the value of the dual multipliers associated with the last basis of the current RLP. For notational simplicity, let $\tilde{\nu}_d := 0$ and $\tilde{\pi}_d^d := 0$, for all $d \in \mathcal{D}$, denote *artificial* variables. We compute for each variable the reduced costs

(7.2)
$$\overline{c}_{ij}^d := c_{ij}^d - \tilde{\nu}_i - \tilde{\pi}_i^d + \tilde{\pi}_j^d - \left\{ \begin{array}{c} 0 \\ \tilde{\beta}_d - \tilde{\gamma}_d \end{array} \right\} \quad \forall \ (i,j) \in A \text{ such that } i \left\{ \begin{array}{c} \notin \\ \in \end{array} \right\} \mathcal{D}.$$

Note that $\overline{c}_a \geqslant 0$ for all active columns $a \in \tilde{A}$ if the last RLP was solved to optimality. If $\overline{c}_a \geqslant 0$ for all $a \in A$, the global optimality of the current basis is proved and we can stop. Otherwise, we search for some (inactive) variables $a \in A \setminus \tilde{A}$ and generate their corresponding columns.

Standard column generation schemes generate only those columns that violate the reduced cost criterion $\overline{c} \geqslant 0$, i. e., variables with negative reduced costs. But, as we will see below, it turned out that adding also columns with nonnegative reduced costs may be advantageous. Having selected the variables that become active, $\tilde{A}$ and the corresponding RLP are redefined appropriately. The enlarged LP is reoptimized or a limited number of simplex iterations is performed, and we iterate until we prove optimality, i. e., $\overline{c}_a \geqslant 0$.

Obviously, to achieve any progress, at least one variable having negative reduced cost must be activated between two consecutive RLPs. Tests in practice have shown that it is impossible to generate all inactive columns with negative reduced costs since the next RLP gets far too large and cannot be handled at all. Therefore, we restrict the number of new arcs to some parameter controlled limit. This limit ranges from 200 to 3000 variables for each depot, depending on the problem size.

For the standard column generation scheme, we use the original pricing rule due to Dantzig [1963]. We select the variables with most invalid reduced costs as candidates to become activated. With this approach, it is also possible to prove the global optimality of some RLP, provided that the last RLP has been solved to optimality and includes some optimal basis. We have also tried to use more advanced pricing rules such as Devex pricing proposed by Harris [1973] and steepest-edge pricing proposed by Goldfarb and Reid [1977]. Similar to Dantzig's rule, these rules generate only columns with negative reduced costs, but we could not observe better computational results. Therefore, we have rejected those advanced pricing rules and apply only Dantzig's rule. Lagrangean pricing.

To avoid that the RLPs become too large, we must also remove obsolete columns in each iteration of the column generation process. All columns whose reduced costs exceed some predefined parameter controlled positive threshold are therefore eliminated.

7.1.2 Lagrangean pricing

In a first version, we have tried to solve large MDVSP instances using only standard column generation and elimination schemes. But this approach failed. One main obstacle is the completely degenerate LP relaxation. A second reason for the difficulties is as follows: The standard column generation scheme activates only variables with negative reduced cost. These variables can locally promise some progress in the objective value, but it is not clear whether they may have any influence on the solution and the objective value without an interaction with some other related nonactive variables. Therefore, we came up with the idea that the nonactive variables should be not only evaluated alone by its reduced costs, but also in interaction with all the other active and inactive variables. However, how can this be done efficiently? We have to find a method that determines good (nonactive) variables that may give progress in the objective value as best as possible. To use the information already compiled within the previous RLPs, this method should also use dual information as pricing methods do. It may also be a good idea to invoke also Lagrangean relaxation techniques that turned out to give good approximations of our hard solvable LP relaxations.

The answer to these questions is *Lagrangean pricing*: The inner minimization problems L_{fcs} (6.4) and L_{fcd} (6.7) of the presented Lagrangean relaxations LR_{fcs} and LR_{fcd} can be solved efficiently, even for the complete variable set, and give excellent approximations of the LP relaxation. So, we evaluate the inner minimization problems $L_{\text{fcs}}(\tilde{\pi})$ and $L_{\text{fcd}}(\tilde{\nu})$. Remember, $\tilde{\nu}$ and $\tilde{\pi}$ denote the value of the dual multipliers associated with the flow conditions (7.1b) and the flow conservations (7.1c) of the last basis of the current RLP.

Obviously, both relaxations approximate the LP relaxation with all active and inactive variables, use dual information given by the last RLP, are based on good relaxations of the LP relaxation, and can be evaluated efficiently. We still have to show how good nonactive variables can be determined. The solution of each inner minimization problem can be interpreted as a set of vehicle schedules that seem to be advantageous for the given shadow prices of the current RLP relaxation. In the case of the Lagrangean relaxation L_{fcd}, these vehicle schedules may include unloaded trips of different depots. Consider all the vehicle schedules defined by the optimal solutions attaining the values of $L_{\text{fcs}}(\tilde{\pi})$ and $L_{\text{fcd}}(\tilde{\nu})$. Each still nonactive variable according to some unloaded trip of some of these vehicle schedules determines a candidate to become active.

7.1.3 The Basic Ingredients

We have made many computational experiments to find out the right mixture of the techniques presented above. The basic ingredients, each being indispensable to solve large-scale instances at all, are as follows:

Initial RLP relaxation: The initial RLP should contain at least some primal feasible solution yielding a value as close to the LP optimum as possible. A very efficient way to heuristically determine some solution is a *schedule – cluster – reschedule heuristic* (SCR). A faster method is a *nearest depot heuristic* (ND), which assigns each timetabled trip to some depot with the smallest sum of the pull-out and pull-in costs. This kind of opening heuristic, however, yields rather poor starting points and, theoretically, can produce arbitrarily bad solutions, see Chap. 8. Nonetheless, we will see in Chap. 12 that the performance results are, on the average, comparable regardless whether we start with the ND or the more sophisticated SCR heuristic.

As soon as each timetabled trip is assigned to some depot, the problem decomposes into $|\mathcal{D}|$ independently solvable single-depot subproblems. We solve for each depot its single-depot instances according to all its heuristically assigned timetabled trips. Each unloaded trip that corresponds to some basic variable becomes active and its column is generated for the initial RLP. Thus, the first RLP includes at least the feasible solution defined by the union of the solutions of all subproblems together. A further idea is to use the union of all columns generated by any primal (opening heuristic) and dual (Lagrangean relaxation) method. Unfortunately, we have not tested such a combination of different heuristics.

The Workhorses: Minimum-Cost Flow and LP Solving the LP relaxation with our approach exactly, requires the efficient solution of minimum-cost flow problems and linear programs at several steps: The minimum-cost flow problems stem from single-depot subproblems and Lagrangean relaxations, the LPs are RLPs. All minimum-cost flow problems have been solved with MCF. The linear programs have been solved with the primal as well as the dual simplex solver of the CPLEX Callable Library, version 4.0.9. CPLEX turned out to be a reliable and robust method for our degenerate (R)LP problems.

For our computations, an important feature of CPLEX 4.0 is the new and more gentle perturbation method. In previous version of CPLEX, the bounds of all variables have been relaxed when perturbing a problem. This perturbation approach led often to numerical problems when we have solved our test instances. With the current version of CPLEX, only all basic variables are perturbed whenever the perturbation starts. As soon as some nonbasic variable has been selected to become basic it will also be perturbed if not already done in some previous iteration. This simple alteration of the perturbation strategy has significantly improved the efficiency of our implementation for large MDVSPs.

The column generation is divided into two phases: First, a *Lagrangean phase* where we apply standard and Lagrangean pricing, and, second, a *standard phase* in which we apply only the standard column generation approach.

Lagrangean phase: This phase precedes always the standard phase and is applied as long as the objective value declines between two consecutive RLPs at least by some predefined parameter controlled threshold (10.0 is used as default). The last basis of the last RLP is always neglected, and each RLP is reduced by LP preprocessing. The columns of each RLP obtained in this phase are, at least for large MDVSPs, far too many for the primal simplex solver. We use here the dual simplex solver. We have also tried to use CPLEX's primal-dual logarithmic barrier solver. It turned out, however, that numerical problems often prevent the barrier solver from proceeding.

As long as there is a sufficiently large gap between the optimal LP value and the value of the current RLP, the Lagrangean phase works well. However, stalling occurs when the current RLP value approaches the LP optimum. This phase is unable to converge to an optimal variable set: Although the objective has been become almost optimal, the standard column generation between two consecutive RLPs finds always thousands up to millions of unloaded trips that do not satisfy the reduced cost criterion. This effect is maybe a result of neglecting always the last basis (i. e., all dual information) of the previous RLP, but we cannot provide any other reasonable explanation.

Thus, we came up with the idea to use at this point only the standard column generation scheme: We switch to the standard phase when the objective progress becomes too small and, therefore, some "approximation of optimality" has been reached.

Standard phase: When we start this phase, we believe that our current RLP contains some almost optimal basis of the complete LP relaxation. The occurring RLPs are now solved with the primal simplex solver and each RLP starts with the last basis of the preceding RLP. This approach iterates until the (global) optimality of some RLP can be proved with the reduced cost criterion.

7.2 Approximation Algorithms

First of all, one has to realize that the term "approximation" stands for an approximation of feasibility problems, i. e., it is the task to ask whether there exists some feasible or almost feasible solution or not. But we consider a minimization problem that can only be solved by such approximation algorithms if we add an extra inequality bounding the maximum allowed objective value that has to be adopted within a binary search procedure.

Let the commodities of an approximation problem be indexed by k. The basic idea of approximation algorithms for multicommodity flow problems is closely related to Lagrangean relaxation, see Plotkin, Shmoys, and Tardos [1991]: At first, some initial flow x

is arbitrarily determined such that each single-commodity flow x^k, for all commodities k, satisfies at least the individual flow conservation constraints and individual flow bounds. The coupling constraints – the flow conditions and the upper bound on the objective value as in our case – will be most probably violated. Then some optimization problems over the individual constraints are repeatedly solved to find directions in which the violation of the coupling constraints can be decreased. To do this, penalties for the coupling constraints corresponding to Lagrange multipliers are defined. Constraints being more violated get larger penalties than constraints being less violated. The coupling constraints are relaxed by a Lagrangean relaxation approach and the subproblem L is evaluated for the given penalties. The heuristical idea here is that large penalties tend to imply that the resulting optimal point $\tilde{x}$ (attaining the value of L) improves the corresponding invalid constraints. For a properly small number σ, the new flow vector is set to $x := (1 - \sigma)x + \sigma\tilde{x}$. This procedure is repeated until some given degree of approximation is reached or the problem is determined to be infeasible.

Plotkin et al. remark that Lagrangean relaxation approaches are often used to obtain **empirically** good algorithms for solving linear programs. For instance, our Lagrangean relaxation are such approaches. Unlike those methods, they give a rule for adjusting the Lagrange multipliers such that a run time analysis proves a very favourable **theoretical** performance. The interesting result for approximation algorithms is that the run time is not as sensitive to the number of commodities as one might expect: $\mathcal{O}(k \cdot \log k)$ for a randomized version and $\mathcal{O}(k^2 \cdot \log k)$ for a deterministic version.

Leong, Shor, and Stein [1993] and Borger, Kang, and Klein [1993] report on computational investigations and comparisons for the concurrent multicommodity flow problem, the latter for problems with unit commodity demands and unit edge capacities. Their results provide support for the theoretical run time behaviour. These investigations are based on randomly generated test sets, all with less than 1000 nodes and less than 3480 edges. Only one relatively small real-world test problem having 49 nodes, 260 arcs, and 585 commodities is presented in Leong et al. For problems with a small number of commodities (up to 40 – 50), the approximation codes are outperformed by a special purpose simplex code. Moreover, the same clearly holds for the interesting real-world problem. Leong et al. call this behaviour "an anomaly".

Unfortunately, these computational results did not encouraged us to use such approximation algorithms: First, for problems with a small number of commodities and especially for the interesting real-world problem, the approximation code was clearly outperformed. This is very important to us since the number of commodities (resp., depots) is small for our problems, and we are more interested in the empirical than in the worst case performance. Second, none of the inventors of approximation algorithms has computationally attacked **minimum-cost** multicommodity flow problems with this kind of approach. There are also some further obstacles that put us off from approximation algorithms:

- MDVSPs have no source and sink nodes. If we consider only the flow conservations and the individual lower and upper bounds for a nonnegative cost function, the optimal solution is obviously zero. However, the flow is enforced by the flow conservations for each timetabled trip. It seems to be an open problem how the approximation method can be adopted to equations avoiding numerical difficulties and convergence problems.

- In each main iteration of the approximation method, the flow is always partly rerouted. At last, the solution might have significantly more nonzero values than the solution of the LP relaxation. This may badly influence the performance of a branch-and-cut approach.
- Our minimization problem must be solved by binary search. Although we can apply quite good methods to compute a lower bound c_L and an upper bound c_U efficiently (see Chaps. 6, 8, and 12), we believe that $\mathcal{O}(\ln(c_U - c_L))$ approximation problems that have to be solved are still far too much since, for our test instances, the number of approximation problems can easily grow up to more than 20.

Chapter 8

Primal Heuristics

All methods in the literature about solving large real-world MDVSP systems are, to the best of our knowledge, heuristics. The core of our thesis, however, is to solve large MDVSPs exactly. Nonetheless, the exact branch-and-cut method presented here requires primal opening and improvement heuristics that help to reduce the branch-and-bound tree and, hence, accelerate the solution process significantly. We have implemented three heuristics that we will explain in this section. First, a cluster first – schedule second heuristic that is based on nearest depot approach, denoted by ND. Second, a schedule – cluster – reschedule heuristic based on the Lagrangean relaxation $\mathrm{LR}_{\mathrm{fcs}}$, denoted by SCR. Third, an LP based iterative rounding heuristic, called LP-plunging, that exploits information compiled in an (R)LP and its optimal solution. In addition, we briefly describe the mathematical ingredients of the assignment approach implemented in HOT II.

Because it is $\mathcal{NP}$-hard to find feasible solutions for capacitated problems, see Theorem 3.10, Remark 3.11 gives some justification that our opening heuristics ND and SCR as well as the assignment approach of HOT II consider depot capacities only heuristically.

8.1 Cluster First – Schedule Second

The idea of cluster first – schedule second approaches (CF-SS) is as follows: First, each timetabled trip is assigned to exactly one depot (cluster part) decomposing the problem into $|\mathcal{D}|$ independently solvable single-depot instances. Second, each of these single-depot problems is efficiently solved to optimality (schedule part) with a network flow algorithm.

We have implemented a simple version based on a nearest depot heuristic that considers depot capacities heuristically. Of course, the following procedure could be further improved, but this is not our goal.

(8.1) Algorithm. ND heuristic for the MDVSP.
Input: MDVSP instance with depot lower and upper capacities.
Output: Set of vehicle schedules that possible violate depot capacities.

1. Assign each trip $t \in \mathcal{T}$ to the depot $\arg\min\{c^d_{(d,t)} + c^d_{(t,d)} | \ d \in G(t)\}$ and solve each resulting single-depot problem to optimality using MCF. Let all depots be unexamined.
2. If all depot capacities are satisfied, we have generated a feasible solution and stop.
3. If there exist unexamined depots with violated lower or upper capacities, select some depot l with this property. Otherwise, we stop with an infeasible solution violating some depot capacities.
4. If the upper capacity of l is exceeded by the amount of $\hat{\kappa}_l$, we heuristically select $\hat{\kappa}_l$ vehicle schedules whose timetabled trips can be assigned to other depots with free capacity as cheaply as possible. If at least one circulation has been shifted to a free depot, we let all depots be unexamined and continue with 6. Otherwise, we go to 3.
5. If the lower capacity of l is violated by the amount of $\hat{\lambda}_l$, we heuristically search for vehicle schedules or blocks that have been assigned to other depots, but could also be serviced by l. If at least one circulation has been shifted to the depot l, we let all depots be unexamined and continue with 6. Otherwise, we go to 3.
6. The new resulting assignment of timetabled trips to depots is depot wise scheduled to optimality, and we go to 2.

In the worst case, ND may theoretically produce arbitrary bad solutions. Consider the following small uncapacitated instance with $\mathcal{D} := \{\mathrm{r}, \mathrm{g}\}$, $\mathcal{T} \neq \emptyset$, and $A^{\text{d-trip}}_{\mathrm{r}} := A^{\text{d-trip}}_{\mathrm{g}} := \{(i, i+1) | \ 0 < i < |\mathcal{T}|\}$; the weight of each unloaded trip is set to zero, except the weights of $\{(\mathrm{r}, i), (i, \mathrm{r}) | \ i \text{ odd}\} \subset A^{\text{d-trip}}_{\mathrm{r}}$ and $\{(\mathrm{g}, i), (i, \mathrm{g}) | \ i \text{ even}\} \subset A^{\text{d-trip}}_{\mathrm{g}}$ that are all set to 1. ND assigns each timetabled trip with an even number to the depot g and with an odd number to r. The scheduling part cannot use any of the possible dead-head trips for this assignment and comes up with a solution using exactly $|\mathcal{T}|$ vehicle schedules. The worst case ratio, however, is $\frac{|\mathcal{T}|}{1}$ since the optimal solution is exactly one vehicle schedule if all timetabled trips are serviced either by g or r.

8.2 Schedule – Cluster – Reschedule

The heuristic that we describe in this section is currently installed in BERTA of the Berliner Verkehrsbetriebe (BVG) and is used in Berlin for bus and tram scheduling. It is also installed in MICROBUS 2 of the IVU GmbH, Berlin.

The schedule – cluster – reschedule approach is based on the following idea: There is a natural composition of schedule first – cluster second (SF-CS) and CF-SS heuristics since the output from the first one can obviously be used as the input for the second resulting in a schedule – cluster – reschedule heuristic (SCR).

Our version of SCR considers depot upper capacities heuristically, but considers no depot lower bounds. It is based on the function $L_{\text{fcs}}(\pi)$, which was presented in Chap. 6 for the Lagrangean relaxation with respect to the flow conservations and can be applied to any arbitrarily chosen Lagrange multipliers π. Currently, we apply SCR always with $\pi := 0$. However, it is also possible to embed SCR in the subgradient method for LR_{fcs}. Let π be given, we perform the following procedures:

Scheduling.

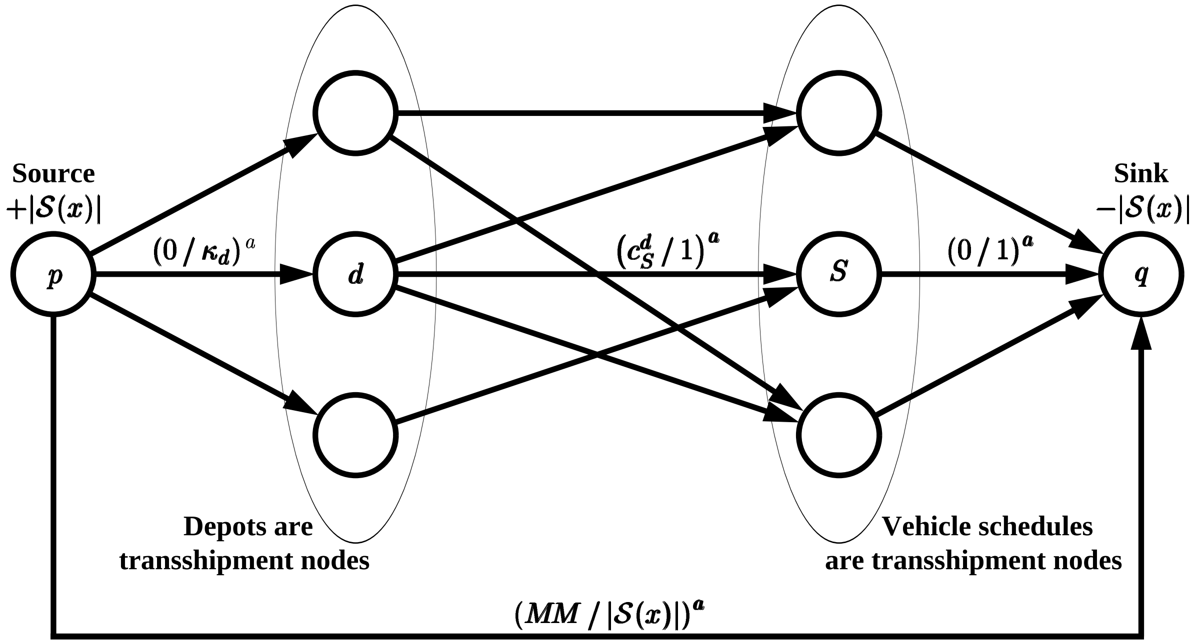
We start by evaluating $L_{\text{fcs}}(\pi)$ with the network simplex code MCF. Let x be some optimal solution for $L_{\text{fcs}}(\pi)$ provided by MCF, and let $\mathcal{S}(x) \subseteq 2^A$ denote the set of all vehicle schedules defined by x. Note that each $S \in \mathcal{S}(x)$ represents all unloaded trips of its corresponding vehicle schedule.

Vehicle demand estimation.

In the following, we try to give an estimation of the necessary fleet size for x: Consider a fixed vehicle schedule $S \in \mathcal{S}(x)$. Let $\tilde{S} \subset A$ denote all arcs with zero reduced costs connecting two subsequent timetabled trips of S, being a pull-out trip that enters the first timetabled trip of S, or being a pull-in trip that leaves the last timetabled trip of S. Depending on our parameter settings, we allow, in addition, also to consider arcs in $\tilde{S}$ with positive reduced costs being smaller than a predefined small value. Let $G(S) := \{d \in \mathcal{D} |\ \max_{l \in \mathcal{D}} |S \cap A_l| = |S \cap A_d|\}$ denote those depots fitting best for S.

All vehicle schedules of $\mathcal{S}(x)$ are heuristically subdivided to the depots such that the depot capacity constraints are satisfied as much as possible. We construct the following minimum-cost flow problem as shown in Fig. 8.1 and solve it with MCF. The node set is defined by $\mathcal{D} \cup \mathcal{S}(x) \cup \{p, q\}$ with source node p, sink node q, and transshipment nodes $\mathcal{D} \cup \mathcal{S}(x)$. The arc set consists of the arcs (p, d) connecting the source node p with each depot node $d \in \mathcal{D}$, (S, q) connecting each vehicle schedules $S \in \mathcal{S}(x)$ with the sink node q, an arc connecting the sink with the source node, and all arcs $\bigcup_{S \in \mathcal{S}} G(S) \times \{S\}$ connecting each vehicle schedule with its fitting depots. The arc costs and upper capacities are given as shown in Fig. 8.1. The weights $c_S^d := \left(|S| - |\tilde{S} \cap A_d^{\text{u-trip}}|\right) \cdot M + \sum_{a \in \tilde{S} \cap A_d^{\text{u-trip}}} c_a^d$ approximate the costs if the timetabled trips of the vehicle schedule S (or a proper subset thereof) are assigned to depot d. M denotes our used capital costs of a vehicle, and the term $|S| - |\tilde{S} \cap A_d^{\text{u-trip}}|$ tries to measure those parts of S that cannot be assigned to the depot d and, should this situation arise, must (expensively) be assigned to other depots using possibly more vehicles.

The interpretation of this minimum-cost flow problem is as follows: It is the task to send exactly $|\mathcal{S}(x)|$ units of flow from the source to the sink node as cheaply as possible. Since the costs of the arc (p, q) dominate the sum of all the other arcs together, as much flow as possible is routed through the transshipment nodes $\mathcal{D} \cup \mathcal{S}$, i. e., as many vehicle schedules



[a](cost/capacity) of an arc. κ_d is set to ∞ for uncapacitated depots. MM denotes a huge number dominating the sum of all the other objective coefficients.

Figure 8.1: Minimum-cost flow problem for vehicle demand estimation

S as possible will be assigned to one of its depots $G(S)$. This assignment, however, must consider the given depot capacities in any case since only κ_d units of flow can be routed through each node $d \in \mathcal{D}$. On the other side, at most one unit of flow can be routed on each node $S \in \mathcal{S}(x)$, i.e., it can only be assigned at most once. It is easy to see that the maximum flow through $\mathcal{D} \cap \mathcal{S}(x)$ is routed as cheaply as possible with respect to the cost coefficients c_S^d.

With the optimal solution of this minimum-cost flow problem, we receive an estimation of the necessary fleet size for each depot according to x. Moreover, the flow value of (p, q) gives some estimation of the quality of x by the number of vehicle schedules that presumable cannot be assigned to depots without violating depot capacities. Last, but not least, it helps us to derive a clustering from x as described in the next two procedures.

Minimization of violated flow conservations.

We invoke a heuristic that tries to modify each vehicle schedule $S \in \mathcal{S}(x)$ by exchanging arcs such that the modified solution satisfies the depot capacities as much as possible and violates fewer flow conservations. It is motivated by the fact that we presumable can preserve more vehicle schedules of the modified solution than of the original one when the timetabled trips will be serviced by one of its fitting depots.

For each $S \in \mathcal{S}(x)$, we determine its *favourite* depot, denoted by $dep(S)$ and defined by d, if the flow value of the arc (d, S) was set to one, or defined by $\arg\max_{d \in \mathcal{D}} |S \cap A_d^{\text{u-trip}}|$ in the case that S was not assigned to any depot. By exchanging unloaded trip $a \in S$ by existing counterparts $\tilde{a} \in \tilde{S}$, we heuristically try to determine parts of or complete blocks or better vehicle schedules whose timetabled trips can together be assigned to the same depot. It is the task to perform these exchanges such that

- the resulting blocks and vehicle schedules are as large as possible and violate therefore as few flow conservations as possible,
- the objective value is declined minimally, and
- as much blocks and vehicle schedules stemming from S have been assigned to its favourite depot $dep(S)$ as possible.

Clustering and rescheduling.

Each trip $t \in \mathcal{T}$ is now assigned to that depot providing the unloaded trip that services t in our modified solution, and the CF-SS heuristic applies.

Tabu Search.

Whenever not all timetabled trips of a modified vehicle schedule can be completely assigned to the same depot, we forbid the use of those user-defined dead-head trip that connect two timetabled trips that are serviced in sequence by some $S \in \mathcal{S}(x)$, but have been assigned to two different depots. As long as new arcs have been forbidden, we restart with the scheduling procedure using the smaller network without the forbidden arcs.

8.3 LP-Plunging

Our real-world MDVSP instances exhibit in practice a nice "almost-integrality property": solutions x of the LP relaxation (7.1) or an RLP include few fractional variables. It is often the case that x is already integral or there exists some integral solution yielding (almost) the same objective value. Moreover, the gap between the optimal LP or RLP value and its optimal integer value is often small or zero. This property of real-world problems was also observed and described by Forbes, Holt, and Watts [1994]. **LP-plunging** makes use of this property by iteratively rounding up and fixing components of the LP solution and reoptimizing the enlarged LP.

Given an LP relaxation or an RLP and a nonintegral feasible vector x. Let $\Delta \in (0.5, 1.0)$ denote some threshold value for which all fractional variables having a value within $(\Delta, 1)$ are rounded up and fixed to one, and let $\alpha \in (0.5, 1.0)$ denote some shrink factor for Δ. The standard values for Δ and α are 0.95 and 0.9. As long as the current x is nonintegral and the current (R)LP is primal feasible, the following steps are performed:

1. All variables $x_a^d \in (\Delta, 1)$ are rounded up and fixed to 1.
2. If no variable was fixed to 1 and if $\alpha\Delta$ is still greater than 0.5, we reset $\Delta := \alpha\Delta$ and go to 1. Otherwise, the value of every fractional variable is not larger than 0.5, and we fix the first variable to 1 yielding the largest fractional value.
3. Logical implications are performed, i. e., for each variable x_{ij}^d being fixed to one, we fix the variables of all arcs $(\delta^+(i) \cup \delta^-(j)) \cap A_d^{\text{u-trip}}$ and $(\delta(i) \cup \delta(j)) \setminus A_d^{\text{u-trip}}$ to zero.
4. The LP enlarged by the variable fixings is reoptimized with the dual simplex algorithm.

If the LP-plunging succeeds, the clustering defined by the last (integral) x is depot-wise rescheduled to optimality using all possible unloaded trips of each depot. From this point of view, LP-plunging can also be viewed as a SCR heuristic that is based on the LP relaxation.

Since the restricted column set of an RLP generally includes only a small part of $A^{\text{u-trip}}$, the LP-plunging generates in many cases only poor or infeasible integer solutions. If this is the case, we enlarge the current RLP parameter controlled by inactive columns (such that the probability to find a better integer solution is presumably increased, but the dual feasibility of the optimal basis of the RLP is not destroyed and the main memory limit of the workstation is not exceeded) and apply the LP-plunging a second time.

8.4 Vehicle Scheduling in HOT II — Hamburger Optimization Technique

We shall now describe another SF-CS approach that has been developed, implemented, and successfully employed in practice at several German and international transportation companies: the *sensitivity analysis module* (SAM) and the *vehicle scheduling module* (VSM) of the HOT II system of the Hamburger Hochbahn AG and the HanseCom GmbH, Hamburg. This section is only focused on the mathematical details, the advantages, and the weakpoints of the vehicle scheduling in HOT. For a complete description of the HOT system, we refer the reader to the articles of Daduna, Mojsilovic, and Schütze [1993] and Daduna and Mojsilovic [1988].

It is an important property of HOT that only dead-head trips having a maximum predefined duration ranging between 40 and 120 minutes are considered. The concept of pull-in-pull-out trips is not used. Therefore, SAM is designed to build blocks, but not complete vehicle schedules. It includes procedures that, for a given set of timetabled trips, build blocks heuristically without considering pull-out and pull-in trips and interactively try to modify iteratively the timetable data such that the current solution can be improved with respect to the number of blocks. The output of SAM is used as the starting point for VSM. Its main mathematical part consists of distributing the blocks to depots such that operational costs are minimized and depot capacities are satisfied as good as possible.

SAM includes the scheduling part of SF-CS: It is formulated as a depot-independent assignment problem being based on the cost matrix with coefficients

$$c_{ij} \begin{cases} \in [0, \ll M) & \text{if it is possible to link the timetabled} \\ & \text{trips } i \text{ and } j \text{ within some maximum pre-} \\ & \text{defined turning time,} \\ = M & \text{otherwise,} \end{cases}$$

with a sufficiently large M dominating the sum of all cost coefficients of feasible links.

We have already discussed in Chap 2.1 that a problem formulation like in HOT cannot handle depot groups correctly. Therefore, before the cluster part in VSM can be applied, the resulting blocks of SAM must be splitted into smaller parts such that each component becomes feasible with respect to the depot groups.

The assignment problem is solved with the Hungarian method (for a description see, e. g., Ahuja, Magnanti, and Orlin [1993], page 471). The heuristic idea here is to minimize the number of blocks, and one hopes that the number of vehicle schedules is simultaneously minimized. We have also shown in Chap 2.1 that, in general, these two objectives do not exactly coincide: A solution with the minimum number of blocks does not imply a solution with the minimum number of vehicle schedules, and vice versa

The somewhat incorrect handling of depot groups together with the fact that blocks are generated without a consideration of all pull-in-pull-out trips are the reasons why the assignment approach of HOT does not necessarily produce optimal solutions.

The scheduling part is followed by a so-called sensitivity analysis. For critical time periods as rush hours, the system interactively tries to modify the timetable data within reasonable bounds such that the number of blocks can be further reduced, but the level of service keeps constant. More precisely, the system iteratively offers parameter controlled possible modifications of the departure times of timetabled trips and the delay buffers of dead-head trips such that inadmissible links become feasible and, in each step, two existing blocks can be merged together. The heuristic motivation to reduce the number of blocks is the same as above: Such a reduction may also possibly reduce the number of vehicle schedules. This approach, however, can also fail because the number of vehicle schedules can increase: Figure 8.2 shows a similar counterexample as given in Fig. 2.10, but with a modified end time of block"b" and a modified starting time of "c".

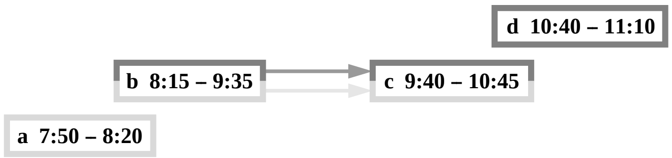


Figure 8.2: Counterexample to the heuristic motivation of sensitivity analysis.

Let us assume that the link between the blocks “b” and “c” is inadmissible as long as the timetable data keep unchanged. Obviously, the blocks “a” and “c” in sequence as well as the blocks “b” and “d” in sequence can each be serviced by one vehicle, i. e., two vehicles can service all blocks and thus all timetabled trips. But if the linking conditions are somewhat relaxed such that it becomes feasible to link “b” with “c” to a new block “b→c”, an additional vehicle becomes obviously necessary. Nevertheless, the sensitivity analysis seems to work very well in practice as indicated by the vehicle given in Daduna, Mojsilovic, and Schütze [1993].

At last, clustering in done in VSM. First, it is checked whether each block can be assigned to some depot or whether there are infeasible combinations of depot groups. In the latter case, such a block has to be splitted such that each part can be assigned to some depot. In the worst case, an additional vehicle becomes necessary. Considering depot capacities heuristically, each block is assigned to some valid depot such that the costs for the pull-out and pull-in trips are as small as possible.

Although some parts of the vehicle scheduling in HOT do not model the real problem exactly, all weakpoints can be repaired heuristically. On the other side, some degrees of freedom get lost resulting in suboptimal solutions. In Grötschel, Löbel, and Völker [1997], we have already obtained savings compared to the solutions of HOT that indicate, on the average, reductions of the fleet size of about 3 % and of the operational costs of about 10 %. From a practical point of view, the scheduling in HOT is nevertheless an efficient primal heuristic with a quite satisfiable solution quality.

Chapter 9

Polyhedral investigations

In this chapter, we give some results concerning our polyhedral investigations of the MDVSP. Readers who are orientated practically rather than theoretically may possibly wish to continue with the next chapter.

Polyhedral investigations play a basic role in solving many combinatorial optimization problems as, for instance, the travelling salesman problem. The thesis of Thienel [1995] gives a computational study for various problems that are solved with a polyhedral cutting plane approach.

Our early theoretical investigations of the MDVSP started with a characterization of the facial structure of its 0/1-polytope. In the course of our algorithmic developments and computational investigations in solving the MDVSP, however, it turned out soon that we are faced with completely different obstacles than finding (violated) inequalities of the integer polytope. For the few problems of our test set that could not be solved to proven integer optimality using only our column generation together with LP-plunging, we could not successfully apply a cutting plane approach. Our separators are able to determine violated inequalities for fractional LP solutions, but using them as cutting planes gives rise to the following two problems: First, the enlarged LP relaxations become harder such that they could often not be resolved to optimality for large instances. Second, if reoptimization succeeds, the flow is always completely rerouted with the same costs. Finally, when no more new violated cuts could be determined, we must start branch-and-bound with much harder LPs. It turned out that it is for our test set better to use only branch-and-bound without cutting planes.

Although cutting planes do not contribute to the solution of our problems, we will, for the sake of completeness, at least give the theoretical results of our polyhedral investigations, but do not describe the separation routines that we have implemented. We start by defining the polytope associated with the MDVSP and give the results concerning the polytope's dimension. We continue with the results concerning trivial inequalities $x_a^d \geqslant 0$ and introduce new kinds of valid inequalities for the uncapacitated case. Last but not least, we give an extended version of minimum cover inequalities that are valid for the capacitated version of the MDVSP polytope.

9.1 The MDVSP Polytope

By $P_{\text{MDVSP}} := \text{conv}\left\{x \in \mathbb{R}^A \,|\, x \text{ satisfies (2.14b) – (2.14f)}\right\}$ we denote the 0/1-polytope associated with the incidence vectors of feasible solutions of the MDVSP. Without lost of generality, we assume in this chapter $\mathcal{D} := \{1, \ldots, |\mathcal{D}|\}$.

Since the feasibility problem of the (capacitated) MDVSP is $\mathcal{NP}$-complete, see Theorem 3.10, it is also $\mathcal{NP}$-complete to determine the dimension of P_{MDVSP} and impossible to make any statements on its facial structure. Therefore, our polyhedral investigations have been mainly concentrated on the uncapacitated case. Our first result is about the equality set of P_{MDVSP} for uncapacitated problems.

(9.1) Lemma. The equality set $\text{eq}(P_{\text{MDVSP}})$ for uncapacitated problems is given by the flow conditions (2.14b) and the flow conservations (2.14c).

Proof: We have to show that each valid equation $e^{\mathrm{T}}x = f$ is a linear combination of (2.14b) and (2.14c). For convenience of notation, we define for each $d \in \mathcal{D}$ artificial scalars τ_d and π_d^d and set them to zero. Given some valid equation $e^{\mathrm{T}}x = f$, we have to show that there exists vectors $\tau \in \mathbb{R}^{\mathcal{T}}$ and $\pi := \left(\pi^d \in \mathbb{R}^{\mathcal{T}_d}\right)_{d \in \mathcal{D}}$ such that $\overline{e}_{ij}^d := e_{ij}^d - \tau_i - \pi_i^d + \pi_j^d = 0$, for all $(i,j) \in A_d^{\text{u-trip}}$ and all $d \in \mathcal{D}$, and $\overline{f} := f - \tau^{\mathrm{T}}\mathbb{1} = 0$. We set $\pi_i^d := -e_{di}^d$ for all $i \in \mathcal{T}_d$ and all $d \in \mathcal{D}$, and we set $\tau_i := e_{di}^d + e_{id}^d$ for $d := \min G(i)$ and for all $i \in \mathcal{T}$. Inserting τ and π into $\overline{e}$ proves

$$\overline{e}_{di}^d = 0, \qquad \forall\, i \in \mathcal{T}_d \;\; \forall\, d \in \mathcal{D}, \tag{9.2a}$$

$$\overline{e}_{id}^l = 0, \qquad d = \min G(i) \;\; \forall\, i \in \mathcal{T}. \tag{9.2b}$$

Using $x_{li}^l := x_{il}^l := 1$, $l = \min G(i)$ for all $i \in \mathcal{T}$, and $x_a^d := 0$, otherwise, we can prove

$$\overline{f} = \overline{e}^{\mathrm{T}}x \overset{(9.2)}{=} 0. \tag{9.3}$$

For each $i \in \mathcal{T}$ we consider $x_{di}^d := x_{id}^d$, for all $d \in G(i) \setminus \{\min G(i)\}$, $x_{lj}^l := x_{jl}^l := 1$, $l = \min G(j)$ for all $j \in \mathcal{T} \setminus \{i\}$, and $x_a^d := 0$, otherwise, which proves

$$\overline{e}_{id}^d \overset{(9.2)}{=} \overline{e}^{\mathrm{T}}x \overset{(9.3)}{=} 0. \tag{9.4}$$

For each $d \in \mathcal{D}$ and each arc $(i,j) \in A_d^{\text{d-trip}}$, consider $x_{di}^d := x_{ij}^d := x_{jd}^d := 1$ and $x_{lt}^l := x_{tl}^l := 1$, $l = \min G(t)$ for all $t \in \mathcal{T} \setminus \{i,j\}$, which proves

$$\overline{e}_{ij}^d \underset{(9.4)}{\overset{(9.2)}{=}} \overline{e}^{\mathrm{T}}x \overset{(9.3)}{=} 0. \tag{9.5}$$

❑

Note, the equality set of P_{MDVSP} is determined alone by the existence of all pull-out and pull-in trips. Knowing $eq(P_{\text{MDVSP}})$, the dimension $dim(P_{\text{MDVSP}})$ is:

(9.6) Lemma. The dimension of the uncapacitated MDVSP polytope is

$$|A^{\text{u-trip}}| \quad - \quad \sum_{d \in \mathcal{D}} |\mathcal{T}_d| \quad - \quad |\mathcal{T}|.$$

Proof: Lemmata 2.16 and 9.1. ❑

Our polyhedral investigations benefit from the program package PORTA written by Thomas Christof. PORTA is an abbreviation for POlyhedron Representation Transformation Algorithm. It provides tools with which a polyhedron given by a convex set of vertices and (possibly) a cone of extremal rays can be transformed into a representation given by a set of linear equations and inequalities, and vice versa. PORTA is available via WWW, see Christof [1994].

We have used PORTA to enumerate all facets of small problems for which we assume that $G(t) = \mathcal{D}$, for all $t \in \mathcal{T}$, and each subset of timetabled trips can be serviced in sequence by one vehicle. Table 9.1 displays the number of facets for problems with 3 timetabled trips and problems with 4 timetabled trips up to 4 depots. For 3 trips, there are exactly 7 trivial facets $x_a^d \geqslant 0$ per depot, see Lemma 9.7, and $2^{|\mathcal{D}|} - 2$ nontrivial 2-cut or, for $|\mathcal{T}| = 3$ equivalently, 1-path inequalities, see Lemma 9.9 and Lemma 9.14. The number of facets of larger problems could not be determined, but one can easily guess their combinatorial explosion.

$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{D}\rvert$			
	2	3	4	
3	16	27	42	$7\lvert\mathcal{D}\rvert + 2^{\lvert\mathcal{D}\rvert} - 2$
4	50	264	1658	?
5	?	?	?	?

Table 9.1: Number of facets for *complete* instances.

Although we have determined different kinds of valid inequalities (and facets for the above kind of complete problems), almost nothing is actually known about the facial structure of P_{MDVSP}.

9.2 Trivial inequalities

For arbitrary uncapacitated problems, the next lemma gives sufficient conditions when trivial inequalities are facet defining.

(9.7) Lemma. (i) $x_a^d \geqslant 0$ is facet defining for each $a \in A_d^{\text{d-trip}}$ and each $d \in \mathcal{D}$.
(ii) $x_{td}^d \geqslant 0$ is facet defining for $t \in \mathcal{T}_d$ and $d \in \mathcal{D}$ if $|\delta^+(t) \cap A_d^{\text{d-trip}}| \geqslant 3$.
(iii) $x_{dt}^d \geqslant 0$ is facet defining for $t \in \mathcal{T}_d$ and $d \in \mathcal{D}$ if $|\delta^-(t) \cap A_d^{\text{d-trip}}| \geqslant 3$.

Proof: Obviously, these inequalities are valid. We show (i): Let F be the face induced by $x_a^d \geqslant 0$. F defines the same set of feasible solutions as the problem without the deadhead trip $a \in A_d^{\text{d-trip}}$. Hence, the statement follows straightforwardly by Theorem 9.6 and Theorem 1.1. Showing (ii) requires a quite long technical proof whose length would not be justified by its contents. The idea is to prove condition iii of Theorem 1.1. Last but not least, (iii) follows immediately from (ii) by symmetrical reasons. ❑

(9.8) Remark. If $|\delta^+(t) \cap A_d^{\text{d-trip}}| < 3$, the inequality $x_{td}^d \geqslant 0$ is valid, but not always facet defining. For instance, $x_{td}^d \geqslant 0$ is facet defining if

- $\delta^+(t) \cap A_d^{\text{d-trip}} = \emptyset$ and $\delta^-(t) \cap A_d^{\text{d-trip}} = \emptyset$ or
- $\delta^+(t) \cap A_d^{\text{d-trip}} = (t, j)$, $\delta(j) \setminus A_d \neq \emptyset$, and $\delta(t) \setminus A_d \neq \emptyset$,

but not if

- $\delta^+(t) \cap A_d^{\text{d-trip}} = \emptyset$ and $\delta^-(t) \cap A_d^{\text{d-trip}} \neq \emptyset$ or
- $\delta^+(t) \cap A_d^{\text{d-trip}} = (t, j)$, $\delta(j) \setminus A_d \neq \emptyset$, and $\delta(t) \setminus A_d = \emptyset$.

Equivalent results hold for $x_{dt}^d \geqslant 0$ if $|\delta^-(t) \cap A_d^{\text{d-trip}}| < 3$,

9.3 2-cut and Extended 2-cut Inequalities

The facial structure for the following inequalities was investigated for the case where $G(t) = \mathcal{D}$, for all $t \in \mathcal{T}$, and each subset of timetabled trips can be serviced in sequence by one vehicle. In general, they define only valid inequalities.

(9.9) Lemma. 2-cut inequalities. Given some number $p \in \{3, \ldots, |\mathcal{T}|\}$. Let $U := [u_1, \ldots, u_p] \subseteq \mathcal{T}$ denote an ordered set of timetabled trips such that their starting times satisfy $s_{u_1} < \ldots < s_{u_p}$. Additionally, let $\mathcal{D}$ be partitioned into $p-1$ sets $I_1, \ldots, I_{p-1} \in 2^{\mathcal{D}}$ such that their disjoint union equals $\mathcal{D}$ and at least I_1 and I_{p-1} are nonempty. Then the inequality

$$a^{\mathrm{T}}x \quad := \quad x\left(\delta^-(U)\right) \; + \; \sum_{i=1}^{p-1} \sum_{d \in I_i} x^d\left(\{u_i\} \to \{u_{i+1}\}\right) \quad \geqslant \quad 2,$$

called *2-cut inequality*, is valid for P_{MDVSP}.

Proof: Since all coefficients of the vector a are nonnegative, each integer feasible solution covering U with at least two vehicle schedules satisfies $a^{\mathrm{T}}x \geqslant x\left(\delta^{-}(U)\right) \geqslant 2$. Otherwise, if $x\left(\delta^{-}(U)\right) = 1$, the trips of U are serviced by exactly one vehicle, say of depot $d \in \mathcal{D}$. Then for i satisfying $d \in I_i$ holds $x^d_{u_i u_{i+1}} = 1$, i. e., $a^{\mathrm{T}}x = 2$. ❑

(9.10) Lemma. Extended 2-cut inequalities. Given some number $q \in \{3, \ldots, |\mathcal{T}|\}$. Let $\bar{U} := [\bar{u}_1, \ldots, \bar{u}_q] \subseteq \mathcal{T}$ denote an ordered set of timetabled trips such that their starting times satisfy $s_{\bar{u}_1} < \ldots < s_{\bar{u}_q}$. For each $d \in \mathcal{D}$, let $U_d \subseteq \bar{U}$ be given with $U := \bigcap_{d \in \mathcal{D}} U_d = [u_1, \ldots, u_p]$ and $p \geqslant 3$. Additionally, let $\mathcal{D}$ be partitioned into $p-1$ sets $I_1, \ldots, I_{p-1} \in 2^{\mathcal{D}}$ such that their disjoint union equals $\mathcal{D}$ and at least I_1 and I_{p-1} are nonempty. Then the inequality

$$\begin{aligned} b^{\mathrm{T}}x \quad := \quad & x\left(\delta^{-}(\bar{U})\right) + \sum_{d \in I_1} x^d\left(U_d \to \{u_2\}\right) + \sum_{d \in I_{p-1}} x^d\left(\{u_{p-1}\} \to U_d\right) + \\ & \sum_{i=2}^{p-2} \left[\sum_{d \in I_i} \left[x^d\left(\{u_i\} \to \{u_{i+1}\}\right) + \sum_{\substack{\bar{u} \in U_d : \\ u_i < \bar{u} < u_{i+1}}} \left\{ \begin{array}{cc} \text{“either”} & x^d\left(\{u_i\} \to \{\bar{u}\}\right) \\ \text{“or”} & x^d\left(\{\bar{u}\} \to \{u_i\}\right) \end{array} \right\} \right] \right] + \\ & \sum_{d \in \mathcal{D}} \left[x^d\left(\delta^{+}(U_d) \cap \delta^{-}(\bar{U} \setminus U_d)\right) - x^d\left(\delta^{+}(\bar{U}) \cap \delta^{+}(\bar{U} \setminus U_d)\right) \right] \quad \geqslant \quad 2, \end{aligned}$$

called *extended 2-cut inequality*, is valid for P_{MDVSP}.

Proof: The idea of this proof is as follows: For each arbitrary, but fixed integer feasible solution x, we iteratively eliminate nodes $u \in \bar{U} \setminus U$ and shrink paths including such nodes, respectively, until we receive an extended 2-cut inequality that is known to be valid for the shrunken x.

Assume that $x^d\left(\delta^{+}(\bar{U}) \cap \delta^{+}(\bar{U} \setminus U_d)\right) = 0$, for all $d \in \mathcal{D}$, i. e., x uses only arcs having a nonnegative coefficient in b^{T}. Obviously, $b^{\mathrm{T}}x \geqslant x\left(\delta^{-}(\bar{U})\right) \geqslant x\left(\delta^{-}(\{\bar{u}_1\})\right) = 1$. If x covers $\bar{U}$ with two or more vehicle schedules it is easy to see that $b^{\mathrm{T}}x \geqslant x\left(\delta^{-}(\bar{U})\right) \geqslant 2$. Otherwise, x uses exactly one vehicle schedule, say of depot $d \in \mathcal{D}$, to cover $\bar{U}$. Then there exists an unique i satisfying $d \in I_i$, and either $x^d\left(U_d \to \{u_2\}\right) = 1$ or $x^d\left(\{u_{p-1}\} \to U_d\right) = 1$ or

$$x^d\left(\{u_i\} \to \{u_{i+1}\}\right) + \sum_{\substack{\bar{u} \in U_d : \\ u_i < \bar{u} < u_{i+1}}} \left\{ \begin{array}{cc} \text{“either”} & x^d\left(\{u_i\} \to \{\bar{u}\}\right) \\ \text{“or”} & x^d\left(\{\bar{u}\} \to \{u_i\}\right) \end{array} \right\} = 1,$$

and, thus, $b^{\mathrm{T}}x \geqslant 1 + 1 = 2$.

Conversely, assume that there exists some $l \in \mathcal{D}$ and some $u \in \bar{U} \setminus U_l$ such that $x^l\left(\delta^{+}(\bar{U}) \cap \delta^{+}(\{u\})\right) = 1$. Using the flow conservation of u, it follows that

$$1 \quad = \quad x^l\left(\delta^{-}(\{u\})\right) \quad = \quad x^l\left(\delta^{-}(\bar{U}) \cap \delta^{-}(\{u\})\right) + x^l\left(U_l \to \{u\}\right) + x^l\left((\bar{U} \setminus U_l) \to \{u\}\right)$$

Let $\tilde{b}^{\mathrm{T}}x$ denote the extended 2-cut inequality derived from $b^{\mathrm{T}}x$ without u. For the considered (fixed) x, it is easy to see that $(b^d)^{\mathrm{T}}x^d = \left(\tilde{b}^d\right)^{\mathrm{T}}x^d$, for all $d \neq l$. So, we have to show that the same holds for l: If $x^l\left(\delta^-(\bar{U}) \cap \delta^-(\{u\})\right) + x^l(U_l \to \{u\}) = 1$, it is easy to verify that ${b^l}^{\mathrm{T}} x^l =$

$$(\tilde{b}^l)^{\mathrm{T}} x^l + x^l\left(\delta^-(\bar{U}) \cap \delta^-(\{u\})\right) + x^l(U_l \to \{u\}) - x^l\left(\delta^+(\bar{U}) \cap \delta^+(\{u\})\right) = (\tilde{b}^l)^{\mathrm{T}} x^l.$$

If $x^l\left((\bar{U} \setminus U_l) \to \{u\}\right) = 1$, there exists exactly one $v \in \bar{U} \setminus U_l$ such that $x^l_{vu} = 1$. Since $b^l_{vu} x^l_{vu} = 0$, but $\tilde{b}^l_{vu} x^l_{vu} = -1$, it follows that ${b^l}^{\mathrm{T}} x^l = (\tilde{b}^l)^{\mathrm{T}} x^l$. So, we have shown that $b^{\mathrm{T}} x = \tilde{b}^{\mathrm{T}} x$ and, thus, by induction over the nodes in $\bar{U} \setminus U$ follows that at some step $x^d\left(\delta^+(\bar{U}) \cap \delta^+(\bar{U} \setminus U_d)\right) = 0$, i. e., the extended 2-cut inequality is valid. ❑

(9.12) Remark. In the case of $\bar{U} = U$, $b^{\mathrm{T}} x$ reduces to a 2-cut inequality.

9.4 1-path Inequalities

The 1-path inequalities, which we present in the next lemma, have been investigated by Alexander Martin.

(9.13) Definition. Given some (with respect to the starting times) ordered set $V := [u_1, \ldots, u_p] \subseteq \mathcal{T}$ and some partition of V in sets U and $\bar{U}$. Then we say U **dominates** $\bar{U}$ if $|U \cap \{1, \ldots, i\}| \geqslant |\bar{U} \cap \{1, \ldots, i\}|$, for all $i = 1, \ldots, p-1$; we say $\bar{U}$ **subdominates** U if $|\bar{U} \cap \{i, \ldots, p\}| \geqslant |U \cap \{i, \ldots, p\}|$, for all $i = 2, \ldots, p$. If we replace "$\geqslant$" by "$>$", we say that U **strictly** dominates $\bar{U}$, and $\bar{U}$ **strictly** subdominates U.

(9.14) Lemma. 1-path inequalities. Given some $U \subseteq \mathcal{T}$ such that $p := |U| \geqslant 3$ and odd. For each $d \in \mathcal{D}$, let U_d and $\bar{U}_d$ be some partition of U such that

i) U_d dominates $\bar{U}_d$,

ii) $\bar{U}_d$ subdominates U_d,

iii) $\bigcup_{d \in \mathcal{D}} U_d$ strictly dominates $U \setminus \bigcup_{d \in \mathcal{D}} U_d$, and

iv) $\bigcup_{d \in \mathcal{D}} \bar{U}_d$ strictly subdominates $U \setminus \bigcup_{d \in \mathcal{D}} \bar{U}_d$.

Then the inequality

$$d^{\mathrm{T}} x \quad := \quad x\left(\delta^-(U)\right) + \sum_{d \in \mathcal{D}} x^d\left(A_d^{\text{d-trip}}(U) \setminus (U_d \to \bar{U}_d)\right) \quad \geqslant \quad \left\lceil \frac{p}{2} \right\rceil,$$

called *1-path inequality*, is valid for P_{MDVSP}.

Proof: Consider x in terms of cycle flows $x^d = \sum_{w \in W_d} \mu^d_w w$, $\mu^d \in \{0,1\}^{W_d}$. For each $d \in \mathcal{D}$ and each $w \in W_d$, let $U_w \subseteq U$ denote the trips covered by w. It is easy to check that

$d^{\mathrm{T}}w \geqslant \lceil \frac{U_w}{2} \rceil$ and, thus, $d^{\mathrm{T}}x = \sum_{d\in\mathcal{D}} \sum_{w\in W_d} \mu_w^d \cdot d^{\mathrm{T}}w \geqslant \sum_{d\in\mathcal{D}} \sum_{w\in W_d} \mu_w^d \cdot \lceil \frac{U_w}{2} \rceil \geqslant \lceil \frac{p}{2} \rceil$. ❑

Note that conditions i) – iv) are not necessary to show validity of the inequality. If, however, some of these conditions is unsatisfied, the resulting inequality can, at least in the complete case, be strengthened. The name 1-path originates from the fact that each vehicle schedule $w \in W_d$ entering U exactly once and having an empty intersection with $A_d^{\text{d-trip}}(U) \setminus (U_d \to \bar{U}_d)$ uses at most one arc of $(U_d \to \bar{U}_d) \subset A_d^{\text{u-trip}}(U)$.

(9.15) Remark. For $p = 3$, 2-cut and 1-path inequalities are the same.

9.5 Extended Cover Inequalities for the Capacitated MDVSP

We present for the capacitated MDVSP polytope a generalized version of the minimal cover inequality, which has been first presented by Balas [1975], Hammer, Johnson, and Peled [1975], and Wolsey [1975]. Given some depot capacity $x^d(\delta^+(d)) \leqslant \kappa_d$. A set $S \subseteq A_d^{\text{pull-out}}$ is called a **cover** if $|S| > \kappa_d$. The cover is **minimal** if $|S| = \kappa_d + 1$. Given some minimal cover $S \subseteq A_d^{\text{pull-out}}$, the inequality

$$x^d(S) \leqslant \kappa_d$$

is called the **minimal cover inequality** corresponding to S. The next lemma shows how the inequality can be strengthened.

(9.16) Lemma. Extended minimal cover inequality. Given some depot capacity κ_l for depot l and some $U \subseteq \mathcal{T}_l$ such that $|U_l| = \kappa_l + 1$. Then the inequality

$$e^{\mathrm{T}}x \quad := \quad x^l\left(\delta^-(U)\right) + \sum_{d\in\mathcal{D}:\, d\neq l} x^d\left(A_d^{\text{d-trip}}(U)\right) \quad \leqslant \quad \kappa_l,$$

called *extended minimal cover inequality*, and its equivalent representation

$$\tilde{e}^{\mathrm{T}}x \quad := \quad \sum_{d\in\mathcal{D}:\, d\neq l} x^d\left(\delta^-(U)\right) + x^l\left(A_l^{\text{d-trip}}(U)\right) \quad \geqslant \quad 1$$

are valid for the capacitated P_{MDVSP}.

Proof: The equivalence of $d^{\mathrm{T}}x \leqslant \kappa_l$ and $\tilde{d}^{\mathrm{T}}x \geqslant 1$ is given by subtracting the sum of the flow conditions $x(\delta^-(u)) = 1$, for all $u \in U$, from $d^{\mathrm{T}}x \leqslant \kappa_l$ to receive $-\tilde{d}^{\mathrm{T}}x \leqslant -1$. Each integer feasible x can use at most κ_l vehicle schedules of depot l to service all timetabled trips of $\mathcal{T}_l$ and, thus, also of U. Therefore, either at least one circulation of l covers two or more nodes of U or there exists some other depot covering at least one node of U, i. e., either

$$\sum_{d \in \mathcal{D}:\, d \neq l} x^d\left(\delta^-(U)\right) \geqslant 1 \qquad \text{or} \qquad x^l\left(A_l^{\text{d-trip}}(U)\right) \geqslant 1.$$

❑

(9.17) Remark. The part $x^l\left(\delta^-(U)\right)$ of $e^{\mathrm{T}}x$ includes the left-hand side of the minimal cover inequality corresponding to the pull-out trips entering U.

Chapter 10

Solving the MDVSP exactly

At first glance, it seems to be impossible to solve large-scale MDVSPs using commercial or publicly available standard software, even on the newest and fastest workstations or supercomputers. Nonetheless, with an intelligent combination of available LP and minimum-cost flow codes together with implementations of many concepts of combinatorial optimization and integer linear programming, it has become possible to solve such problems on fast workstations to optimality. Hitherto, we have introduced step by step all basic ingredients that turned out to be indispensable to solve our test instances: We solve the integer linear programming formulation of MDVSPs by primal and dual heuristics, column generation and elimination, and branch-and-cut. We have already discussed each component except branch-and-cut.

We have also investigated a Dantzig-Wolfe decomposition as described in the next chapter. It turned out that this decomposition approach is an unsuitable method to solve the MDVSP. The major obstacle here is that the continuous master problem relaxations become too hard to be solved efficiently. Especially for problems with more than one thousand timetabled trips, the LU factorization in solving the restricted master problems takes far too much time. We will discuss the computational results of our decomposition implementation in Chap. 12.

In what follows below, we first give a brief description of the branch-and-cut approach, show how an approximation guarantee can be determined easily using lower and upper bound values, and describe the implementation details of our method to solve the MDVSP exactly.

10.1 Branch-and-Cut

The basic idea of *branch-and-cut* is simple. Most of the valid inequalities, in general facets, of the convex hull of feasible solutions are not used by the initial LP relaxation since there are too much to handle them efficiently. If the optimal solution of the LP relaxation provided by the used LP solver is not feasible, a separation problem is solved to find violated inequalities cutting off the infeasible solution and strengthening the current LP relaxation. This is called the *cutting plane approach*. The enlarged LP is reoptimized. Separation and reoptimization alternate until either the LP solution becomes feasible or no further violated inequality can be identified. In the latter case, a branch-and-bound procedure starts. We assume the reader to be familiar with it. Branch-and-cut combines branch-and-bound and cutting plane such that separation is allowed at each leave of the branch-and-bound tree.

A comprehensive description including all important ingredients of branch-and-cut is, for instance, given in the thesis of Thienel [1995] providing a computational study of several optimization problems solved with the branch-and-cut system ABACUS.

Unlike expected by the experiences of many branch-and-cut applications reported in the literature, our branch-and-cut plays only a subordinate role in solving our test problems. The bottleneck is rather to solve the LP relaxations to optimality. This was possible for 19 out of our 20 real-world test instances. Out of these, 12 could be solved optimally using only column generation/elimination and LP-plunging. The best integer feasible solution for each of the other 7 problems was almost optimal, which gives rise to the assumption that branch-and-bound may be sufficient to solve further problems to optimality. Indeed, four of them could then be solved to the integer optimum by branch-and-bound, all with less than 10 branching nodes. It was also possible to solve these four problems with branch-and-cut, but, surprisingly, needing significantly longer run times. The remaining three problems could neither be improved by branch-and-bound nor by branch-and-cut.

10.2 Approximation Guarantee

Given a problem instance with an optimal integer solution value c^*. Assume that we have determined a valid lower bound c_L and an integer upper bound c_U. Since $0 \leqslant c_L \leqslant c^* \leqslant c_U$, the percentage deviation between c_U and c^* can be approximated by

$$0 \leqslant \frac{c_U - c^*}{c^*} \leqslant \frac{c_U - c_L}{c_L}.$$

From a practical point of view, it can take a long time to obtain a lower bound by the optimal LP value or an improved one by branch-and-cut. Therefore, as long as the LP relaxation is not solved to optimality, we use the somewhat weaker, but much "faster" lower bound obtained by Lagrangean relaxations.

10.3 Implementation Details

The basic components of our algorithm are the following:

- Lagrangean relaxations to quickly obtain tight lower bounds for the minimum fleet size and the minimum operational costs thereof as close as possible to the integer optimum value.

- Primal opening heuristics to obtain a first integer feasible solution and a good starting point for the LP relaxation.
- The LP relaxation approach with a column generation scheme including Lagrangean pricing.
- *LP-plunging* to exploit the information compiled in each (R)LP and its optimal solution.
- Branch-and-cut to solve a problem to proven optimality.
- The workhorses: MCF combined with a column generation and the LP solver CPLEX.

Figure 10.1 gives the flow chart of our method to solve MDVSPs. We first determine a lower bound c_L by the Lagrangean relaxations $\mathrm{LR}_{\mathrm{fcs}}$ and $\mathrm{LR}_{\mathrm{fcd}}$ as close to the integer optimum value as possible. We know from Kokott and Löbel [1996] that the real-world test instances considered here seem to be fairly well structured. Already the trivial problem relaxation $L_{\mathrm{fcs}}(0)$, i. e., simply neglecting the flow conservations, provides very good lower bounds. Second, we compute an upper bound c_U using the opening heuristics SCR and/or ND. Third, the LP relaxation is solved to optimality using our column generation and column elimination scheme. Besides standard reduced cost pricing, the column generation procedure is reinforced by the new Lagrangean pricing. Within the iterative column generation and elimination process, we optionally call the LP-plunging heuristic to improve the current integer feasible solution. Whenever the upper bound c_U could be improved, we check whether $\frac{c_U}{c_L}$ is *small enough* from a practical point of view and stop if this is the case.

When the LP relaxation has been solved to optimality, our method has already generated an optimal solution by LP-plunging for many test instances. In this case, we stop. Otherwise, let c_{LP} denote the optimal LP value. We generate as many nonactive columns as possible (respecting a main memory limit) that have reduced costs smaller than $c_U - c_{LP}$. Note that none of the other inactive variables can have a positive value in an integer solution yielding a smaller objective value than c_U. The resulting RLP is then fixed and solved by branch-and-cut. Of course, branch-and-cut is only a heuristic if not all columns with reduced costs smaller than $c_U - c_{LP}$ have been generated since neither an optimal nor a feasible solution is guaranteed by such a fixed RLP.

Our network simplex code MCF as well as the CPLEX Callable Library, see CPLEX [1995], are the workhorses of our code: Solving the MDVSP with our algorithm exactly requires at several steps the efficient solution of minimum-cost flow problems and linear programs. Standard tools in vehicle scheduling are network flow models and algorithms, which have been profoundly investigated and are well understood. MCF allows to solve the single-depot problems and subproblems to optimality in a few seconds. The Lagrangean functions can, depending on the problem size, also be evaluated in a few seconds up to

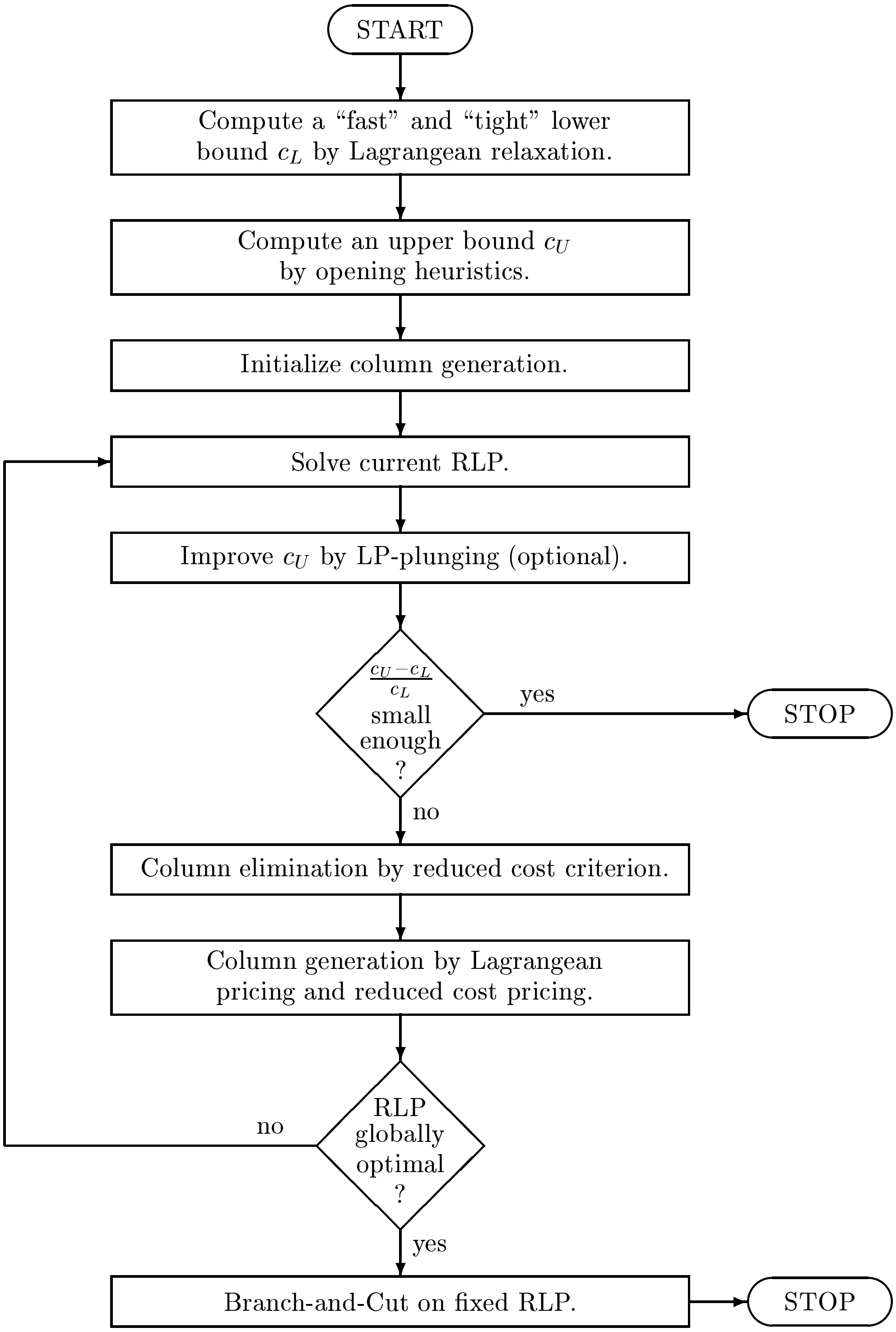


Figure 10.1: Solving MDVSPs: Flow chart.

a few minutes. For instance, L_{fcs} can be exactly evaluated in about 15 minutes for our largest the problem with 70 million of arcs, see Chap 12. The linear programs have been solved with the primal and dual simplex solver of CPLEX, version 4.0.9.

Chapter 11

Dantzig-Wolfe Decomposition

In the previous chapters, we have discussed our branch-and-cut approach with column generation to solve the MDVSP exactly. A further method to solve our problem is Dantzig-Wolfe decomposition, which is due to Dantzig and Wolfe [1960]. This decomposition method has often be used to solve various multicommodity flow problems in the fields of transportation and telecommunication.

The general decomposition principle and its economic interpretation is outlined in Chvátal [1980] and, for network multicommodity flow, in Ahuja, Magnanti, and Orlin [1993]. We will use most of the concepts already proposed by Desrosiers, Dumas, Solomon, and Soumis [1995] for decomposing the MDVSP .

The underlying ILP formulation is based on the contracted digraph D (2.10) reading

$$\min \sum_{d \in \mathcal{D}} \sum_{a \in A_d^{\text{u-trip}}} c_a^d x_a^d \tag{11.1a}$$

subject to

$$
\begin{array}{lrcll}
(11.1\text{b}) & x\big(\delta^+(t)\big) & = & 1, & \forall\, t \in \mathcal{T}, \\
(11.1\text{c}) & x^d\big(\delta^+(d)\big) & \geqslant & \lambda_d, & \forall\, d \in \mathcal{D}, \\
(11.1\text{d}) & x^d\big(\delta^+(d)\big) & \leqslant & \kappa_d, & \forall\, d \in \mathcal{D}, \\
(11.1\text{e}) & x^d\big(\delta^+(t)\big) - x^d\big(\delta^-(t)\big) & = & 0, & \forall\, t \in \mathcal{T}_d \ \ \forall\, d \in \mathcal{D}, \\
(11.1\text{f}) & x & \geqslant & 0. & \\
(11.1\text{l}) & & & x \text{ integral.} &
\end{array}
$$

We will describe in this chapter how this ILP can – at least theoretically – be solved by *branch-and-cut-and-price*: First, (11.1) is reformulated in terms of cycle flows, which give rise to an integer master problem, and we compare this cycle with the original arc formulation. Second, we describe how the linear programming relaxation of this integer master problem can be solved by a delayed column generation giving rise to a pricing problem. Third, we describe the general principle of branch-and-price and how it can be combined with branch-and-cut to a branch-and-cut-and-price approach, which is finally employed to solve the integer master problem to optimality.

11.1 The Master Problem

Hitherto, we have always considered the MDVSP to be given in an arc-oriented formulation with arc flows. It is also possible, however, to consider a cycle flow formulation as follows: For all $d \in \mathcal{D}$, let

$$X_d := \{x^d \in \mathbb{R}^{A_d^{\text{u-trip}}} \mid x^d \geqslant 0 \text{ satisfies (11.1e)}\},$$

$$\mathcal{W}_d := \{S \subset A_d^{\text{u-trip}} \mid S \text{ is a vehicle schedule for depot } d\},$$

and

$$W_d := \{\chi^S \mid S \in \mathcal{W}_d\}.$$

(11.2) Remark. Obviously, each $\mathcal{W}_d$ describes exactly the set of directed cycles in D_d, and W_d describes the incidence vectors according to the dicycles in D_d. We will consider W_d also as the matrix $(\chi^S)_{S \in \mathcal{W}_d}$ that is defined by arranging the elements of W_d columnwise.

The set of all incidence vectors is denoted by $W := \dot{\bigcup}_{d \in \mathcal{D}} W_d$.

(11.3) Lemma. For each $d \in \mathcal{D}$ holds:

$$x^d \in X_d \quad \Longleftrightarrow \quad x^d \in \text{cone}(W_d).$$

Proof: It is obvious that $\text{cone}(W_d) \subseteq X_d$. Conversely, each flow $x^d \in X_d$ can be decomposed with Theorem 1.20, Lemma 1.21, and Remark 11.2 into a dicycle flow, i. e., for each $w \in W_d$ exists a scalar $f_w \geqslant 0$ such that $x^d = \sum_{w \in W_d} f_w w$ or $x^d \in \text{cone}(W_d)$. ❏

(11.4) Remark. Lemma 11.3 follows as an application of the Decomposition Theorem for Polyhedra due to Motzkin, 1936: A set of vectors in Euclidean space is a polyhedron if and only if it can be represented as the sum of some convex set and some cone. In our case, the convex set is $\{0\}$ and the cone is $\times_{d \in \mathcal{D}} \text{cone}(\mathcal{W}_d)$

Inserting each $x^d = W_d \, \mu^d$, $0 \leqslant \mu^d \in \mathbb{R}^{W_d}$, into (11.1) results in an *integer master problem* (**IMP**)

$$\min \sum_{d \in \mathcal{D}} \sum_{w \in W_d} c_w^d \, \mu_w^d \tag{11.5a}$$

subject to

$$\sum_{d\in\mathcal{D}}\sum_{w\in W_d} w\big(\delta^+(t)\big)\,\mu^d_w \;=\; 1, \qquad \forall\, t\in\mathcal{T}, \tag{11.5b}$$

$$\sum_{w\in W_d}\mu^d_w \;\geqslant\; \lambda_d, \qquad \forall\, d\in\mathcal{D}, \tag{11.5c}$$

$$\sum_{w\in W_d}\mu^d_w \;\leqslant\; \kappa_d, \qquad \forall\, d\in\mathcal{D}, \tag{11.5d}$$

$$\mu \;\geqslant\; 0, \tag{11.5e}$$

$$\mu \;\in\; \{0,1\}^W, \tag{11.5f}$$

where each $c^d_w := \sum_{a\in A^{\text{u-trip}}_d} c^d_a\, w_a$ denotes the costs of the vehicle schedule associated with the dicycle $w\in W_d$ within the digraph D_d. In the following, we will call the linear programming relaxation of (11.5) the *master problem* (**MP**).

Per definition of each X_d, the constraints (11.1e) and (11.1f) are always satisfied by $W_d\,\mu^d$ and can therefore be neglected. The transformation of the objective function (11.1a) into (11.5a), the flow condition (11.1b) into (11.5b), and the depot capacities (11.1c) and (11.1d) into (11.5c) and (11.5d) are validated as follows:

- For each $d\in\mathcal{D}$ holds

$$\sum_{a\in A^{\text{u-trip}}_d} c^d_a\, x^d_a \;=\; \sum_{a\in A^{\text{u-trip}}_d} c^d_a\,(W_d\,\mu^d)_a \;=\; \sum_{a\in A^{\text{u-trip}}_d}\sum_{w\in W_d} c^d_a\,(w\,\mu^d_w)_a \;=$$
$$\sum_{w\in W_d}\sum_{a\in A^{\text{u-trip}}_d} c^d_a\, w_a\,\mu^d_w \;=\; \sum_{w\in W_d} c^d_w\,\mu^d_w.$$

- For each $t\in\mathcal{T}$ holds

$$x\big(\delta^+(t)\big) \;=\; \sum_{d\in\mathcal{D}} x^d\big(\delta^+(t)\big) \;=\; \sum_{d\in\mathcal{D}}\sum_{a\in\delta^+(t)\cap A^{\text{u-trip}}_d} x^d_a \;=$$
$$\sum_{d\in\mathcal{D}}\sum_{a\in\delta^+(t)\cap A^{\text{u-trip}}_d} (W_d\,\mu^d)_a \;=\; \sum_{d\in\mathcal{D}}\sum_{a\in\delta^+(t)\cap A^{\text{u-trip}}_d}\sum_{w\in W_d} w_a\,\mu^d_w \;=$$
$$\sum_{d\in\mathcal{D}}\sum_{w\in W_d}\sum_{a\in\delta^+(t)\cap A^{\text{u-trip}}_d} w_a\,\mu^d_w \;=\; \sum_{d\in\mathcal{D}}\sum_{w\in W_d} w\big(\delta^+(t)\big)\,\mu^d_w.$$

- For each $d \in \mathcal{D}$ holds

$$
\begin{aligned}
x^d\big(\delta^+(d)\big) &= \sum_{a\in\delta^+(d)\cap A_d^{\text{u-trip}}} x_a^d = \sum_{a\in\delta^+(d)\cap A_d^{\text{u-trip}}} (W_d\,\mu^d)_a = \\
&\sum_{a\in\delta^+(d)\cap A_d^{\text{u-trip}}} \sum_{w\in W_d} (w\,\mu_w^d)_a = \sum_{w\in W_d} \sum_{a\in\delta^+(d)\cap A_d^{\text{u-trip}}} w_a\,\mu_w^d = \\
&\sum_{w\in W_d} \big(\mu_w^d \sum_{a\in\delta^+(d)\cap A_d^{\text{u-trip}}} w_a\big) = \sum_{w\in W_d} \mu_w^d
\end{aligned}
$$

(11.6) Remark. It is easy to see that each $w\big(\delta^+(t)\big)$ is equal to one if and only if the dicycle $w \in W$ covers the timetabled trip $t \in \mathcal{T}$, otherwise, $w\big(\delta^+(t)\big)$ is zero. Therefore, the system (11.5b) corresponds to a constraint matrix of a set partitioning problem.

11.2 Relation between the Master Problem and the LP Relaxation

It is obvious that the ILP formulation and IMP yield the same optimal integer value. The same holds for the LP relaxation and MP, but MP provides more fractional basic solutions. More precisely, let P_{LP} and P_{MP} denote the polytopes associated with the LP relaxation and MP. It is easy to show that there exist vertices $\bar{\mu} \in P_{\text{MP}}$ such that $(x^d = W_d\,\bar{\mu}^d)_{d\in\mathcal{D}} \in P_{\text{LP}}$ is a nontrivial convex combination of vertices in P_{LP}. For instance, we have enumerated the vertices of P_{LP} and P_{MP} for a uncapacitated problem with $|\mathcal{D}| = 2$, $|\mathcal{T}| = 3$, and $G \equiv \mathcal{D}$, and we have enumerated all facets of the integer version of P_{LP} and P_{MP}, which we denote by P_{ILP} and P_{IMP} These enumerations have performed with PORTA.

For this small instance, the polytopes P_{LP} and P_{MP} contain exactly 22 equivalent integral and four equivalent fractional vertices: The integral vertices of P_{MP} are the following vectors

$$
\begin{pmatrix}1\\0\\0\end{pmatrix}+\begin{pmatrix}0\\1\\0\end{pmatrix}+\begin{pmatrix}0\\0\\1\end{pmatrix},\quad
\begin{pmatrix}1\\1\\0\end{pmatrix}+\begin{pmatrix}0\\0\\1\end{pmatrix},\quad
\begin{pmatrix}1\\0\\1\end{pmatrix}+\begin{pmatrix}0\\1\\0\end{pmatrix},\quad
\begin{pmatrix}0\\1\\1\end{pmatrix}+\begin{pmatrix}1\\0\\0\end{pmatrix},\quad\text{and}\quad
\begin{pmatrix}1\\1\\1\end{pmatrix}
$$

whereas each occurring column vector can belong to any of the two depots. It is easy to verify that there are exactly 22 combinations possible. The fractional vertices of P_{MP} are the eight possible combinations of

$$
\frac{1}{2}\begin{pmatrix}1\\1\\0\end{pmatrix}+\frac{1}{2}\begin{pmatrix}1\\0\\1\end{pmatrix}+\frac{1}{2}\begin{pmatrix}0\\1\\1\end{pmatrix}. \tag{11.7}
$$

However, only four of them, for which the two columns $(1,1,0)^{\mathrm{T}}$ and $(0,1,1)^{\mathrm{T}}$ belong to different depots, can be transformed to vertices of P_{LP}. If, for instance, all three columns belong to the same depot, the corresponding solution in P_{LP} can be represented as a nontrivial convex combination of the vertices in P_{LP} corresponding to $(1,1,1)^{\mathrm{T}}$ and $(1,0,1)^{\mathrm{T}} + (0,1,0)^{\mathrm{T}}$ in P_{MP}. Based on heuristical arguments, Forbes, Holt, and Watts [1994] claim that such cases are unusual: Either the column $(1,1,1)^{\mathrm{T}}$ or the columns $(1,0,1)^{\mathrm{T}} + (0,1,0)^{\mathrm{T}}$ would be most probably cheaper than (11.7) such that it would be unlikely that a solution like (11.7) is optimal. We agree with them, however, only if the columns $(1,1,0)^{\mathrm{T}}$ and $(0,1,1)^{\mathrm{T}}$ of (11.7) belong to the same depot and if the optimal solution in P_{LP} is unique.

Based on our data we cannot guarantee that the optimal solution of P_{LP} is unique. Therefore, solving the MP requires either a proper perturbation of the objective function or we cannot avoid that we obtain an optimal solution such as (11.7), which may require additional branching steps in the branch-and-cut-and-price algorithm that we will describe below.

11.3 The Pricing Problem

Although the master problem has significantly fewer equations than the LP relaxation, it contains exponentially many variables. Nevertheless, we can solve MP with a delayed column generation approach. The general column generation principle was outlined for the LP relaxation in Chap. 7.

Let the *restricted master problem* (**RMP**) be the linear program that is defined by the columns of MP corresponding to some $\tilde{W} \subseteq W$. We assume that RMP is primal feasible. Let $\nu \in \mathbb{R}^{\mathcal{T}}$, $0 \leqslant \beta \in \mathbb{R}^{\mathcal{D}}$, and $0 \leqslant \gamma \in \mathbb{R}^{\mathcal{D}}$ denote the dual multipliers according to (11.5b), (11.5c), and (11.5d) for the optimal basis of a current RMP. The reduced costs of the variables of MP are given by

$$\overline{c}_w^d := c_w^d - \nu^{\mathrm{T}} \Big(w\big(\delta^+(t)\big)\Big)_{t\in\mathcal{T}} - \beta_d + \gamma_d \qquad \forall\, w \in W_d \;\; \forall\, d \in \mathcal{D}.$$

For convenience of notation, we define for each $d \in \mathcal{D}$ an artificial variable $\nu_d \in \mathbb{R}$ and set it to zero. It follows straightforwardly that

$$\begin{aligned} c_w^d \;-\; \nu^{\mathrm{T}} \Big(w\big(\delta^+(t)\big)\Big)_{t\in\mathcal{T}} \quad &= \quad c_w^d \;-\; \sum_{t\in\mathcal{T}} \nu_t w\big(\delta^+(t)\big) \;-\; \nu_d\, w\big(\delta^+(d)\big) \quad = \\ &\sum_{ij\in A_d^{\text{u-trip}}} c_{ij}^d\, w_{ij} \;-\; \sum_{ij\in A_d^{\text{u-trip}}} \nu_{ij}\, w_{ij} \quad = \quad \sum_{ij\in A_d^{\text{u-trip}}} \big(c_{ij}^d - \nu_i\big)\, w_{ij} \end{aligned}$$

A given basis is optimal if and only if for all $w \in W_d$ and for all $d \in \mathcal{D}$ the reduced costs $\overline{c}_w^d$ are nonnegative. Therefore, a basis is optimal for RMP as well as for MP if the *pricing* *problem* (**PP**)

$$\min_{d\in\mathcal{D}}\left[\gamma_d - \beta_d + \min_{w\in W_d} \sum_{ij\in A_d^{\text{u-trip}}} (c_{ij}^d - \nu_i)\, w_{ij}\right] \tag{11.8}$$

yields a nonnegative value.

11.4 Solving the Master Problem

From a complexity theory's point of view, it is a $\mathcal{NP}$-hard problem assuming the initial RMP to be primal feasible; however, we refer to Rem. 3.11, in which we determined depot capacities to be often soft constraints. We initialize the first RMP with the columns associated with a feasible set of vehicle schedules given by some primal heuristics as, for instance, a nearest depot heuristic, see Chap. 8.

We attack the pricing problem as follows: Let PP_d denote the pricing problem for a fixed $d \in \mathcal{D}$. We split the depot node d into its original nodes d^+ and d^- such that d^+ becomes the new tail node of all pull-out trips, and d^- becomes the new head node of all pull-in trips. Then, PP_d is the problem to find the shortest path from d^+ to d^- according to the arc weights $\tilde{c}^d := (c_{ij}^d - \nu_i)_{ij\in A_d}$. Fortunately, the underlying network for each PP_d with the two new depot nodes is acyclic. The shortest paths – even if negative arc weights occur – can therefore be computed in $\mathcal{O}(|A_d|)$ time using, e. g., the *reaching* algorithm as proposed in Ahuja, Magnanti, and Orlin [1993].

The algorithm proceeds in the following way: For $d \in \mathcal{D}$, let $\tilde{w}^d$ denote the shortest path for PP_d. If $\gamma_d - \beta_d + \tilde{c}^d\,\tilde{w}^d \geqslant 0$ for all $d \in \mathcal{D}$, the optimal basis of the current RMP is also optimal for MP, and we are done. Otherwise, the column of at least one $\tilde{w}^d$ violating the optimality condition is generated and added to RMP, the enlarged RMP is reoptimized, and we iterate. Between two consecutive RMPs, we generate for each depot d the column corresponding to the shortest path $\tilde{w}^d$ if $\gamma_d - \beta_d + \tilde{c}^d\,\tilde{w}^d < 0$.

Our computational investigations have shown that Lagrangean pricing is indispensable to solve the LP relaxation of larger MDVSP instances. Thus, we came up with the idea that our decomposition may also benefit from Lagrangean pricing. And indeed, it can help to significantly accelerate the solution of MP: Consider the subproblems L_{fcd} of the Lagrangean relaxation LR_{fcd} for which the flow conditions have been put into the objective function. We use the value of the optimal dual multipliers ν of the last RMP as estimators of the Lagrange multipliers of the flow conditions and evaluate L_{fcd} at ν. In Chap. 6, we have shown that each optimal solution of $L_{\text{fcd}}(\nu)$ corresponds to a set of vehicle schedules that seem to be advantageous for the given Lagrange multipliers ν. Thus, we generate the columns of all these suggested vehicle schedules for which the reduced cost criterion for MP is not satisfied and which are currently not active. In addition to the standard column generation scheme, this Lagrangean pricing is always used between two consecutive RMPs to generate further columns.

11.5 Solving the Integer Master Problem

11.5.1 Branch-and-Price

Applying a branch-and-bound procedure to a fixed RMP that contains an optimal basis according to MP, the current set of active columns will neither guarantee an optimal nor a feasible integral solution (see, e. g., Barnhart, Johnson, Nemhauser, Savelsbergh, and Vance [1994] or Sol [1994]). Barnhart et al. claim that, nevertheless, many decomposition problem instances have been solved successfully, but not to proven optimality, by the heuristic of limiting the column generation to the root node of the branch-and-bound tree.

Incorporating column generation in each node of a branch-and-bound tree – this is called *branch-and-price* – is hard: Standard branching on variables of MP may be ineffective since fixing a variable μ_w^d to zero destroys the easy structure of PP. Moreover, there no guarantee and it is even most likely that the pricing problem will regenerate the column of μ_w^d within the branching node corresponding to $\mu_w^d := 0$. To avoid a reactivation of μ_w^d, we have to compute also the second shortest path, and, for a branching depth k, we would have to compute in the worst case the k^{th} shortest path. For this reason, we should use branching strategies that are *compatible* to PP. The following actions can be easily performed for PP:

Fixing arc variables to zero. To fix some arc $a \in A_d^{\text{u-trip}}$ to zero, we set its weight $\tilde{c}_a^d$ in PP_d to infinity. Consequently, all columns μ_w^d satisfying $w_{ij} = 1$, $w \in W_d$, can also be fixed to zero.

Removing depots from a depot group. To forbid that a trip $t \in \mathcal{T}$ is serviced by a depot $d \in G(t)$, we can easily remove d from $G(t)$ and fix all arcs of $A_d^{\text{u-trip}}(\{t\})$ to zero.

Contracting two timetabled trips. If for some depot d two subsequent timetabled trips i and $j \in \mathcal{T}_d$ are serviced in sequence, we have to contract the nodes i and j. Note that each arc weight $\tilde{c}_{ij}^d$ must be shifted on the node j (or i) to each arc of $\delta^+(j) \cap A_d^{\text{u-trip}}$ (or $\delta^-(i) \cap A_d^{\text{u-trip}}$).

It would be advantageous to have branching rules that can make use of some of these actions: We compute the vector $x := \left(W_d\, \mu^d\right)_{d \in \mathcal{D}}$. Obviously, x is integral if and only if μ is integral. If this is the case, MP has been solved to optimality, and we can stop. Otherwise, we perform one of the following briefly sketched branching strategies and iterate the branch-and-price algorithm until optimality is proved.

Standard branching on the original variables in (11.1)**.** We select some $d \in \mathcal{D}$ and some arc $(i, j) \in A_d^{\text{u-trip}}$ such that x_{ij}^d is fractional. On the left branch, we set $x_{ij}^d := 0$ and $\mu_w^d := 0$, for all $w \in W_d$ with $w_{ij} = 1$. On the right branch, we set $x_{ij}^d := 1$ and

$\sum_{w \in W_d:\, w_{ij}=1} \mu_w^d := 1$; in addition, all the other depots than d can be removed from the depot groups $G(i)$ and $G(j)$.

Branching on depot groups. We select some $t \in \mathcal{T}$ and divide $G(t)$ in two subsets $G_l(t)$ and $G_r(t)$ such that $\sum_{d \in G_l(t)} x^d\big(\delta^+(t)\big)$ and $\sum_{d \in G_r(t)} x^d\big(\delta^+(t)\big)$ are both fractional. On the left branch, we remove $G_l(t)$ from $G(t)$, and on the left branch, we remove $G_r(t)$.

The branching strategy proposed by Ryan and Foster [1981]. Let i and j be two timetabled trips such that $x(i \to j)$ is fractional. On the left branch, we set $x(i \to j) := 0$, i. e., we fix each arc $a \in (i \to j)$ to zero. On the right branch, we contract i and j and fix each arc $a \in \big(\delta^+(i) \cup \delta^-(j)\big) \setminus (i \to j)$ to zero.

11.5.2 Branch-and-Cut-and-Price

Branch-and-cut and branch-and-price can be combined to a *branch-and-cut-and-price* approach. Such an approach tries to tighten MP by column generation and cutting plane generation. The difficulty of branch-and-cut-and-price is the incompatibility of its two parts: PP can become much harder and even impossible to be solved if we add valid cuts (or facets) to MP since a new cut can destroy the well suited structure of PP. Vice versa, generating new columns may also have a negative effect on separation, see Barnhart et al.

For instance, the only facet of MP for our small example from Sect. 11.2 is the following clique inequality (see Padberg [1973]):

$$\sum_{d \in \mathcal{D}} \sum_{w \in \left\{\left(\begin{smallmatrix}1\\1\\0\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\0\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}1\\1\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}0\\1\\1\end{smallmatrix}\right)\right\}} \mu_w^d \leqslant 1 \tag{11.9}$$

It is easy to check that there exists no vector $e^d \in \mathbb{R}^{A_d^{\text{u-trip}}}$ such that ${e^d}^{\mathrm{T}} W_d\, \mu^d$ is equal to the left-hand side of (11.9) for any fixed d. This means that it would be impossible to consider (11.9) within PP.

Each inequality

$$\sum_{d \in \mathcal{D}} {e^d}^{\mathrm{T}} x^d \geqslant f \tag{11.10}$$

being valid for the integer version of (11.1) (and for the ILP (2.14), resp.) defines a valid inequality

$$\sum_{d \in \mathcal{D}} {e^d}^{\mathrm{T}} W_d\, \mu^d \geqslant f \tag{11.11}$$

for MP. In our case, separation can be attacked similarly to the branching for branch-and-bound: We have to separate $x := \big(W_d\, \mu^d\big)_{d \in \mathcal{D}}$ for the arc formulation and must transform identified violated cuts to a representation valid for MP. This strategy is capable to separate each fractional solution of MP whose equivalent fractional representation within the LP relaxation could be separated anyhow. Only those fractional solutions that correspond to some nontrivial convex combinations of integral solutions within the LP relaxation can not be attacked at all by such a separation approach. If, however, such a degenerated case occurs, branching becomes indispensable.

This separation scheme is compatible to PP: Let us enlarge the LP relaxation (11.1) by some valid cut (11.10) and the master problem (11.5) by the corresponding cut (11.11). Let $0 \leqslant \zeta \in \mathbb{R}$ denote the dual multiplier according to (11.11). The pricing problem (11.8) becomes

$$\min_{d \in \mathcal{D}} \left[\gamma^d - \beta^d + \min_{w \in W_d} \sum_{ij \in A_d^{\text{u-trip}}} (c_{ij}^d - \nu_i - e_{ij}^d\, \zeta)\, w_{ij} \right], \tag{11.12}$$

i. e., the arc weights $\tilde{c}^d$ simply become $\tilde{c}^d - \zeta e^d$, and each cut of type (11.11) can be easily incorporated in our pricing procedure.

To sum up, one can say that branching and separating on the original variables is a good compromise for branch-and-cut-and-price approaches if the problem can completely be described with an arc formulation and with a path and cycle formulation.

Chapter 12

Computational Results

This is the largest chapter of this thesis summarizing all our computational results for the MDVSP. With the results presented here, we want to prove the effectiveness of our developed and implemented methods to solve large MDVSP instances from practice.

We start the presentation of our computational results in Sect. 12.1 with a description of the individual depot data of the city of Berlin (BVG), the city of Hamburg (HHA), and the region around Hamburg (VHH) in Tabs. 12.1–12.3. This data are the basis of our single-depot and multiple-depot test instances, which are in detail presented in Sect. 12.2.

The solution statistics of the single-depot instances obtained by the individual depot data and single-depot relaxations are illustrated in Tabs. 12.5–12.8 in Sect. 12.3. The central results of the investigated multiple-depot instances such as the objective values (of the lower bounds, the optimal integer value, and the upper bounds) and the run times are given in Tabs. 12.9–12.12 in Sect. 12.4. Detailed results of the subgradient methods for the Lagrangean relaxations are given in Sect. 12.5. Some specific results of solving the LP relaxation without LP-plunging is presented in Sect. 12.6. We close this chapter with the results obtained for the Dantzig-Wolfe decomposition method in Sect. 12.7.

Admittedly, all our computational tests have been performed without a consideration of depot capacities for two reasons: First, HanseCom has not provided depot capacities for HHA and VHH. Second, each BVG test instance is just based on a subset of all its timetabled trips and/or depots making it pointless to consider the given capacities. It also speaks well for ignoring depot capacities that they can be considered as soft constraints, see Rem. 3.11, and none of our partners has ever complained about the necessary fleet sizes of our solutions.

All computational tests have been performed on a SUN Model 170 UltraSPARC with 512 MByte main memory and 1.7 MByte virtual memory. We were the only user on this machine during our test runs. All linear programs have been solved with the CPLEX Callable Library, version 4.0.7 and 4.0.9, all minimum-cost flow problems and single-depot subproblems have been solved with the callable library of our network simplex code MCF combined with a column generation.

12.1 Real-World Data Specifications

Currently, BVG maintains 9 garages and runs 10 different vehicle types (2 double-decker types, 6 single-decker types, and 3 articulated bus types). Combining the garages with their available vehicle types results in 44 depots, see Tab. 12.1. For a normal weekday, about 28,000 timetabled trips have to be serviced. Since BVG outsources some trips to third-party companies, this number reduces to 24,906. Using all degrees of freedom, these 25 thousand timetabled trips can be linked with about 70 million unloaded trips.

Depots	$\|\mathcal{T}\|$	$\|A^{\text{u-trip}}\|$		Depots	$\|\mathcal{T}\|$	$\|A^{\text{u-trip}}\|$	
		User-def.	Total			User-def.	Total
B-Depot 1	2139	15156	1,309,598	B-Depot 23	2403	16201	1,917,649
B-Depot 2	2468	22631	1,610,588	B-Depot 24	2370	15666	1,706,145
B-Depot 3	1872	12813	849,007	B-Depot 25	2143	14850	1,516,896
B-Depot 4	2223	15898	1,444,077	B-Depot 26	4225	28243	3,036,693
B-Depot 5	2386	17117	1,498,079	B-Depot 27	3349	20741	3,027,439
B-Depot 6	2623	19800	1,775,719	B-Depot 28	2764	17518	2,205,739
B-Depot 7	1771	13264	821,797	B-Depot 29	2920	18112	2,343,428
B-Depot 8	1867	13612	904,995	B-Depot 30	2665	17287	2,015,129
B-Depot 9	1664	12586	346,470	B-Depot 31	2981	18582	2,415,834
B-Depot 10	940	8922	117,164	B-Depot 32	2340	16617	1,360,858
B-Depot 11	1513	11421	363,965	B-Depot 33	2812	18995	1,949,709
B-Depot 12	867	8631	109,722	B-Depot 34	2179	16270	1,021,680
B-Depot 13	2396	23897	1,756,254	B-Depot 35	2772	19897	1,769,839
B-Depot 14	2218	27201	1,713,144	B-Depot 36	2577	17969	1,487,242
B-Depot 15	1741	20797	1,009,292	B-Depot 37	2202	12811	1,572,817
B-Depot 16	1644	20352	824,190	B-Depot 38	1965	13634	1,496,895
B-Depot 17	3464	27543	2,981,149	B-Depot 39	2049	11948	1,499,119
B-Depot 18	2738	22784	1,733,559	B-Depot 40	1369	9320	813,746
B-Depot 19	2186	16758	1,395,188	B-Depot 41	1565	10110	977,644
B-Depot 20	2751	23428	1,855,240	B-Depot 42	2500	14781	2,094,639
B-Depot 21	2964	21279	2,740,790	B-Depot 43	1720	10938	1,141,457
B-Depot 22	2631	19156	2,148,047	B-Depot 44	1369	9320	813,746

Table 12.1: BVG depots.

HHA together with some other transportation companies maintain 14 garages with 9 different vehicle types resulting in 40 depots, see Tab. 12.2. More than 16,000 daily trips must be scheduled with about 15.1 million unloaded trips. This problem decomposes into seven multiple-depot and nine single-depot instances.

VHH currently plans 10 garages with 9 different vehicle types. The garage-vehicle combinations define 19 depots, see Tab. 12.3. The 5,447 timetabled trips of VHH can be linked with about 10 million unloaded trips.

Depots	$\lvert\mathcal{T}\rvert$	$\lvert A^{\text{u-trip}}\rvert$	
		User-def.	Total
Hamburg 1 - Depot 1	2,900	399,601	2,170,962
Hamburg 1 - Depot 2	2,277	143,146	2,285,501
Hamburg 1 - Depot 3	1,716	116,142	850,828
Hamburg 1 - Depot 4	1,065	31,520	488,655
Hamburg 1 - Depot 5	1,413	48,374	339,891
Hamburg 1 - Depot 6	1,076	45,723	318,626
Hamburg 1 - Depot 7	728	13,654	233,010
Hamburg 1 - Depot 8	2,288	257,733	1,360,544
Hamburg 1 - Depot 9	1,882	102,764	1,579,315
Hamburg 1 - Depot 10	1,588	79,955	404,016
Hamburg 1 - Depot 11	1,296	77,201	459,468
Hamburg 1 - Depot 12	892	25,100	348,230
Hamburg 2 - Depot 1	4	9	12
Hamburg 2 - Depot 2	214	2,059	18,700
Hamburg 2 - Depot 3	21	195	225
Hamburg 2 - Depot 4	211	2,651	16,854
Hamburg 2 - Depot 5	648	14,136	180,554
Hamburg 2 - Depot 6	55	468	1,430
Hamburg 2 - Depot 7	695	16,526	186,863
Hamburg 2 - Depot 8	493	8,322	104,153
Hamburg 2 - Depot 9	1,365	56,988	475,533
Hamburg 3 - Depot 1	521	19,003	65,650
Hamburg 3 - Depot 2	521	10,745	122,692
Hamburg 4 - Depot 1	230	1,858	22,151
Hamburg 4 - Depot 2	17	54	152
Hamburg 5 - Depot 1	986	56,760	211,153
Hamburg 5 - Depot 2	930	27,452	368,242
Hamburg 6 - Depot 1	1,345	102,253	322,546
Hamburg 6 - Depot 2	1,693	72,882	1,233,766
Hamburg 7 - Depot 1	232	3,922	12,510
Hamburg 7 - Depot 2	220	2,348	21,315
Hamburg 8	20	66	147
Hamburg 9	10	29	54
Hamburg 10	126	442	3,322
Hamburg 11	14	53	68
Hamburg 12	1	2	2
Hamburg 13	71	355	2,388
Hamburg 14	109	617	5,241
Hamburg 15	183	1,025	14,347
Hamburg 16	194	1,943	15,419

Table 12.2: HHA depots.

Depots	$\lvert\mathcal{T}\rvert$	$\lvert A^{\text{u-trip}}\rvert$	
		User-def.	Total
Hamburg-Holstein - Depot 1	10	29	50
Hamburg-Holstein - Depot 2	1508	137,965	158,607
Hamburg-Holstein - Depot 3	2920	241,720	787,772
Hamburg-Holstein - Depot 4	2355	158,053	2,501,906
Hamburg-Holstein - Depot 5	14	36	90
Hamburg-Holstein - Depot 6	700	12,705	207,772
Hamburg-Holstein - Depot 7	2351	145,095	688.574
Hamburg-Holstein - Depot 8	1665	79,404	1,262,356
Hamburg-Holstein - Depot 9	2	4	4
Hamburg-Holstein - Depot 10	758	19,138	54,345
Hamburg-Holstein - Depot 11	602	12,059	160,374
Hamburg-Holstein - Depot 12	373	5,563	57,451
Hamburg-Holstein - Depot 13	1	2	2
Hamburg-Holstein - Depot 14	953	25,334	64,447
Hamburg-Holstein - Depot 15	799	18,641	283,643
Hamburg-Holstein - Depot 16	417	5,095	73,045
Hamburg-Holstein - Depot 17	2451	150,944	2,637,754
Hamburg-Holstein - Depot 18	1213	28,594	188,124
Hamburg-Holstein - Depot 19	795	13,037	272,893

Table 12.3: VHH depots.

12.2 The Test Instances

Multiple-depot problems.

Different parameter settings and optimization aspects yielded in the test instances that are displayed in Tab. 12.4. Besides the total number of arcs in $A^{\text{u-trip}}$, we also give the number of user-defined unloaded trips without pull-in-pull-out trips. The term $\varnothing G := \sum_{t\in\mathcal{T}} G(t)/|\mathcal{T}|$ denotes the average depot group size. Note that the number of equations of (2.14) is equal to the number of flow conditions and flow conservations.

Multiple-depot test sets	$\|\mathcal{D}\|$	$\|\mathcal{T}\|$	$\|A^{\text{u-trip}}\|/1,000$		$\varnothing G$	number of equations
			User-def.	All		
Berlin 1	44	24,906	846	69,700	4.03	125,255
Berlin 2	49	24,906	304	13,200	1.56	63,641
Berlin 3	3	1,313	77	2,300	2.33	4,370
Berlin-Spandau 1	9	2,424	164	3,700	4.94	14,418
Berlin-Spandau 2	9	3,308	327	8,800	5.49	21,470
Berlin-Spandau 3	13	2,424	39	590	1.92	7,103
Berlin-Spandau 4	13	3,308	72	1,530	2.25	10,753
Berlin-Spandau 5	13	3,331	75	1,550	2.25	10,834
Berlin-Spandau 6	13	1,998	28	380	1.90	5,798
Berlin-Spandau 7	7	2,424	145	3,300	4.16	12,506
Berlin-Spandau 8	7	3,308	283	7,800	5.02	18,376
Hamburg 1	12	8,563	1,322	10,900	2.23	27,696
Hamburg 2	9	1,834	99	1,000	2.02	5,549
Hamburg 3	2	791	30	200	1.32	1,835
Hamburg 4	2	238	2	23	1.04	487
Hamburg 5	2	1,461	85	580	1.31	3,379
Hamburg 6	2	2,283	176	1,600	1.33	5,323
Hamburg 7	2	341	6	34	1.32	795
Hamburg-Holstein 1	4	3,413	230	4,000	1.68	9,167
Hamburg-Holstein 2	19	5,447	1,054	9,400	3.65	25,334

Table 12.4: Real-world multiple-depot test instances.

There are no depots containing vehicles of different garages for each of these multiple-depot test instances.

Berlin 1: This is the complete BVG problem with all possible degrees of freedom.

Berlin 2: This problem is based on the timetabled trip set of Berlin 1, but the depots and the dead-head trips are generated with different rules resulting in fewer degrees of freedom.

Berlin 3: This is a small test instance including 9 lines from the south of Berlin and 3 depots from one single garage.

Berlin-Spandau 1 – 8: All the test sets denoted by Berlin-Spandau are defined on the data of the district of Spandau for different weekdays and different depot generation rules.

Hamburg 1 – 7: Here we consider the multiple-depot subproblems of HHA.

Hamburg-Holstein 1: This is a subset of VHH containing not all its depots and trips.

Hamburg-Holstein 2: This test set is based on the complete data of VHH.

Single-depot problems.

The single-depot instances are obtained by the individual depot data of BVG and HHA, assuming that each depot must service all its possible timetabled trips, and by the single-depot relaxations $\mathrm{LR}_{\mathrm{fcs}}(0)$ of each multiple-depot instance.

12.3 Solving the Single-Depot Instances

Single-depot instances can be modelled as minimum-cost flow problems. Hence, real-world SDVSPs of any size can be efficiently solvable as shown in Tables 12.5–12.8. These tables give for the single-depot problems information about the run times and performed simplex iterations for the default version of MCF and MCF with column generation. For the column generation, there are also given the number of generated columns and the number of restricted minimum-cost flow problems that have been solved until optimality has been proved.

With the default version of MCF, all BVG depots together can be solved in less than 80 minutes. The solution time can be accelerated to less than 10 minutes with column generation. For the HHA problems, these two times are 19 and 3 minutes. Reinforcing the default version of MCF with column generation, the run times could be decreased, on the average, by about 89 % and the performed simplex iterations by about 83 %.

Table 12.8 shows the results for truly large-scale instances obtained by Lagrangean relaxation. Even the largest problem with 25 thousand timetabled trips and 70 million unloaded trips, which results in a minimum-cost flow problem with 50 thousand nodes and 70 million arcs, can be solved in about 15 minutes to optimality using MCF with column generation. MCF needs about 25 minutes to solve all single-depot relaxations together to optimality.

These results support the conclusion that each real-world single-depot problem can be scheduled optimally in a few minutes. In particular, clustered multiple-depot problems can be solved efficiently for every public transportation company in the world.

Depots	$\lvert V\rvert$	$\lvert A\rvert$	MCF default		MCF with column generation			
			Simplex iterations	CPU time[a]	CPU time[a]	Simplex iterations	# Gener. columns[b]	# Restr. problems[c]
Berlin-Depot 1	4,279	1,309,598	129,125	68	9	25,773	97,050	4
Berlin-Depot 2	4,937	1,610,588	161,791	117	13	31,390	126,723	5
Berlin-Depot 3	3,745	849,007	65,867	40	4	19,307	48,911	3
Berlin-Depot 4	4,447	1,444,077	193,945	119	6	24,380	54,639	3
Berlin-Depot 5	4,773	1,498,079	175,840	124	6	24,577	56,660	3
Berlin-Depot 6	5,247	1,775,719	261,205	151	10	31,160	89,384	4
Berlin-Depot 7	3,543	821,797	63,988	38	4	19,653	49,044	3
Berlin-Depot 8	3,735	904,995	72,084	49	4	20,381	50,612	3
Berlin-Depot 9	3,329	346,470	16,163	13	3	17,563	63,524	3
Berlin-Depot 10	1,881	117,164	4,613	2	1	10,654	28,618	2
Berlin-Depot 11	3,027	363,965	19,531	13	3	17,454	48,209	3
Berlin-Depot 12	1,735	109,722	4,133	3	1	10,380	26,233	2
Berlin-Depot 13	4,793	1,756,254	215,499	119	12	38,121	98,054	4
Berlin-Depot 14	4,437	1,713,144	196,786	94	11	44,649	97,936	4
Berlin-Depot 15	3,483	1,009,292	104,520	43	8	33,791	89,195	4
Berlin-Depot 16	3,289	824,190	69,789	38	7	32,362	88,328	4
Berlin-Depot 17	6,929	2,981,149	365,002	276	25	47,482	158,118	6
Berlin-Depot 18	5,477	1,733,559	200,484	180	11	34,431	92,166	4
Berlin-Depot 19	4,373	1,395,188	180,030	95	9	28,150	84,142	4
Berlin-Depot 20	5,503	1,855,240	220,541	158	12	37,342	105,372	4
Berlin-Depot 21	5,929	2,740,790	386,941	284	14	34,887	91,498	4
Berlin-Depot 22	5,263	2,148,047	253,242	148	14	32,593	88,240	5

[a] CPU run times in seconds without reading the problems.
[b] Number of active columns of the last restricted arc set; note, there was no column elimination.
[c] Number of restricted problems that have been solved until optimality has been proved.

Table 12.5: Minimum-cost flow problems from BVG Depots 1–22.

Depots	$\|V\|$	$\|A\|$	MCF default		MCF with column generation			
			Simplex iterations	CPU time[a]	CPU time[a]	Simplex iterations	# Gener. columns[b]	# Restr. problems[c]
Berlin-Depot 23	4,807	1,917,649	195,814	134	8	25,165	56,313	4
Berlin-Depot 24	4,741	1,706,145	162,903	128	10	25,660	83,333	5
Berlin-Depot 25	4,287	1,516,896	125,018	101	9	24,888	81,623	4
Berlin-Depot 26	8,451	3,036,693	411,807	291	39	44,244	137,002	6
Berlin-Depot 27	6,699	3,027,439	440,860	317	28	38,762	172,072	7
Berlin-Depot 28	5,529	2,205,739	286,844	164	13	30,450	86,708	4
Berlin-Depot 29	5,841	2,343,428	303,000	223	23	33,507	161,984	6
Berlin-Depot 30	5,331	2,015,129	234,430	140	12	29,578	86,147	4
Berlin-Depot 31	5,963	2,415,834	310,058	172	17	33,589	106,700	5
Berlin-Depot 32	4,681	1,360,858	137,648	67	10	27,581	128,344	5
Berlin-Depot 33	5,625	1,949,709	253,413	126	17	31,944	147,490	6
Berlin-Depot 34	4,359	1,021,680	102,569	51	10	26,833	113,194	5
Berlin-Depot 35	5,545	1,769,839	222,475	130	16	32,036	132,442	5
Berlin-Depot 36	5,155	1,487,242	181,006	98	12	30,190	127,656	5
Berlin-Depot 37	4,405	1,572,817	156,316	131	12	21,477	132,346	5
Berlin-Depot 38	3,931	1,496,895	149,545	74	12	24,254	134,701	5
Berlin-Depot 39	4,099	1,499,119	188,056	80	16	25,168	168,289	7
Berlin-Depot 40	2,739	813,746	79,403	33	8	15,969	108,389	5
Berlin-Depot 41	3,131	977,644	96,296	40	9	19,075	127,846	5
Berlin-Depot 42	5,001	2,094,639	239,039	124	16	25,985	166,107	6
Berlin-Depot 43	3,441	1,141,457	119,167	55	6	21,201	76,146	4
Berlin-Depot 44	2,739	813,746	79,403	33	8	15,969	108,389	5
All depots of Berlin 1 together			7,836,189	4,722	498	1,220,005		

[a]CPU run times in seconds without reading the problems.
[b]Number of active columns of the last restricted arc set; note, there was no column elimination.
[c]Number of restricted problems that have been solved until optimality has been proved.

Table 12.6: Minimum-cost flow problems from BVG Depots 23–44.

Depots	$\lvert V\rvert$	$\lvert A\rvert$	MCF default		MCF with column generation			
			Simplex iterations	CPU time[a]	CPU time[a]	Simplex iterations	# Gener. columns[b]	# Restr. problems[c]
Hamburg 1 - Depot 1	5,801	2,170,962	316,034	310	33	62,590	447,943	3
Hamburg 1 - Depot 2	4,555	2,285,501	362,766	201	13	29,804	175,423	2
Hamburg 1 - Depot 3	3,433	850,828	77,838	63	13	26,835	154,521	3
Hamburg 1 - Depot 4	2,131	488,655	35,061	15	2	12,514	57,826	2
Hamburg 1 - Depot 5	2,827	339,891	29,331	31	3	14,600	79,827	3
Hamburg 1 - Depot 6	2,153	318,626	15,827	15	3	14,743	76,910	3
Hamburg 1 - Depot 7	1,457	233,010	11,522	6	1	6,480	29,141	2
Hamburg 1 - Depot 8	4,577	1,360,544	183,555	172	21	43,360	304,579	3
Hamburg 1 - Depot 9	3,765	1,579,315	180,625	96	10	22,982	134,646	2
Hamburg 1 - Depot 10	3,177	404,016	24,988	35	4	20,322	111,543	2
Hamburg 1 - Depot 11	2,593	459,468	35,402	25	5	25,113	109,078	3
Hamburg 1 - Depot 12	1,785	348,230	18,360	9	3	11,664	55,993	3
Hamburg 2 - Depot 5	1,297	180,554	7,898	6	1	5,532	19,718	2
Hamburg 2 - Depot 7	1,391	186,863	11,957	7	1	6,502	20,467	2
Hamburg 2 - Depot 8	987	104,153	3,339	3	1	3,828	10,232	2
Hamburg 2 - Depot 9	2,731	475,533	38,668	16	4	19,609	68,406	3
Hamburg 3 - Depot 1	1,043	65,650	10,777	2	1	6,761	26,999	2
Hamburg 3 - Depot 2	1,043	122,692	4,160	3	1	5,044	20,936	2
Hamburg 5 - Depot 1	1,973	211,153	16,507	11	3	18,282	84,589	2
Hamburg 5 - Depot 2	1,861	368,242	22,399	10	2	11,217	58,382	2
Hamburg 6 - Depot 1	2,691	322,546	25,023	26	10	19,951	136,053	3
Hamburg 6 - Depot 2	3,387	1,233,766	129,959	54	6	21,322	104,577	3
Others [d]	500	23,000	3,200	1	1	2,000	7,400	3
All depots of Hamburg together			1,579,268	1,123	147	421,583		

[a] CPU run times in seconds without reading the problems.
[b] Number of active columns of the last restricted arc set; note, there was no column elimination.
[c] Number of restricted problems that have been solved until optimality has been proved.
[d] The entries of each of the missing depot problems from HHA are always smaller then the given values in this row.

Table 12.7: Minimum-cost flow problems from HHA depots.

Depots	$\|V\|$	$\|A\|$	MCF with column generation							
			CPU time[a]		Simplex iterations	# Arcs / 1,000				# Restr. probs.[c]
						initially	generated[b]	eliminated	finally	
Berlin 1	49,813	69,700,000	877	(916)	271,398	846	5,614	4,120	2,340	5
Berlin 2	49,813	13,200,000	133	(229)	166,778	304	1,834	431	1,707	5
Berlin 3	2,627	2,300,000	13	(17)	13,491	77	2,000	1	2,076	3
B-Spandau 1	4,849	3,700,000	20	(27)	8,005	164	350	17	497	3
B-Spandau 2	6,617	8,800,000	74	(93)	37,520	327	997	9	1,315	4
B-Spandau 3	4,849	590,000	5	(9)	10,730	39	109	13	135	3
B-Spandau 4	6,617	1,530,000	21	(25)	35,355	72	323	27	368	4
B-Spandau 5	6,663	1,550,000	25	(31)	33,943	75	292	28	339	5
B-Spandau 6	3,997	380,000	4	(17)	9,564	28	92	18	102	3
B-Spandau 7	4,849	3,300,000	17	(23)	8,076	145	310	15	440	3
B-Spandau 8	6,617	7,800,000	60	(67)	34,613	283	816	8	1,091	4
Hamburg 1	17,127	10,900,000	145	(185)	78,923	1,322	1,751	278	2,795	3
Hamburg 2	3,669	1,000,000	6	(12)	8,163	99	123	0	222	2
Hamburg 3	1,583	200,000	3	(4)	4,426	30	43	0	73	2
Hamburg 4	477	23,000	1	(2)	575	2	2	0	4	3
Hamburg 5	2,923	580,000	7	(10)	10,387	85	148	0	233	2
Hamburg 6	4,567	1,600,000	14	(18)	10,566	176	175	0	351	2
Hamburg 7	683	34,000	1	(2)	1,462	6	9	0	17	2
H-Holstein 1	6,827	4,000,000	32	(40)	20,148	230	433	6	657	3
H-Holstein 2	10,895	9,400,000	54	(101)	61,598	1,054	547	161	1,440	4
All Lagrangean problems together			1,512		825,721					

[a]CPU run times in seconds without reading the problems. The times in brackets include reading.
[b]Number of generated pull-in-pull-out trips.
[c]Number of restricted problems that have been solved until optimality has been proved.

Table 12.8: Minimum-cost flow problems $LR_{fcs}(0)$.

12.4 Solving the Multiple-Depot Instances

In this section, we summarize the computational results of our multiple-depot instances. The following objective values (fleet sizes and operational weights) are given in Tab. 12.9: (i) the lower bounds obtained with $L_{\text{fcs}}(0)$ and the LP relaxations; (ii) the integer optimum or, if the optimum is still unknown, the best integer solution values; (iii) the upper bounds obtained by our opening heuristics as well as obtained by our branch-and-cut method starting with SCR or ND and terminating after a maximum run time limit of 10 hours (and 16 hours for Berlin 1 starting with ND).

The largest problem, Berlin 1, has not been solved to optimality. Berlin 2 and Berlin-Spandau 2 and 8 have been solved fleet minimally, but not to proven cost minimality. In Tab. 12.10 and 12.11, we give the lower bound values in proportion to the integer optimal values (or best known upper bound values) and the upper bound values in proportion to the integer optimal values (or the best known lower bound values). Note that the proportion for the operational weights is only considered for a problem if its corresponding fleet size gap is equal to zero. The run times that have been required to solve the function $L_{\text{fcs}}(0)$, the LP relaxation pure without LP-plunging, the opening heuristics SCR and ND, and our exact method (with and without using the optional LP-plunging within the column generation) are given in Tab. 12.12.

Lower Bounds.

For the Lagrangean relaxations, we have only considered $L_{\text{fcs}}(0)$. However, we have shown in Chap. 6 that $L_{\text{fcs}}(0)$ and $L_{\text{fcd}}(\nu^+ - \nu^-)$ provide the same optimal values (ν^+ and ν^- denote the optimal values of the dual multipliers associated with the flow conditions in $L_{\text{fcs}}(0)$). The values obtained by $L_{\text{fcs}}(0)$ give excellent approximations. The minimum integral fleet sizes can be approximated, on the average, by 99.94 %. It is remarkable that the trivial problem relaxation – simply neglecting the flow conservations – gives such tight approximations. For 15 out of our 20 instances, the fleet sizes can be exactly approximated. Ignoring for those problems the values for the fleet size, the gap between the operational costs of $L_{\text{fcs}}(0)$ and the optimum is at most 16 % and 5 % on the average. From our point of view, these gaps are acceptable, but we will show in the next section that they can be significantly decreased using in addition a subgradient method.

All LP relaxations, except for Berlin 1, have been solved to optimality. To find a fleet minimal LP value for Berlin 1, our column generation requires about 200 hours cpu time. The values obtained by the LP relaxation give lower bounds quite close to the integer optimal values. For 12 out of the 20 considered instances, the LP relaxation already provides the integer optimal value, and for 3 instances, it can be obtained by rounding up the LP value to the next integer value. For Berlin 1, we do not know the minimal number of vehicles, but expect that the fleet size lower bound provided by the LP relaxation is also tight. Whenever the LP relaxation provides an exact fleet size, it also provides the minimal operational weights.

Test Sets	Lower bounds				Optimum or best int. solution		Upper bounds							
	Lagrangean relax. $L_{\text{fcs}}(0)$		LP relaxation				Schedule – cluster – reschedule				Nearest depot heuristic			
							pure[a]		+ LP method[b]		pure[c]		+ LP method[d]	
	Fleet	Weight	Fleet	Weight	Fleet	Weight	Fleet	Weight	Fleet	Weight	Fleet	Weight	Fleet	Weight
Berlin 1	1323	715714	1323	759162.0[c]	1329	850680	1347	1317379	1335	1118287	1575	3279774	1356	982914
Berlin 2	1350	715623	1353.7	797918.8	**1354**	777823	1366	1318085	**1354**	809611	1655	3900191	**1354**	788958
Berlin 3	**69**	14043	**69**	**14119**	**69**	**14119**	**69**	14122	**69**	**14119**	70	14366	**69**	**14119**
B-Spandau 1	**125**	65585	**125**	65610.5	**125**	**65611**	**125**	125786	**125**	65835	139	70092	**125**	65901
B-Spandau 2	184	78947	184.5	79110.0	**185**	79052	**185**	289262	**185**	80430	207	213333	**185**	92249
B-Spandau 3	**127**	90514	**127**	**93745**	**127**	**93745**	**127**	152109	**127**	**93745**	135	149629	**127**	**93745**
B-Spandau 4	**191**	195844	**191**	**230846**	**191**	**230846**	192	395891	**191**	**230846**	222	475552	**191**	**230846**
B-Spandau 5	**191**	191141	**191**	**227580**	**191**	**227580**	194	393922	**191**	**227580**	220	487944	**191**	**227580**
B-Spandau 6	**98**	91109	**98**	**101075**	**98**	**101075**	**98**	132650	**98**	**101075**	109	139593	**98**	**101075**
B-Spandau 7	**125**	65585	**125**	65610.5	**125**	**65611**	**125**	105853	**125**	**65611**	139	70092	**125**	65724
B-Spandau 8	184	78947	184.5	79110.0	**185**	79093	**185**	259406	**185**	79273	207	213333	**185**	79959
Hamburg 1	**432**	66874	**432**	71068.3	**432**	**71069**	446	70291	434	73066	489	76054	**432**	71270
Hamburg 2	**103**	15356	**103**	**16070**	**103**	**16070**	104	16792	**103**	**16070**	114	15849	**103**	**16070**
Hamburg 3	**39**	5557	**39**	**5860**	**39**	**5860**	**39**	6298	**39**	**5860**	41	5429	**39**	**5860**
Hamburg 4	**6**	**1358**	**6**	**1358**	**6**	**1358**	**6**	**1358**	**6**	**1358**	**6**	**1358**	**6**	**1358**
Hamburg 5	**62**	12092	**62**	**12502**	**62**	**12502**	**62**	13535	**62**	**12502**	65	12101	**62**	**12502**
Hamburg 6	**111**	15705	**111**	**15791**	**111**	**15791**	**111**	16588	**111**	**15791**	**111**	**15791**	**111**	**15791**
Hamburg 7	**15**	2832	**15**	**2961**	**15**	**2961**	16	2836	**15**	**2961**	16	2836	**15**	**2961**
H-Holstein 1	**201**	28697	**201**	**29027**	**201**	**29027**	**201**	30497	**201**	**29027**	213	32132	**201**	**29027**
H-Holstein 2	360	51084	**362**	52787.7	**362**	**52788**	363	72700	**362**	52986	393	62598	**362**	53090

[a]Results obtained with the opening heuristics SCR and ND, respectively.

[b]Results obtained with our LP method including branch-and-cut, starting with SCR or ND, using LP-plunging, and terminating after a run time limit of ten hours (and 16 hours for Berlin 1 starting with ND).

[c]Best known feasible LP value.

Table 12.9: Fleet sizes and operational costs (optimal integer values are in bold face).

Test Sets	Lower bounds / Optimum or Best integer solution: Lagrangean relaxation $L_{\text{fcs}}(0)$ Fleet	Lagrangean relaxation $L_{\text{fcs}}(0)$ Weight	LP relaxation Fleet	LP relaxation Weight
Berlin 1	0.9954	—	0.9954	—
Berlin 2	0.9970	—	0.9997	—
Berlin 3	**1.0000**	0.9946	**1.0000**	**1.0000**
B-Spandau 1	**1.0000**	0.9996	**1.0000**	1.0000
B-Spandau 2	0.9946	—	0.9973	—
B-Spandau 3	**1.0000**	0.9655	**1.0000**	**1.0000**
B-Spandau 4	**1.0000**	0.8484	**1.0000**	**1.0000**
B-Spandau 5	**1.0000**	0.8399	**1.0000**	**1.0000**
B-Spandau 6	**1.0000**	0.9014	**1.0000**	**1.0000**
B-Spandau 7	**1.0000**	0.9996	**1.0000**	1.0000
B-Spandau 8	0.9946	—	0.9973	—
Average	0.9983	0.9356	0.9991	1.0000
$\sqrt{\sigma_n}$[a]	0.0023	0.0660	0.0015	0.0000
Hamburg 1	**1.0000**	0.9410	**1.0000**	1.0000
Hamburg 2	**1.0000**	0.9556	**1.0000**	**1.0000**
Hamburg 3	**1.0000**	0.9483	**1.0000**	**1.0000**
Hamburg 4	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Hamburg 5	**1.0000**	0.9672	**1.0000**	**1.0000**
Hamburg 6	**1.0000**	0.9946	**1.0000**	**1.0000**
Hamburg 7	**1.0000**	0.9564	**1.0000**	**1.0000**
H-Holstein 1	**1.0000**	0.9886	**1.0000**	**1.0000**
H-Holstein 2	**0.9945**	—	**1.0000**	1.0000
Average	0.9994	0.9690	1.0000	1.0000
$\sqrt{\sigma_n}$	0.0017	0.0211	0.0000	0.0000
All Average	0.9988	0.9534	0.9995	1.0000
All $\sqrt{\sigma_n}$	0.0021	0.0505	0.0012	0.0000

[a]Standard deviation

Table 12.10: Lower bounds relative to the integer optimal values or best upper bounds. Optimal values are in bold face.

Test Sets	Upper bounds / Optimum or Best integer solution							
	Schedule – cluster – reschedule				Nearest depot heuristic			
	pure[a]		+ LP method[b]		pure[a]		+ LP method[b]	
	Fleet	Weight	Fleet	Weight	Fleet	Weight	Fleet	Weight
Berlin 1	1.0181	—	1.0091	—	1.1851	—	1.0249	—
Berlin 2	1.0089	—	**1.0000**	—	1.2223	—	**1.0000**	—
Berlin 3	**1.0000**	1.0002	**1.0000**	**1.0000**	1.0145	—	**1.0000**	**1.0000**
B-Spandau 1	**1.0000**	1.9171	**1.0000**	1.0034	1.1120	—	**1.0000**	1.0044
B-Spandau 2	**1.0000**	—	**1.0000**	—	1.1189	—	**1.0000**	—
B-Spandau 3	**1.0000**	1.6225	**1.0000**	**1.0000**	1.0630	—	**1.0000**	**1.0000**
B-Spandau 4	1.0052	—	**1.0000**	**1.0000**	1.1623	—	**1.0000**	**1.0000**
B-Spandau 5	1.0157	—	**1.0000**	**1.0000**	1.1518	—	**1.0000**	**1.0000**
B-Spandau 6	**1.0000**	1.3124	**1.0000**	**1.0000**	1.1122	—	**1.0000**	**1.0000**
B-Spandau 7	**1.0000**	1.6133	**1.0000**	**1.0000**	1.1120	—	**1.0000**	1.0017
B-Spandau 8	**1.0000**	—	**1.0000**	—	1.1189	—	**1.0000**	—
Average	1.0044	—	1.0008	1.0005	1.1248	—	1.0023	1.0009
$\sqrt{\sigma_n}$[c]	0.0065	—	0.0026	0.0012	0.0538	—	0.0072	0.0016
Hamburg 1	1.0324	—	1.0046	—	1.1319	—	**1.0000**	1.0028
Hamburg 2	1.0097	—	**1.0000**	**1.0000**	1.1068	—	**1.0000**	**1.0000**
Hamburg 3	**1.0000**	1.0747	**1.0000**	**1.0000**	1.0513	—	**1.0000**	**1.0000**
Hamburg 4	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Hamburg 5	**1.0000**	1.0826	**1.0000**	**1.0000**	1.0484	—	**1.0000**	**1.0000**
Hamburg 6	**1.0000**	1.0505	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Hamburg 7	1.0667	—	**1.0000**	**1.0000**	1.0667	—	**1.0000**	**1.0000**
H-Holstein 1	**1.0000**	1.0506	**1.0000**	**1.0000**	1.0597	—	**1.0000**	**1.0000**
H-Holstein 2	1.0028	—	**1.0000**	1.0038	1.0856	—	**1.0000**	1.0057
Average	1.0124	1.0517	1.0005	1.0005	1.0612	—	1.0000	1.0009
$\sqrt{\sigma_n}$	0.0216	0.0288	0.0014	0.0013	0.0414	—	0.0000	0.0019
All Average	1.0080	—	1.0007	1.0005	1.0962	—	1.0012	1.0009
All $\sqrt{\sigma_n}$	0.0158	—	0.0021	0.0012	0.0580	—	0.0054	0.0018

[a]Results obtained by the opening heuristics SCR and ND, respectively.

[b]Results obtained with our LP method including branch-and-cut, starting with SCR or ND, using LP-plunging, and terminating after a run time limit of ten hours (and 16 hours for Berlin 1 starting with ND).

[c]Standard deviation

Table 12.11: Upper bounds relative to the integer optimal values or best lower bounds. Optimal values are in bold face.

Table 12.12: Run times in seconds.

Test Sets	Lower bounds			Upper bound (or optimum)					
	Lagrangean relaxation: $L_{\mathrm{fcs}}(0)$	LP relaxation[a] starting with		Schedule – cluster – reschedule			Nearest depot heuristic		
					Exact method			Exact method	
					LP-plunging			LP-plunging	
		SCR	ND	pure	on[b]	off[c]	pure	on[b]	off[c]
Berlin 1	916	—[d]	—[d]	12386	—[d]	—[d]	171	—[d]	—[d]
Berlin 2	229	34795	32767	3810	—[d]	35202[e]	79	30985[e]	33248[e]
Berlin 3	17	431	311	25	389	435	7	249	330
B-Spandau 1	27	43777	66501	343	59337	44053	15	68487	134386
B-Spandau 2	93	112337	165048	1939	—[d]	138212[e]	39	—[d]	240852[e]
B-Spandau 3	9	975	739	42	1626	990	7	953	758
B-Spandau 4	25	6014	4384	334	6228	6077	14	6773	4433
B-Spandau 5	31	5264	5618	354	6896	5304	15	13821	5666
B-Spandau 6	17	162	227	7	211	173	6	188	244
B-Spandau 7	23	24717	46597	259	26606	24793	14	45810	52031
B-Spandau 8	67	82284	62041	1238	146284[e]	84725[e]	36	—[d]	63215[e]
Hamburg 1	185	—[d]	50246	2868	—[d]	—[d]	29	88767	53971
Hamburg 2	12	875	685	103	732	902	7	926	708
Hamburg 3	4	35	31	16	37	41	2	34	38
Hamburg 4	2	3	3	1	3	3	1	3	3
Hamburg 5	10	258	155	86	288	279	5	119	174
Hamburg 6	18	148	84	30	158	181	11	124	86
Hamburg 7	2	8	9	2	11	10	1	8	11
H-Holstein 1	40	2619	2087	199	2166	2695	18	2445	2158
H-Holstein 2	101	46673	64489	1696	55915	54604	40	68534	71562

[a] LP times pure without LP-plunging.
[b] Our method using LP-plunging.
[c] Our method using LP-plunging only within branch-and-cut, but not between two successive RLPs.
[d] Not solved to optimality since, for instance, the objective progress was too small or stalling occurred.
[e] The problem has been solved fleet minimally, but not to proven cost minimality.

We have seen that the LP values are quite tight. A similar phenomenon is observed by Forbes, Holt, and Watts [1994]: 22 of their 30 test instances with up to 600 timetabled trips have integral LP solutions, and the largest gap between the LP value and the integral optimum is at most 0.003 % for the remaining problems. So, this observation does not seem to be a *small scale phenomenon*.

The value of the operational weights in the objective value of the lower bounds do not necessarily define lower bounds for the integer optimal weights among all minimal fleet solutions. To estimate the quality of the operational weights requires that the lower bound of the fleet size is tight! For all problems that do not satisfy this condition, however, we believe that they nevertheless give good estimated values for the minimal operational weights.

Comparing the run times of the Lagrangean and LP relaxation, it is obvious that Lagrangean relaxations $L_{\text{fcs}}(0)$ are the faster method to obtain good lower bounds quickly. The better lower bounds provided by the LP relaxation require long run times that are only justified by a succeeding branch-and-cut method. The solution produced by SCR are always significantly better than those of ND. On the average, however, SCR used as the opening heuristic for the branch-and-cut algorithm does not provide better starting points. It is worth mentioning that starting without any heuristically generated solution, our LP method is unable to solve any of our larger problem instances at all.

Upper Bounds.

We will now consider the upper bounds obtained by the two opening heuristics (SCR and ND) and obtained by the exact branch-and-bound method starting with SCR and ND, using LP-plunging between two RLPs, and terminating after a given run time limit of 10 hours (and 16 hours for Berlin 1 starting with ND).

The trivial opening heuristic ND already delivers good results: The fleet size gap is, on the average, about 10 % with a standard deviation of 6 %. From a practical point of view, however, the operational costs of these solutions are unacceptable. The better results are obtained from the SCR heuristic: The average fleet size gap is 0.8 % with a standard deviation of 1.6 %. The operational costs of these solutions are comparable to the results obtained by the best codes currently used in practice.

We almost always obtain optimal results if we apply our exact branch-and-cut method with a time limit of 10 hours. The objective gaps are, on the average, less than 0.12 %. It does not make any difference which opening heuristic we use for the exact method since the run times are comparable for both. The run times of our exact method may be decreased if we use both opening heuristics together to determine the first RLP. This may be the basis for further computational tests.

Figures 12.1–12.4 display the development of the upper bound values (fleet sizes and operational weights) obtained by the LP-plunging heuristic in proportion to the integer optimal (or lower bound) values. Starting our method with the solution obtained with

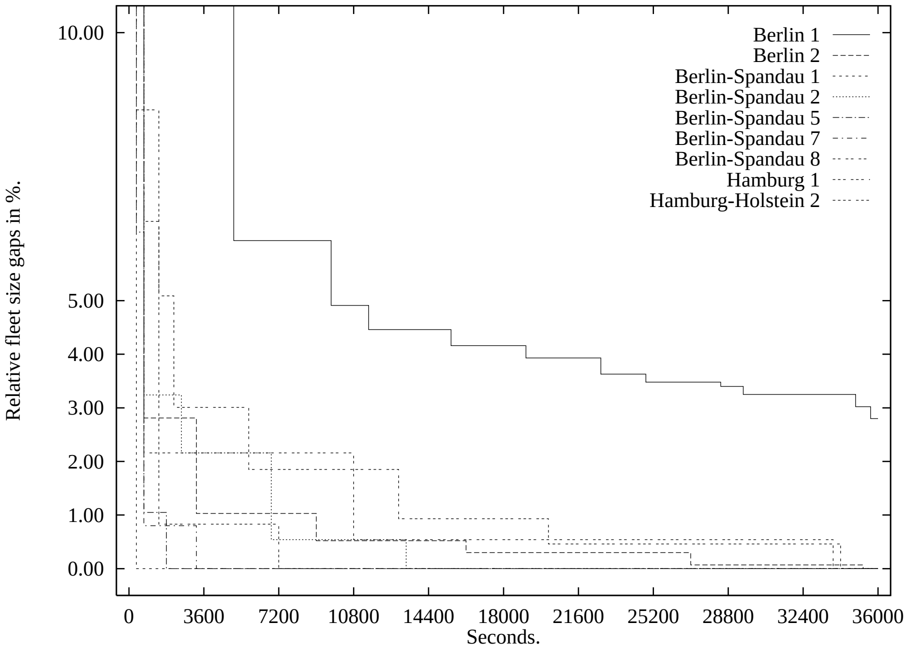


Figure 12.1: Development of fleet size upper bounds of problems requiring more than 2 hours run time to obtain a minimal fleet solution; starting with the ND heuristic.

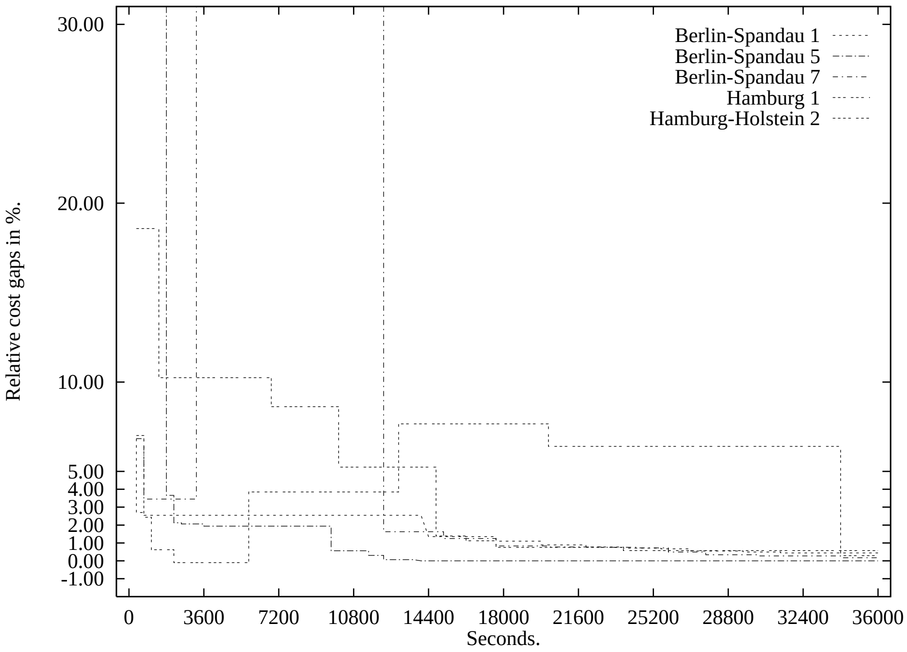


Figure 12.2: Development of operational weight upper bounds of problems requiring more than 2 hours run time to obtain a minimal fleet solution and knowing the minimum weight among all minimal fleet solutions; starting with the ND heuristic.

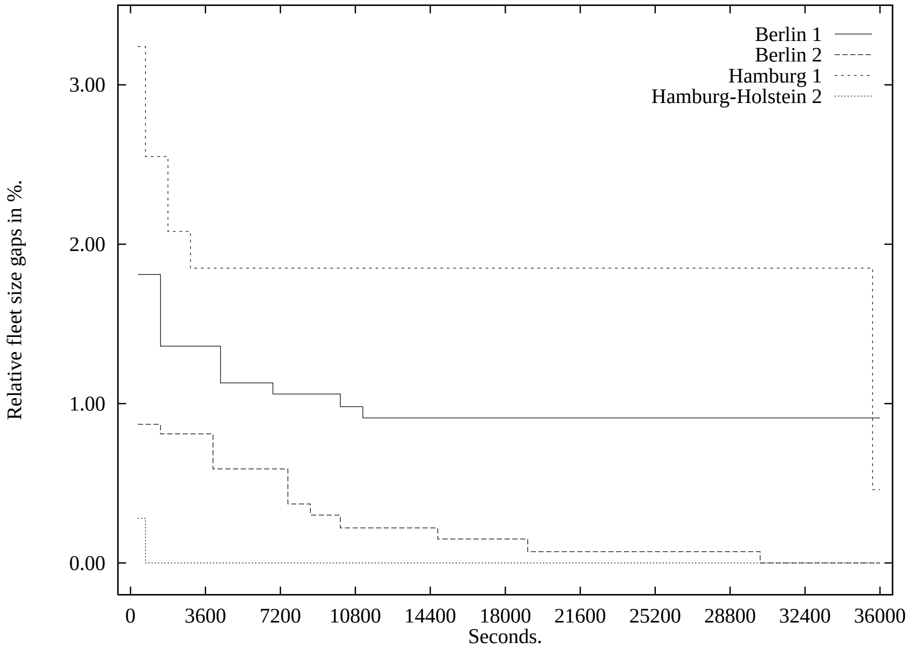


Figure 12.3: Development of fleet size upper bounds of problems requiring more than 2 hours run time to obtain the optimum; starting with the SCR heuristic.

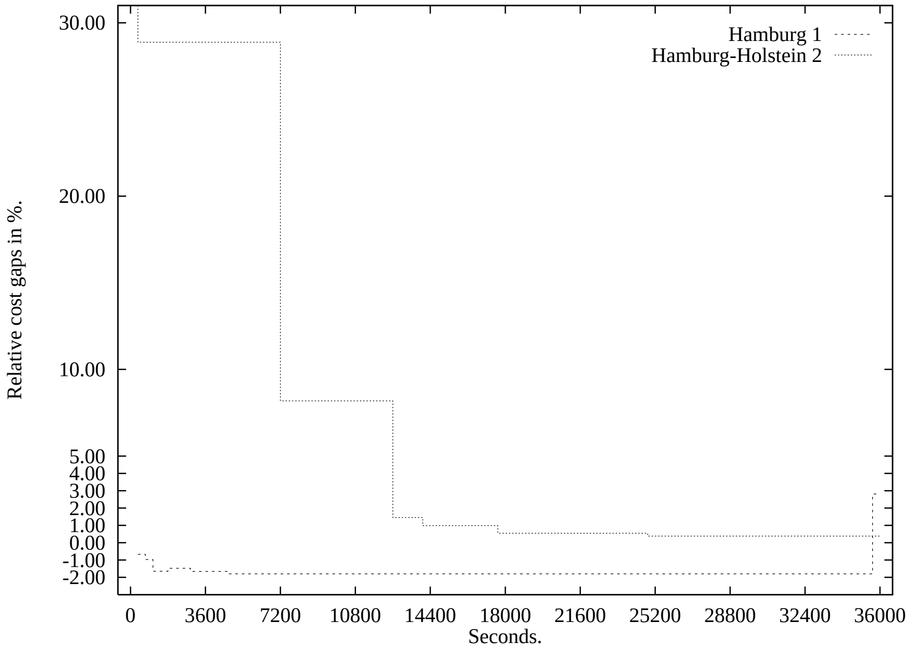


Figure 12.4: Development of operational weight upper bounds of problems requiring more than 2 hours run time to obtain the optimum and knowing the minimum weight among all minimal fleet solutions; starting with the SCR heuristic.

ND, the fleet sizes can be approximated in two hours with a gap less than 3 %, in 4 four hours with gap of about 1 %, and in 6 hours with a gap less than 1 % for all problems except Berlin 1, see Fig. 12.1. Starting with the solution obtained with SCR, the fleet sizes can be approximated in one hour with a gap less than 2 % and in 10 hours with a gap less than 1 %, see Fig. 12.3. There is also a positive development of the operational costs: Compared to the optimal integer costs of minimal fleet solutions, the operational costs can be approximated with a small gap, see Figs. 12.2 and 12.4. If the run time limit is 10 hours or more, the four figures show that it is meaningless which opening heuristic is used, the results are in any case comparable. However, if there is a stronger time limit of two or three hours, starting with SCR provides better results.

Optimal Solutions

Without any run time limit, each instance of our test set, with the exception of the problem Berlin 1, can be solved to proven fleet minimality. With the exceptions of the problems Berlin 2, Berlin-Spandau 2, and Berlin-Spandau 8, each instance can be solved to proven fleet and cost optimality.

With the current version of our branch-and-cut method, solving really large-scale problems to proven optimality leads to impractical run times. In particular, solving Berlin 1 with 70 million variables to optimality is still a challenge to us. Nevertheless, the results obtained with our methods are currently the best obtainable. Solutions providing possibly a gap of a few vehicles, but with reasonable operational weights can be computed in acceptable run times.

12.5 Subgradient Method

In the following, we give some detailed computational results about the subgradient methods that we have applied to solve the Lagrangean relaxations. In Table 12.13 we summarize the operational weight of the investigated subgradient method with different parameter settings (default and best configuration obtained in Kokott [1996]) for $\mathrm{LR}_{\mathrm{fcs}}$ and $\mathrm{LR}_{\mathrm{fcd}}$ in comparison with $L_{\mathrm{fcs}}(0)$. In addition, we also give the weight gaps relative to the integer optimum costs among all minimal fleet solutions.

The results in the column "$\mathrm{LR}_{\mathrm{fcs}}(0)$" are the objective weights obtained by neglecting the flow conservation constraints. The results in the column "Default" illustrate the computational results for parameter settings as follows: A maximal number of iterations $N_1 := 100$, step length rule A with maximal number of consecutive failures $N_2 := 2$, and a step direction $\tilde{g}^{(k)} := 0.6g^{(k)} + 0.2g^{(k-1)} + 0.1g^{(k-2)} + 0.1g^{(k-3)}$. The initial step length parameters $\sigma^{(0)}$ and $\alpha^{(0)}$ depend completely on the considered problem instance, and we are not able to give any good value in advance without making test runs. Therefore, we have decided to use $\sigma^{(0)} := 10$ for the step length rule A and to use $\alpha^{(0)}$ such that the resulting $\sigma^{(0)}$ is about 10 for step length rule B. On the average, B does not yield

Test Sets	Fleet	Lagrangean relaxations & subgradient methods									
		Weights					Weight gaps to the optimum in %				
			Default		Kokott [1996]			Default		Kokott [1996]	
		$L_{\mathrm{fcs}}(0)$	$\mathrm{LR}_{\mathrm{fcs}}$	$\mathrm{LR}_{\mathrm{fcd}}$	$\mathrm{LR}_{\mathrm{fcs}}$	$\mathrm{LR}_{\mathrm{fcd}}$	$L_{\mathrm{fcs}}(0)$	$\mathrm{LR}_{\mathrm{fcs}}$	$\mathrm{LR}_{\mathrm{fcd}}$	$\mathrm{LR}_{\mathrm{fcs}}$	$\mathrm{LR}_{\mathrm{fcd}}$
Berlin 1	1323	715714	708008	704666	—	—	—	—	—	—	—
Berlin 2	1350	715623	719952	718463	—	—	—	—	—	—	—
Berlin 3	**69**	14043	14075	14105	14101	14111	0.54	0.31	0.10	0.13	0.06
B-Spandau 1	**125**	65585	64984	65021	65586	65586	0.03	1.00	0.90	0.04	0.04
B-Spandau 2	184	78947	78204	77342	79038	78963	—	—	—	—	—
B-Spandau 3	**127**	90514	92711	92298	93461	93534	3.45	1.10	1.54	0.30	0.22
B-Spandau 4	**191**	195844	197786	203178	205752	209023	15.16	14.32	11.98	10.87	9.45
B-Spandau 5	**191**	191141	194436	197965	202886	207237	16.01	14.56	13.01	10.85	8.94
B-Spandau 6	**98**	91109	92490	91797	98392	92700	9.86	8.49	9.18	2.65	8.29
B-Spandau 7	**125**	65585	65135	65427	—	—	0.03	0.72	0.28	—	—
B-Spandau 8	184	78947	78657	78072	—	—	—	—	—	—	—
Hamburg 1	**432**	66874	69728	68447	70266	68447	5.90	1.89	3.69	1.13	3.69
Hamburg 2	**103**	15356	15769	15759	15814	15769	4.44	1.87	1.93	1.59	1.87
Hamburg 3	**39**	5557	5796	5733	5796	**5860**[a]	5.17	1.09	2.16	1.09	0.00
Hamburg 4	**6**	**1358**[b]	—	—	—	—	0.00	0.00	0.00	0.00	0.00
Hamburg 5	**62**	12092	12430	12374	12439	**12502**	3.27	0.57	1.02	0.50	0.00
Hamburg 6	**111**	15705	**15791**	**15791**[c]	**15791**[d]	**15791**	0.54	0.00	0.00	0.00	0.00
Hamburg 7	**15**	2832	**2961**	2959	**2961**[e]	**2961**[f]	4.36	0.00	0.07	0.00	0.00
H-Holstein 1	**201**	28697	28860	28916	28972	29018	1.13	0.57	0.38	0.19	0.03

[a] Feasible and optimal after 65 iterations.
[b] Already feasible and optimal.
[c] Feasible and optimal after 24 iterations.
[d] Feasible and optimal after 55 iterations.
[e] Feasible and optimal after 31 iterations.
[f] Feasible and optimal after 50 iterations.

Table 12.13: Objective values using subgradient methods (optimal integer values are bold faced) and their operational gaps relative to the optimum.

better results for our test data. Thus, we give only the computational results for the first strategy, which, in addition, requires no upper bound. The results in the columns "Kokott [1996]" are the values of the best parameter configurations given in the masters thesis of Andreas Kokott.

The most surprising result is that $L_{\text{fcs}}(0)$ almost always provides a tight lower bound for the fleet size, and the largest gap here is at most 0.01%. Rounding up the fleet sizes given by the LP relaxation to the next integer value, the gap to the optimal fleet size is always zero. Hence, we consider our problems to be well conditioned in some sense, which we see to be confirmed by the computational results of our subgradient methods. The gap between the initial Lagrangean function value and the optimum is already less than 16.01% for $L_{\text{fcs}}(0)$. With an appropriate parameter configuration for the subgradient method, these gaps can be best reduced to less than 10%.

On the average, neither LR_{fcs} nor LR_{fcd} yield better lower bounds for our test set. However, LR_{fcd} can be solved faster for our large problem instances, see Tab. 12.14. Additionally, L_{fcd} is decomposable and, thus, its solution can be accelerated by parallelization.

Test Sets	Lagrangean relaxation		
	$L_{\text{fcs}}(0)$	LR_{fcs}	LR_{fcd}
Berlin 1	916	49258	31602
Berlin 2	229	11051	4819
Berlin 3	17	365	417
B-Spandau 1	27	1501	1172
B-Spandau 2	93	4447	3834
B-Spandau 3	9	357	236
B-Spandau 4	25	1116	858
B-Spandau 5	31	1037	818
B-Spandau 6	17	189	142
B-Spandau 7	23	1367	1063
B-Spandau 8	67	3919	3118
Hamburg 1	185	6838	6643
Hamburg 2	12	386	415
Hamburg 3	4	52	89
Hamburg 4	2	2	2
Hamburg 5	10	221	332
Hamburg 6	18	198	267
Hamburg 7	2	7	12
H-Holstein 1	40	1371	1919

Table 12.14: Run times in seconds.

Considering a fixed parameter configuration, the behaviour of the Lagrangean functions within the subgradient methods are similar for almost all test instances. We exemplary give the results varying single parameters of our default parameter configuration just for the problem Hamburg 1.

Figures 12.5 and 12.6 show the influence of the number of used subgradients for the step direction: "1 subgradient" means $\tilde{g}^{(k)} := g^{(k)}$, "2 subgradients" means $\tilde{g}^{(k)} := 0.7g^{(k)} + 0.3g^{(k-1)}$, and "4 subgradients" means $\tilde{g}^{(k)} := 0.6g^{(k)} + 0.2g^{(k-1)} + 0.1g^{(k-2)} + 0.1g^{(k-3)}$. Using more than one subgradient leads to a faster convergence and, in general, yields better lower bounds. As we can see from Figs. 12.7 and 12.8, both presented step size rules perform comparable.

The last Fig. 12.9 shows the behaviour for different maximal allowed consecutive failures N_2. As we can see, the Lagrangean function improves very fast in the first iterations for $N_2 = 1$, but then no further significant improvements are achieved since the step length is reduced far too fast. For $N_2 = 3$, the step lengths is reduced only once or twice such that the subgradient method cannot converge to any optimal solution. Empirically, we can claim that $N_2 = 2$ is a good compromise.

Theoretically, the Lagrangean relaxations $\mathrm{LR}_{\mathrm{fcs}}$ and $\mathrm{LR}_{\mathrm{fcd}}$ and the LP relaxation yield the same optimal value. Nevertheless, our subgradient methods do not attain the optimal values for the large instances, but the resulting gaps are rather small. The outstanding advantage of Lagrangean relaxations is that they provide lower bounds quickly. In particular, the subgradient methods provide a lower bound at every iterations. In contrast to Lagrangean relaxations, the column generation method for the LP relaxation provides a lower bound only if it has been solved to optimality, which can take a long time.

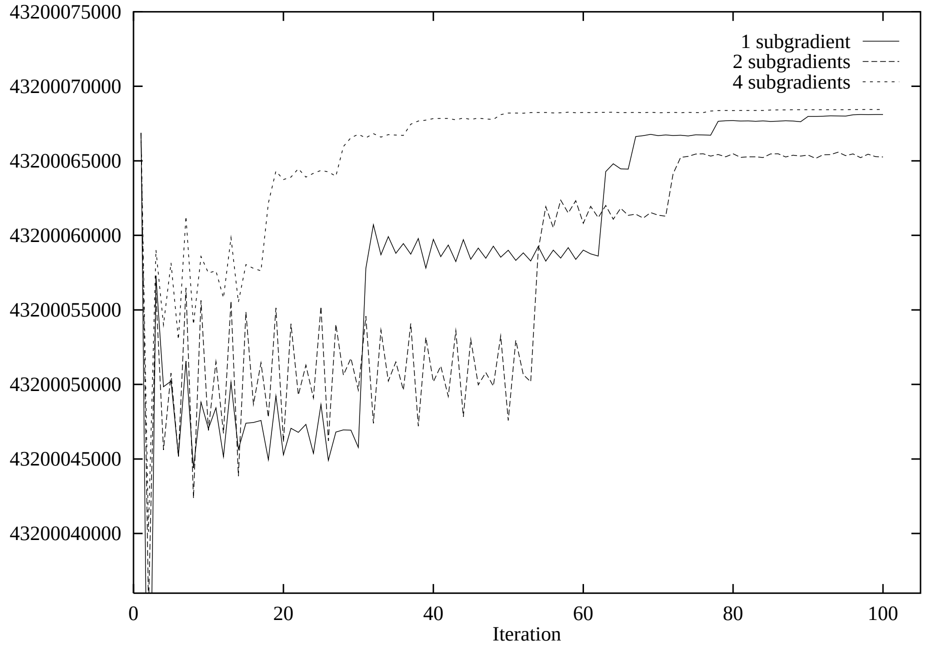


Figure 12.5: Comparing different step directions for Hamburg 1 and $\mathrm{LR}_{\mathrm{fcd}}$.

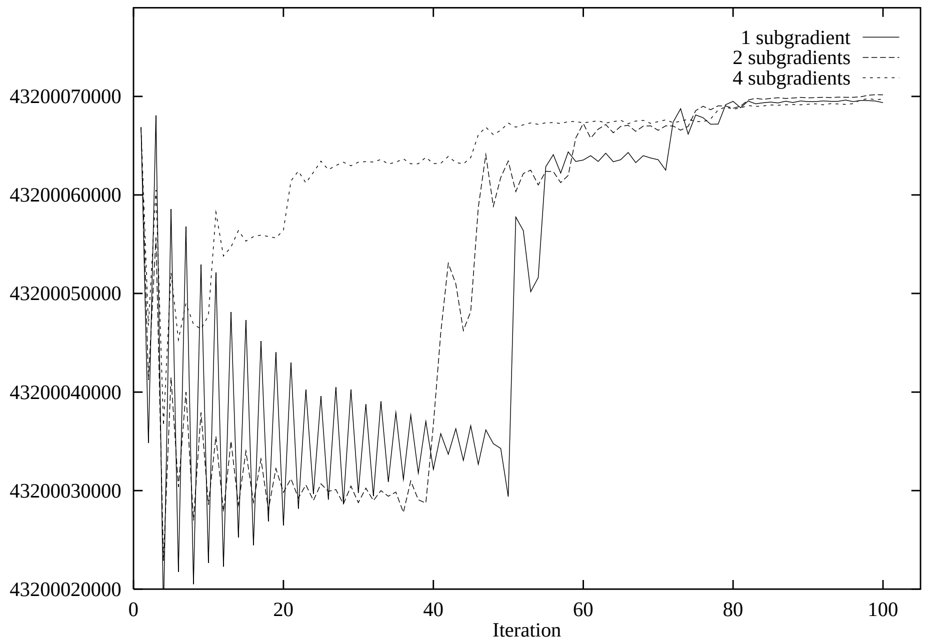


Figure 12.6: Comparing different step directions for Hamburg 1 and $\mathrm{LR}_{\mathrm{fcs}}$.

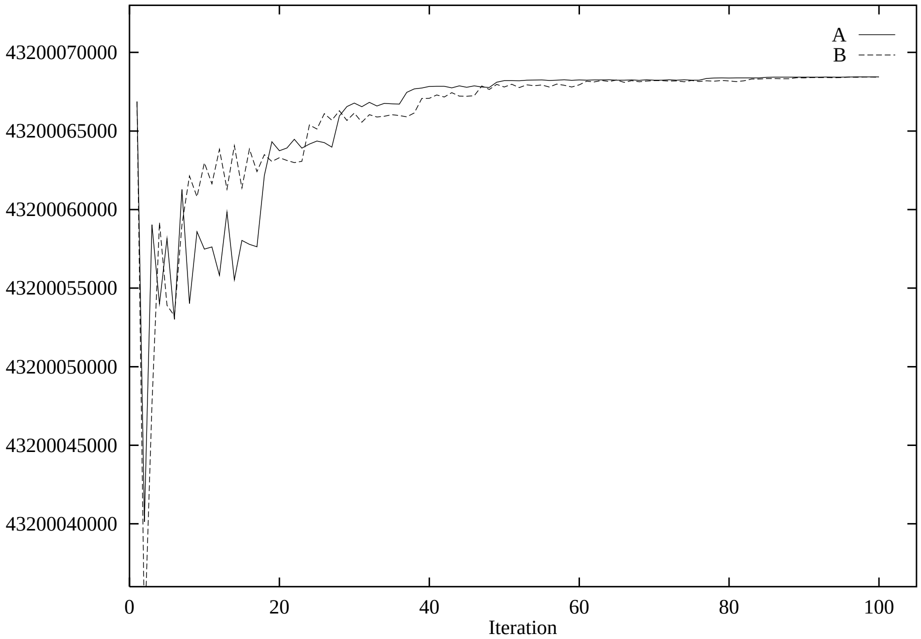


Figure 12.7: Comparing the two step length rules for Hamburg 1 and $\mathrm{LR}_{\mathrm{fcd}}$.

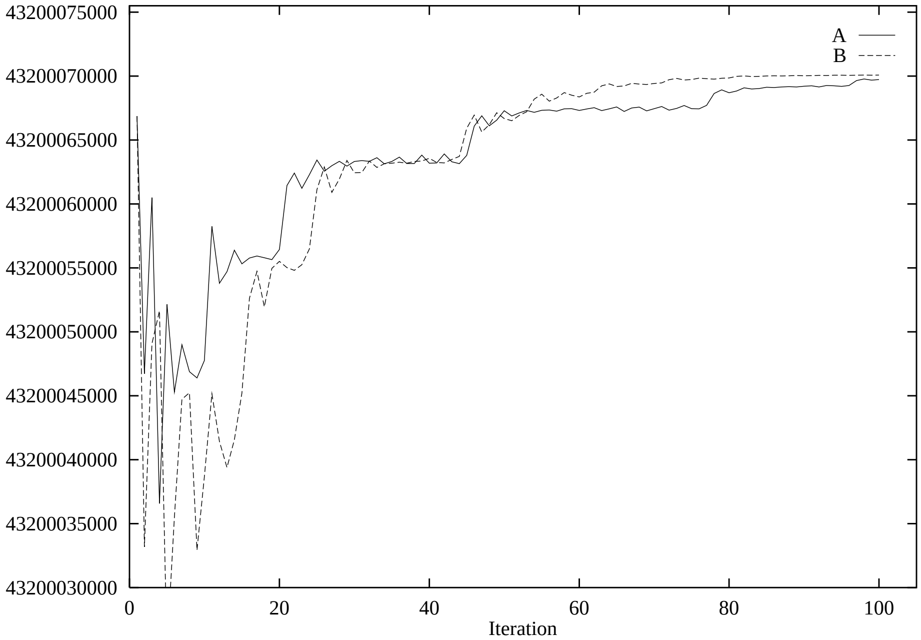


Figure 12.8: Comparing the two step length rules for Hamburg 1 and $\mathrm{LR}_{\mathrm{fcs}}$.

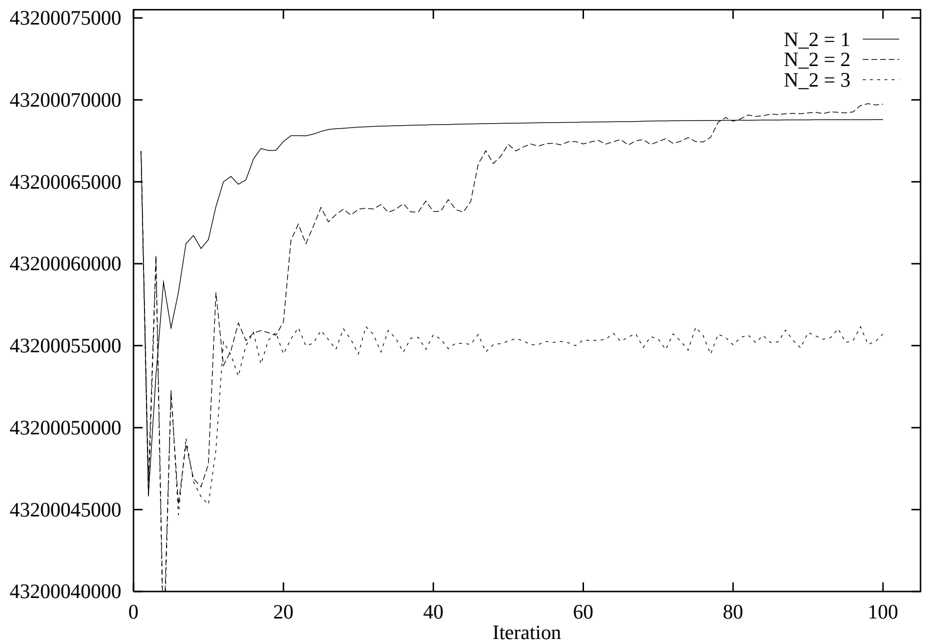


Figure 12.9: Comparing different maximum failure values N_2 for Hamburg 1 and $\mathrm{LR}_{\mathrm{fcs}}$.

12.6 LP Relaxation

In the following, we report on some specific observations we made in solving the LP relaxation without LP-plunging. Table 12.15 shows, for each test instance and for each opening heuristic, the number of RLPs that have been solved until optimality has been proved, the total number of CPLEX iterations that have been performed, and the number of columns that have been generated and eliminated within the column generation process. We shall describe some features of our column generation method with the examples of Berlin 2 starting with SF-CS and Berlin-Spandau 8 and Hamburg 1 both starting with ND. The behaviour of our implementation starting with SF-CS or ND is similar for all the other problems.

We have observed the following: The dominating part of the objective values, the fleet sizes, converge quickly to the minimum value within few iterations and a relatively small part of the total run times. At the *crossover* from the Lagrangean to the standard phase, the objective values yield always the minimal fleet size. The standard phase is necessary to solve the lower part of the objective values, the operational costs, to optimality. For our test set, neither the standard nor the Lagrangean phase dominates the total run time and the number of RLPs. Figures 12.10 and 12.11 show a typical development of the fleet size values (left pictures) and their operational weights (right pictures) in respect to the k^{th} RLP and the run time, respectively.

The number of generated and eliminated columns are almost always about the same for each iteration of the column generation process, see the left pictures of Fig. 12.12. Therefore, the LP sizes are relatively constant during the solution process. The right pictures in this figure show typical developments of the number of rows, columns, and nonzero elements of the k^{th} RLP and of the k^{th} RLP in the Lagrangean phase after LP preprocessing. We can observe from these picture the importance of LP preprocessing within the Lagrangean phase: Without this preprocessing, neither the primal nor the dual simplex solver would not be applicable here because of intolerable long run times. But preprocessing reduces the sizes of these RLPs significantly such that it becomes possible to solve the occurring RLPs within acceptable run times.

The typical number of dual and primal CPLEX iterations that have been performed for the k^{th} RLP are given in Fig. 12.13. In the Lagrangean phase, each RLP is always solved from scratch. In the standard phase, each RLP is always reoptimized using the last optimal basis as the starting point. Therefore, the number of performed simplex iterations is usually much larger in the Lagrangean phase. For the example of Berlin-Spandau 8, however, numerical difficulties occurred for the iterations 17 – 20: the RLPs have been reperturbed two times tribling the number of iterations and run times for these four RLPs. Without this numerical difficulties, Berlin-Spandau 8 would behave as all the other problems.

Test Sets	Starting with SF-CS								Starting with ND							
	RLP's			CPLEX			Col. gen.		RLP's			CPLEX			Col. gen.	
	Number of			Iterations / 1000			Cols. / 1000		Number of			Iterations / 1000			Cols. / 1000	
		Lgr.[a]	Stn.[b]		Lgr.[a]	Stn.[b]	gen.	elim.		Lgr.[a]	Stn.[b]		Lgr.[a]	Stn.[b]	gen.	elim.
Berlin 1	—	24	—	—	6.333	—	(967)	(830)	—	—	—	—	—	—	—	—
Berlin 2	97	25	72	501	307	194	691	634	106	27	79	561	371	190	716	658
Berlin 3	24	2	22	24	1	23	41	36	23	5	18	18	6	12	55	50
B-Spandau 1	23	12	11	881	615	266	250	227	30	16	14	1.399	1.091	308	293	274
B-Spandau 2	26	15	11	1.260	628	632	458	430	28	14	14	1.762	890	872	453	412
B-Spandau 3	34	13	21	70	57	13	211	202	30	11	19	61	49	12	186	177
B-Spandau 4	40	17	23	230	188	42	348	335	46	17	29	178	130	48	336	324
B-Spandau 5	47	14	33	191	135	56	323	310	49	17	32	223	161	62	349	337
B-Spandau 6	20	4	16	13	5	8	70	63	25	7	18	21	15	6	99	92
B-Spandau 7	34	12	22	594	403	191	207	190	30	16	14	1.069	869	200	247	227
B-Spandau 8	28	13	15	1.073	512	561	378	350	26	15	11	966	575	391	279	248
Hamburg 1	—	38	—	—	1.167	—	(296)	(273)	98	24	74	873	656	217	478	451
Hamburg 2	26	9	17	38	21	17	128	122	20	8	12	35	21	14	113	107
Hamburg 3	13	2	11	1	1	0	19	17	23	2	21	1	0	1	19	17
Hamburg 4	3	2	1	0	0	0	1	0	3	2	1	0	0	0	1	0
Hamburg 5	25	6	19	7	4	3	71	67	44	2	42	11	1	10	126	212
Hamburg 6	12	2	10	4	1	3	37	32	2	2	0	1	1	0	12	9
Hamburg 7	13	3	10	1	0	1	9	8	21	3	18	1	0	1	13	12
H-Holstein 1	20	13	7	97	79	18	190	177	16	8	8	79	63	16	125	111

[a]Lagrangean phase in our column generation.
[b]Standard phase in our column generation.

Table 12.15: LP run statistics.

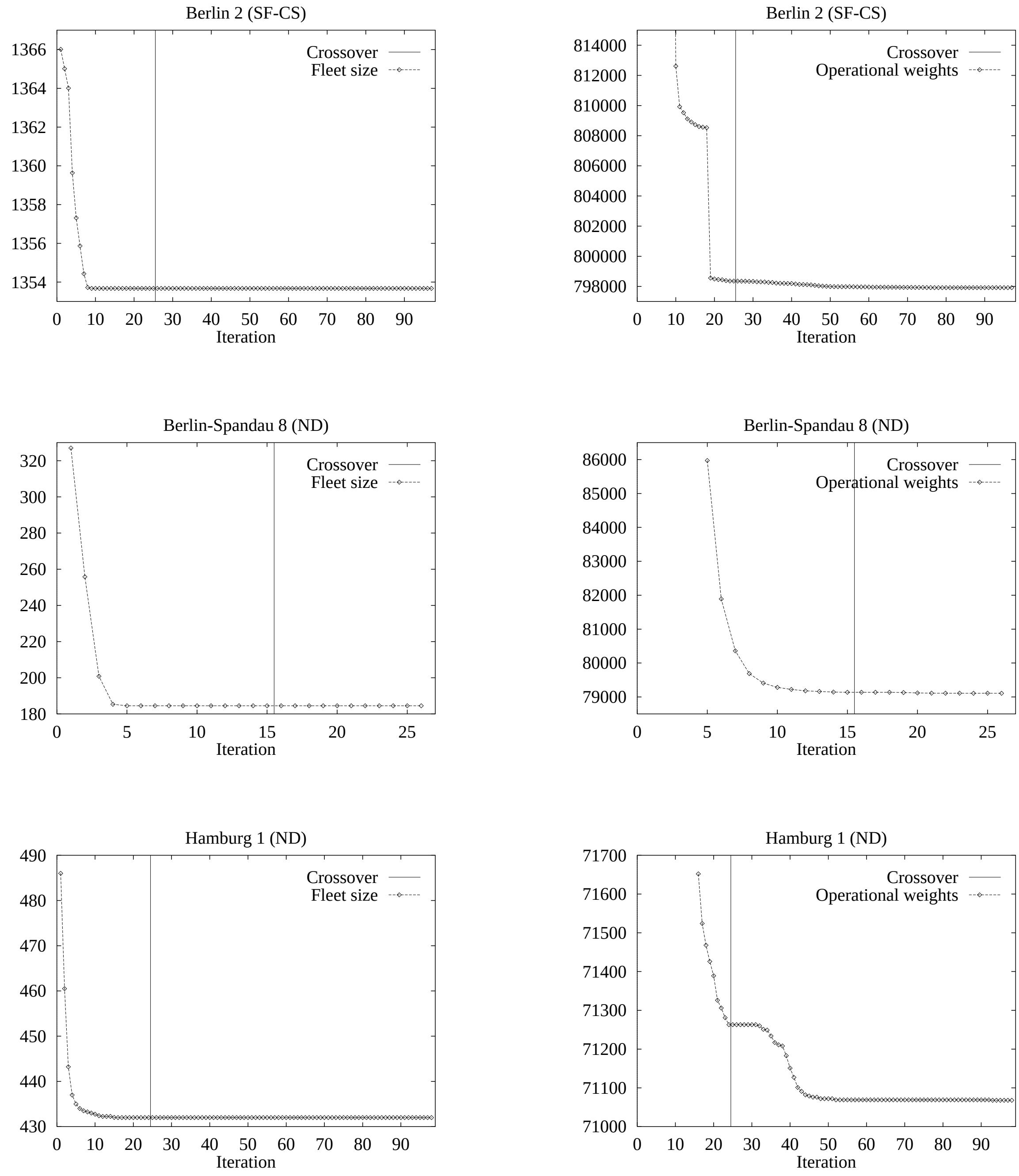


Figure 12.10: Typical development of the fleet size values (left pictures) and the operational weights (right pictures) in respect to the k^{th} RLP.

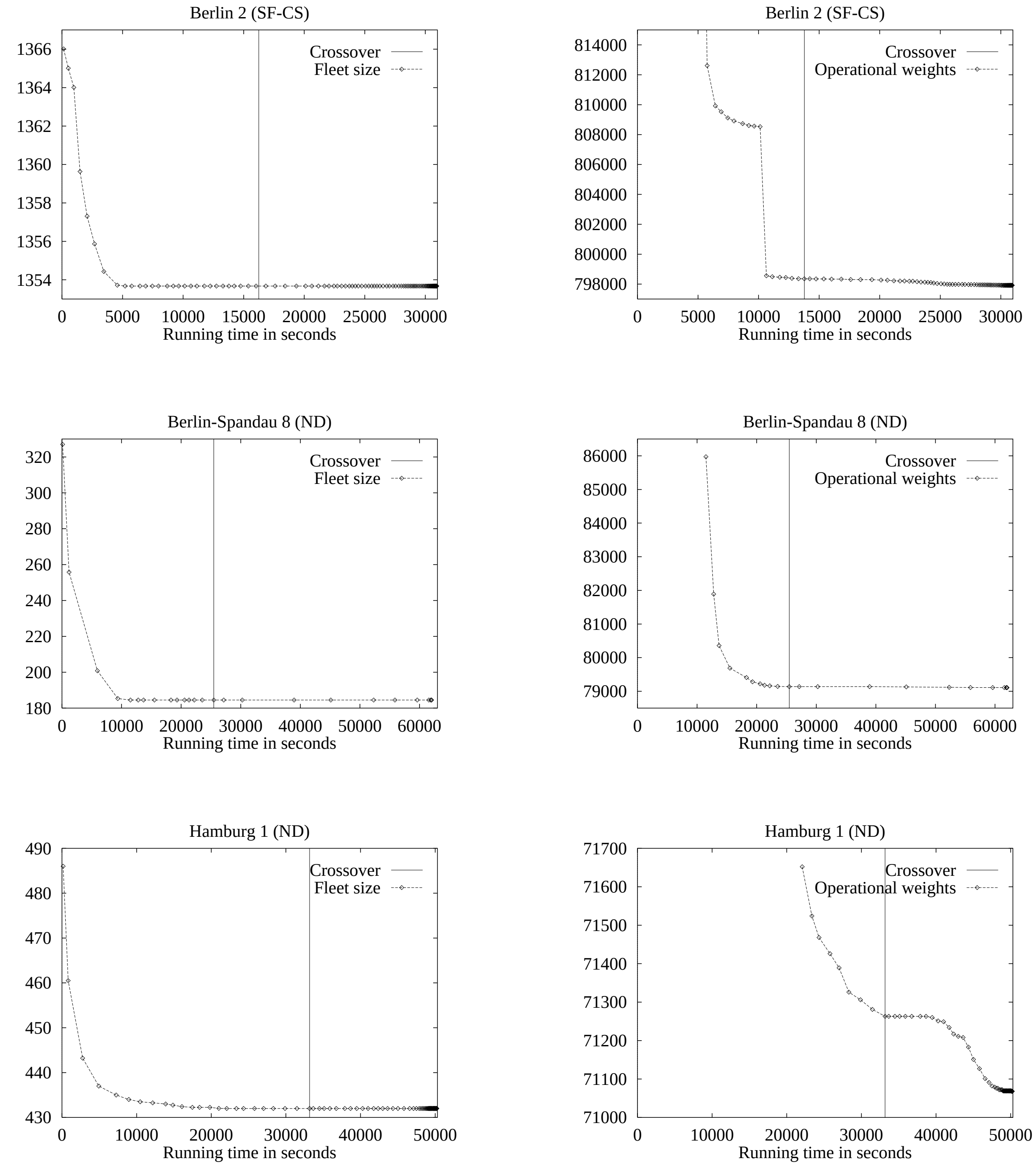


Figure 12.11: Typical development of the fleet size values (left picture) and the operational weights (right pictures) in respect to the run time.

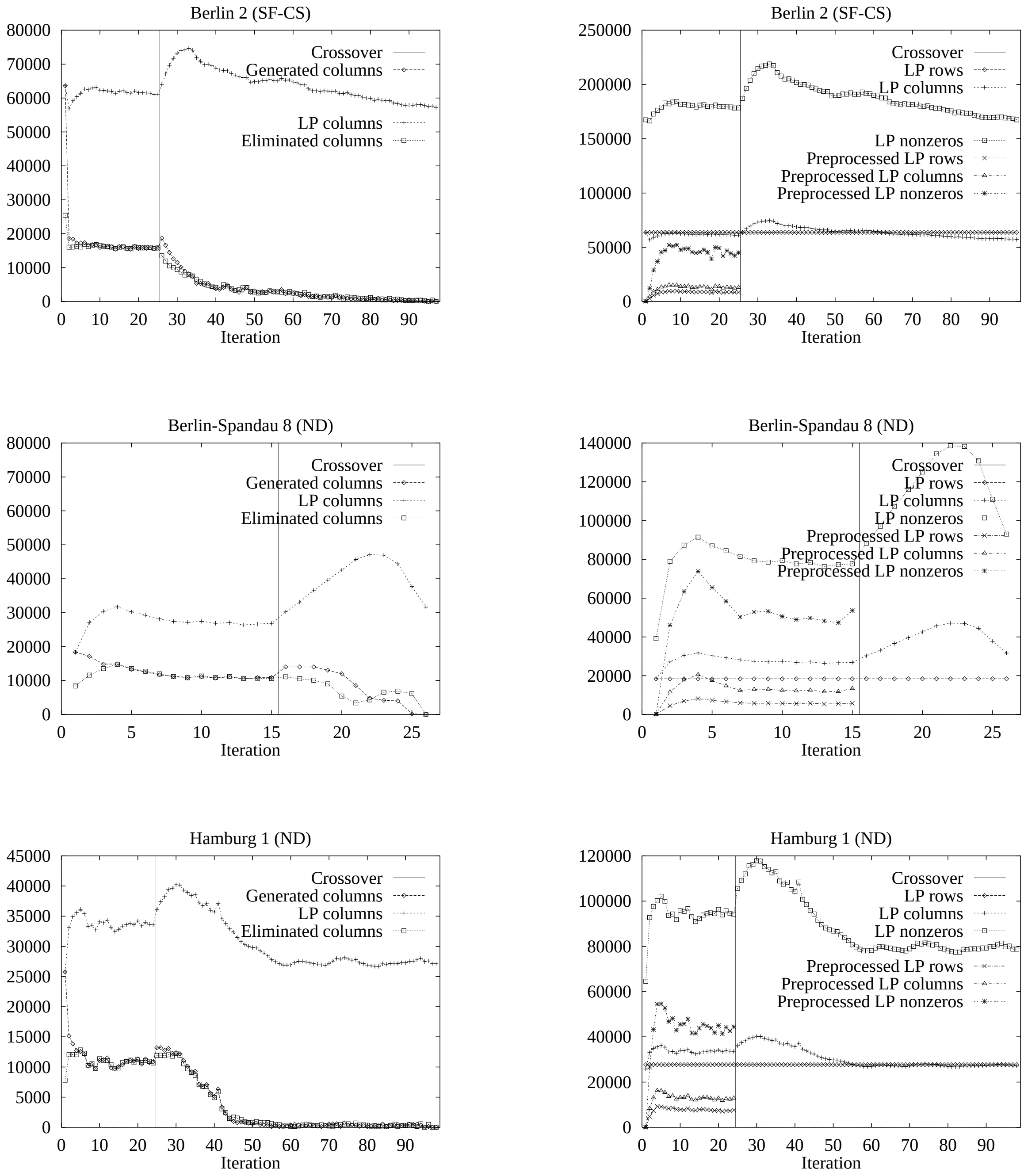


Figure 12.12: Typical number of generated, eliminated, and active LP columns (left pictures) and number of LP columns, rows, and nonzero elements (right pictures) per iteration.

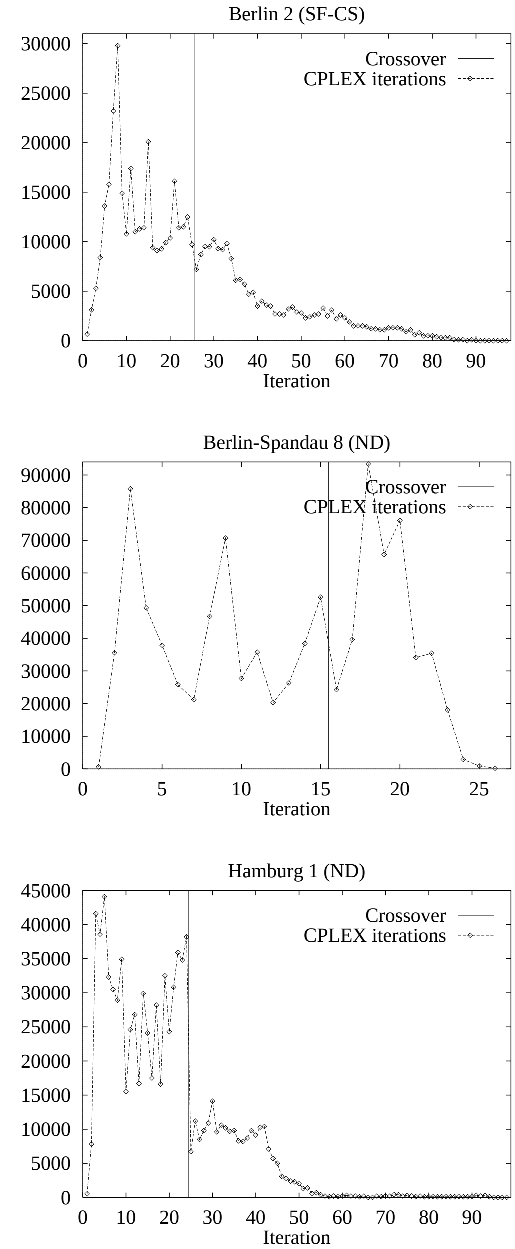


Figure 12.13: Typical number of CPLEX iterations for the k^{th} RLP.

12.7 Dantzig-Wolfe Decomposition

Actually, our main goal was to solve the IMP to optimality requiring first the optimal solution of MP. When we started our computational decomposition tests, it turned out soon that decomposition is an unsuitable method to solve the MDVSP. The major obstacle is that the MPs become too hard to be solved efficiently. Especially for problems with more than one thousand timetabled trips, the LU factorization in solving an RMP takes far too much time.

In what follows, we describe our decomposition test runs for MP in detail and discuss the reason why this kind of approach fails for the MDVSP. These tests are based on some smaller and not so hard problems from HHA. We consider its 19 smallest single-depot instances and all its multiple-depot problems except the large 12-depot instance. All the other HHA instances are too large to be solved by decomposition. For each of the considered problems, the optimal LP, the optimal MP, and the integer optimal values are the same.

Table 12.16 shows some statistics of our decomposition test set such as the number of depots ($|\mathcal{D}|$), the number of timetabled trips ($|\mathcal{T}|$), the number of unloaded trips ($|A^{\text{u-trip}}|$), the average depot group size $\varnothing G$, the number of possible vehicle schedules, and the number of possible blocks.

We give some general observations about our computational tests and report on specific results for the considered problems. For each test problem, we have used a time limit, denoted by TL, of three hours. This time limit is quite generous since the LP relaxation of the biggest decomposition test problem, Hamburg 2, can be solved to optimality within less than 400 seconds.

General Observations.

- For our test set, the pricing problems PP_d are efficiently solvable within a small part of the total run time, which can be neglected.
- The right-hand side of MP is equal to $\mathbb{1}$ leading to degenerate LPs. From version 4.0 on, CPLEX provides a new, less aggressive perturbation method that can handle degenerate LPs efficiently. And indeed, we have never recognized any numerical problems stemming from degeneracy since we have used the first beta version of CPLEX 4.0.
- In one of the first versions of our implementation, each cost coefficient c_w^d was just the sum of M and the operational costs of the corresponding vehicle schedule $w \in W_d$. The naive use of this two stage objective function with a M equal to 10^8 has caused numerical difficulties in proving global optimality: The optimality tolerance of CPLEX can only be set to a value between 10^{-9} and 10^{-4}. This means that the reduced costs must be at least greater than or equal to -10^{-4} for each $w \in \tilde{W}$. If the total fleet size needs 3 digits or more and if the optimality tolerance is at least

Decomposition test set	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert A^{\text{u-trip}}\rvert$		$\varnothing G$	$\lvert W\rvert$	#blocks
			User-def.	Total			
Hamburg 14	1	109	617	5,241	1.0	$2.0 \cdot 10^{13}$	$2.3 \cdot 10^{10}$
Hamburg 15	1	183	1,025	14,347	1.0	$2.0 \cdot 10^{16}$	$7.1 \cdot 10^{9}$
Hamburg 2 - Depot 4	1	211	2,651	16,854	1.0	$3.9 \cdot 10^{14}$	$7.5 \cdot 10^{10}$
Hamburg 7 - Depot 2	1	220	2,348	21,315	1.0	$5.5 \cdot 10^{16}$	$7.0 \cdot 10^{11}$
Hamburg 7 - Depot 1	1	232	3,922	12,510	1.0	$1.1 \cdot 10^{16}$	$4.7 \cdot 10^{11}$
Hamburg 2 - Depot 8	1	493	8,322	104,153	1.0	$1.6 \cdot 10^{18}$	$8.0 \cdot 10^{11}$
Hamburg 3 - Depot 1	1	521	19,003	65,650	1.0	$8.8 \cdot 10^{18}$	$2.4 \cdot 10^{13}$
Hamburg 2 - Depot 5	1	648	14,136	180,554	1.0	$3.9 \cdot 10^{18}$	$4.3 \cdot 10^{11}$
Hamburg 2 - Depot 7	1	695	16,526	186,863	1.0	$1.3 \cdot 10^{20}$	$6.3 \cdot 10^{13}$
Hamburg 1 - Depot 7	1	728	13,654	233,010	1.0	$2.3 \cdot 10^{23}$	$5.9 \cdot 10^{16}$
Hamburg 1 - Depot 12	1	892	25,100	348,230	1.0	$3.2 \cdot 10^{24}$	$1.8 \cdot 10^{17}$
Hamburg 5 - Depot 2	1	930	27,452	368,242	1.0	$5.3 \cdot 10^{25}$	$1.1 \cdot 10^{20}$
Hamburg 5 - Depot 1	1	986	56,760	211,153	1.0	$1.1 \cdot 10^{24}$	$1.6 \cdot 10^{18}$
Hamburg 1 - Depot 4	1	1,065	31,520	488,655	1.0	$1.4 \cdot 10^{25}$	$1.0 \cdot 10^{19}$
Hamburg 1 - Depot 11	1	1,296	77,201	459,468	1.0	$1.3 \cdot 10^{27}$	$2.3 \cdot 10^{20}$
Hamburg 6 - Depot 1	1	1,345	102,253	322,546	1.0	$2.4 \cdot 10^{22}$	$1.2 \cdot 10^{16}$
Hamburg 1 - Depot 10	1	1,588	79,955	404,016	1.0	$5.1 \cdot 10^{20}$	$1.4 \cdot 10^{14}$
Hamburg 6 - Depot 2	1	1,693	72,882	1,233,766	1.0	$5.0 \cdot 10^{28}$	$4.1 \cdot 10^{23}$
Hamburg 1 - Depot 3	1	1,716	116,142	850,828	1.0	$5.6 \cdot 10^{30}$	$1.3 \cdot 10^{25}$
Hamburg 4	2	238	2,000	23,000	1.04	$1.5 \cdot 10^{21}$	$2.4 \cdot 10^{15}$
Hamburg 7	2	341	6,000	34,000	1.32	$5.9 \cdot 10^{16}$	$1.2 \cdot 10^{12}$
Hamburg 3	2	791	30,000	200,000	1.32	$2.3 \cdot 10^{22}$	$3.9 \cdot 10^{17}$
Hamburg 5	2	1,461	85,000	580,000	1.31	$5.3 \cdot 10^{25}$	$1.1 \cdot 10^{20}$
Hamburg 6	2	2,283	176,000	1,600,000	1.33	$5.0 \cdot 10^{28}$	$4.2 \cdot 10^{23}$
Hamburg 2	9	1,834	99,000	1,000,000	2.02	$3.3 \cdot 10^{21}$	$5.3 \cdot 10^{15}$

Table 12.16: Decomposition test set taken from HHA.

4 decimal places, we have to compute accurately for at least 3+8+4 digits (the fleet size plus the operational costs plus the optimality tolerance), which is close to and can be smaller than the machine precision. For an "almost" optimal RMP (i. e., the value of PP (11.8) is smaller than, but close to -10^{-4}), cancellation of important digits can occur in computing the reduced costs. Sometimes, the reduced costs of a generated column turned out to be feasible within the next RMP, but have been indicated to be infeasible in PP.

To avoid such numerical troubles, we scale down the objective coefficients by 10^8 and set the optimality tolerance to 10^{-9}. Then, we have to compute accurately for 8 digits less than before. This simple trick makes the implementation more robust. Since then, we have never recognized any further cancellation of important digits in solving PP.

- Initializing the first RMP only with all columns corresponding with trippers leads to quite bad starting points: Within the time limit, none of the problems could be solved to optimality, and only the single-depot problems with up to 493 timetabled trips and the multiple-depot problem Hamburg 7 could be solved fleet minimally with cost gaps ranging from 14 % up to 138 %. For all the other problems, we could not generate an (MP) solution yielding the minimum number of vehicles. Moreover, the basis solutions of the last RMPs are completely fractional and there are almost no variables with a value greater than or equal to 0.5, but most nonzero values are smaller than 0.2. This may have a negative influence on the performance of a branch-and-price algorithm.

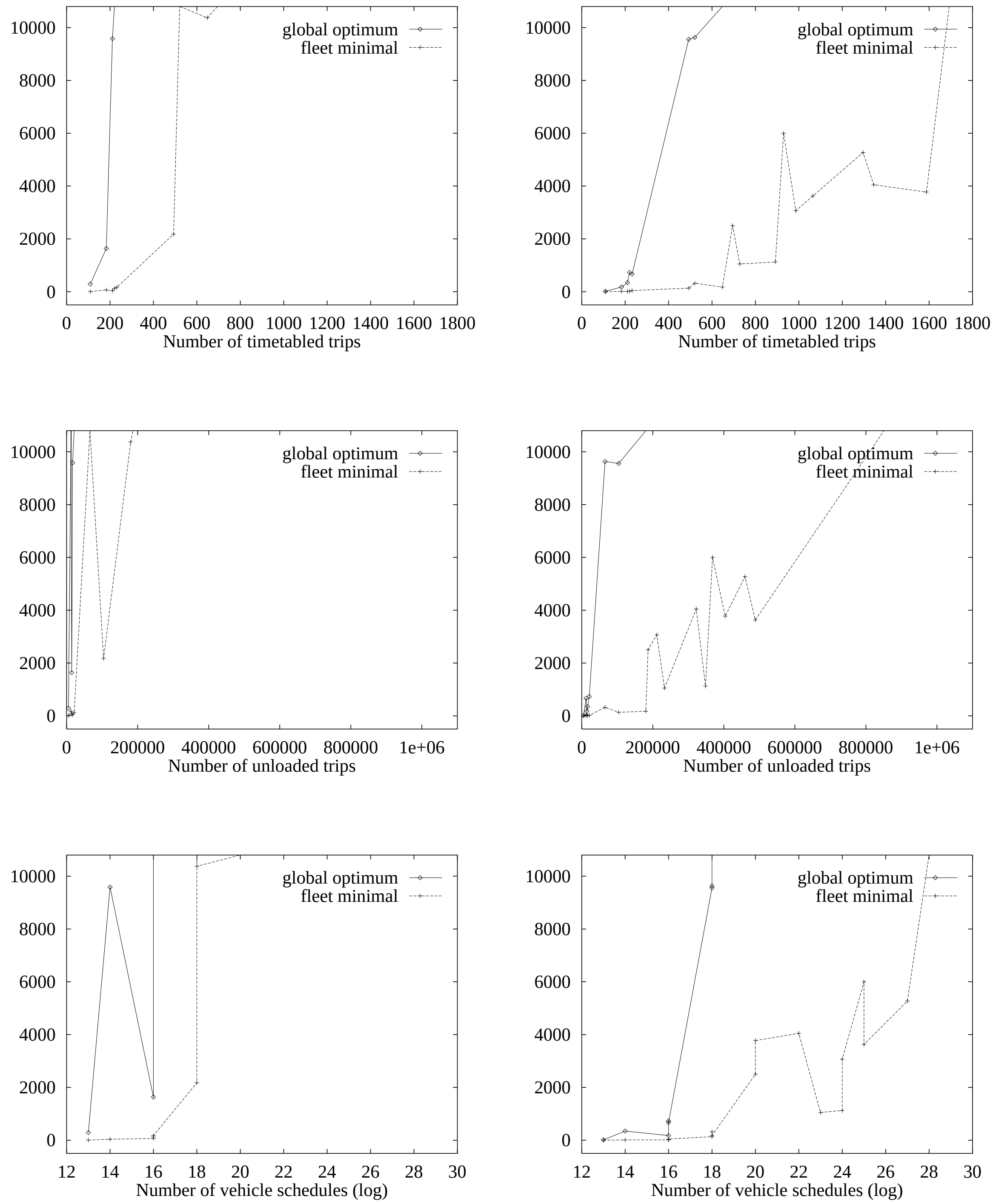
  Instead of starting with all trippers, we use the columns corresponding to the ND solution or, to see some effects, with an integer optimal solution. Note that ND already generates an optimal solution for single-depot instances.

- The average number of nonzero elements per column of the occurring MPs is approximated by the average number ot timetabled trips of the generated vehicle schedules. For our decomposition test set, this number is 17.3 with a standard deviation of 4.8; the lowest average column length of our problems is 8.4 and the largest is 35.2. These MPs are quite dense set partitioning problems. It is known that the LU factorization becomes the bottleneck in solving such problems with more than thousand timetabled trips. A similar observation has been made by Borndörfer [1998], he solves set partitioning problems arising from large-scale vehicle routing from a dial-a-ride system for handicapped people in Berlin.

- Fixing the number of timetabled trips, the investigations of Ribeiro and Soumis [1994] indicate that the run time is linear in the number of depots. We have too few test problems to make such a statement, but this result of Ribeiro and Soumis sounds reasonable.

The Single-Depot Problems.

To get a feeling about the size that a problem can have to be successfully solved with a decomposition approach, we have first investigated our single-depot instances. The results are displayed in Fig. 12.14 showing on the axis of ordinates the run times with respect to the number of timetabled trips, unloaded trips, and vehicle schedules, respectively. The left pictures show the results that we have obtained without Lagrangean pricing: We started with an optimal integral solution and always generated only the column delivered by PP. Stopping whenever the time limit was reached, it turned out that this strategy solves only the master problem of rather tiny problems with up to 200 timetabled trips (which are about 20 thousand dead-head trips and 10^{16} vehicle schedules) to optimality. If we just want to find an RMP including a minimal fleet solution (proved via sufficiently small negative reduced costs), it is possible to solve problems with up to 700 timetabled trips (200 thousand dead-head trips and 10^{20} vehicle schedules). The right pictures show the results that we have obtained using in addition Lagrangean pricing, which significantly



WITHOUT Lagrangean pricing. WITH Lagrangean pricing.

Figure 12.14: Decomposition run times for the single-depot instances

accelerates the solution convergence. With this technique, we are able to solve the master problem of problems with up to 700 timetabled trips optimally and with up to 1700 timetabled trips fleet minimally. A negative side effect of Lagrangean pricing is that there are more columns generated between two consecutive RMPs making each single RMP much harder to solve. It is not clear to us whether there is some break even point where this acceleration of iteration convergence is canceled out by larger LP run times.

The Multiple-Depot Problems.

We have shown that starting from scratch (with all columns corresponding with trippers) does not work at all. So, we have used the solutions obtained by ND as starting points. The decomposition run times with and without Lagrangean pricing of these tests are compared with the LP times. All these times are displayed in Tab. 12.17 showing that the decomposition without Lagrangean pricing solves only the two small problems Hamburg 4 and 7 fleet minimal, but not to global optimality. Using Lagrangean pricing accelerates this solution times, but does not lead to further significant improvements such as solving more problems at least fleet minimal. Compared to our direct LP method, the decomposition implementation is completely inferior.

Test Set[a]	Decomposition				LP	
	Default[b]		Lagr. pricing[c]			
	fleet[d]	global[e]	fleet[d]	global[e]	fleet[d]	global[e]
Hamburg 4	84	TL	25	TL	2	3
Hamburg 7	4,210	TL	1,663	TL	8	9
Hamburg 3	TL[f]	—	TL[f]	—	18	31
Hamburg 5	TL[f]	—	TL	—	135	155
Hamburg 6	TL	—	TL	—	80	84
Hamburg 2	TL[f]	—	TL[f]	—	679	685

[a]ND generates an optimal solution for Hamburg 4 and 6.
[b]Without Lagrangean pricing.
[c]With Lagrangean pricing.
[d]Time to prove that the starting solution is fleet minimal.
[e]Time to prove global optimality of the last RMP and RLP.
[f]No objective improvement within the time limit.

Table 12.17: Run times starting with a solution generated with the ND heuristic.

To see whether the decomposition method can at least prove the optimality of an optimal solution, we started the decomposition and the LP method with an integer optimal solution. Table 12.18 gives the run times, the number of solved RMPs and restricted LPs, the number of generated columns, and the number of CPLEX iterations being performed for these test runs.

First of all, the LP method is able to prove optimality for each of these problems within

Test Set	Decomposition				LP	
	Default[a]		Lagr. pricing[b]			
	A[c]	B[d]	A[c]	B[d]	A[c]	B[d]
Run times						
Hamburg 4	84	TL	27	TL	2	3
Hamburg 7	1,031	TL	204	TL	8	9
Hamburg 3	TL	—	656	TL	22	17
Hamburg 5	TL	—	TL	—	72	78
Hamburg 6	TL	—	TL	—	128	179
Hamburg 2	TL	—	TL	—	388	397
Number of RMPs and RLPs						
Hamburg 4	329	5,658	76	3,991	3	4
Hamburg 7	435	1,882	87	1,439	19	23
Hamburg 3	667	—	50	301	3	13
Hamburg 5	707	—	61	—	19	23
Hamburg 6	668	—	45	—	10	18
Hamburg 2	353	—	34	—	14	18
Generated columns / 1,000						
Hamburg 4	0	6	1	5	2	2
Hamburg 7	1	4	1	4	14	15
Hamburg 3	1	—	2	2	6	16
Hamburg 5	1	—	3	—	57	62
Hamburg 6	1	—	3	—	56	73
Hamburg 2	2	—	4	—	55	55
CPLEX iterations / 1,000						
Hamburg 4	5	487	3	332	0	0
Hamburg 7	41	293	11	210	1	1
Hamburg 3	57	—	7	54	0	1
Hamburg 5	3	—	26	—	2	3
Hamburg 6	1	—	12	—	4	7
Hamburg 2	38	—	19	—	19	19

[a]Without Lagrangean pricing.
[b]With Lagrangean pricing.
[c]Time, iterations, etc. to prove that the starting solution is fleet minimal.
[d]Time, iterations, etc. to prove global optimality of last RMP and RLP.

Table 12.18: Starting with an integer optimal solution.

less than 400 seconds run time. Within a time limit of three hours, our decomposition implementation was unable to prove optimality of any of these problems, only the fleet minimality could be confirmed for the problems with up to 791 timetabled trips.

The larger the number of timetabled trips of an instance is, the more time is spent within the LU factorization. The worst case example is Hamburg 2: Applying in addition the Lagrangean pricing, only 34 RMPs could be solved within the time limit, which are far too less to have a chance to solve such large problems within reasonable run time.

Comparison to Ribeiro and Soumis

Already in 1991, Ribeiro and Soumis [1994] have reported about their computational results using a Dantzig-Wolfe decomposition for the MDVSP. They solved randomly generated capacitated MDVSPs with up to 300 timetabled trips and 6 depots to optimality using a branch-and-bound algorithm. Compared with our computational results, we wonder how they can solve even the integer formulation of their problems to optimality while we are unable to just solve the MP relaxations of our uncapacitated problems with a similar number of timetabled trips using a considerably faster workstation and a more sophisticated version of CPLEX. From our point of view, the reasons are as follows:

- They do not mention the number of dead-head trips. If this number is small, the number of possible vehicle schedules is also relatively small. Each problem instance of Ribeiro and Soumis includes 60 % long timetabled trips with a duration uniformly distributed between three and five hours. The other 40 % are short timetabled trips with a duration uniformly distributed between 5 and 40 minutes. On the average, each timetabled trip has therefore a duration of 2 hours and 33 minutes. 70 % of the short timetabled trips are defined between 8.00 a. m. and 5.00 p. m. The duration of the timetabled trips given by our test set is, on the average, about 30 minutes, the morning peak begins earlier and the afternoon peak ends later than the peaks of their problems.
- The used depot capacities of Ribeiro and Soumis seem to be quite generous: For each test instance, they provide one vehicle for at most two up to three timetabled trips. In a city like Berlin, about 10 thousand buses would be necessary to run about 25 thousand timetabled trips that have to be daily serviced in Berlin. But BVG maintains less than 2,000 buses. Even if we assume that only one half of the available fleet sizes are used for their problems – otherwise, a consideration of depot capacities becomes pointless – the resulting vehicle schedules would on the average contain five timetabled trips. In Berlin and Hamburg, this number is about 15 and 20.
- The only information provided by them is how many columns are generated until optimality can be proved. The largest number of generated columns was 4,585 for an instance with 5 depots and 300 timetabled trips. They give neither information about the average length of the generated vehicle schedules nor about the vehicle

demand of the optimal solutions. For our test set, we have generated, on the average, vehicle schedules with 17.3 timetabled trips (standard deviation was 4.8). The problem with the smallest value was 8.4 timetabled trips and the largest value was 35.2 timetabled trips.

- For the capacitated MDVSP, it is $\mathcal{NP}$-hard to find a feasible solution. Ribeiro and Soumis, however, do not tell how they initialize their method.

Based on the above considerations, we are unable to estimate the average length of the vehicle schedules of their problems, which may be an indicator for the "hardness" of their master problems. It seems to us that the combinatoric of our test set is harder than the combinatoric of their test set. All these open questions give us rise to the suspicion that the artificially generated problems of Ribeiro and Soumis are of a different quality and, from a computational point of view, are far easier than our real-world problems.

Conclusions

This thesis is devoted to the Multiple-Depot Vehicle Scheduling Problem (MDVSP), a problem arising, for instance, in each city or region with a public transportation system. Problems of this type are of very large scale. In the previous chapters, we have presented a variety of methods for its solution. All approaches are based on techniques from combinatorial optimization. A well-chosen combination of these methods turned out to be able to solve (almost) all problems of practical interest in acceptable running times. The success of the implementations, of course, benefited from the (in the recent years drastically increased) computing power of modern workstations and sophisticated commercial optimization software (such as the LP solver CPLEX). We summarize some of our findings:

Single-depot instances can be solved quickly. With state of the art codes for minimum-cost flow problems, such as our implementation MCF, and the use of column generation techniques, practical problem instances of (almost) arbitrary size can be solved to optimality in a few minutes. For instance, we have solved a 70 million variable problem in about 15 minutes, see Tab. 12.8.

Upper bounds that have been generated with the schedule – cluster – reschedule heuristic (SCR), which employs our single-depot solver, are of high quality. Compared with the optimal integer solutions, SCR provides solutions with a fleet size gap and operational weight gap of less than 1.25 % and 5.2 %, respectively, see Tab. 12.11. This shows that MDVSPs can be heuristically solved in reasonable running time, see Tab. 12.12.

Lagrangean relaxations allow to compute tight lower bounds even for large multiple-depot instances, see Tab. 12.13. In particular, neglecting the flow conservation constraints, lower bounds can be computed quite fast, see Tab. 12.14. Lagrangean relaxations can be used to quickly simulate fleet and cost effects of different parameter settings.

Branch-and-cut is capable of solving even very large multiple-depot instances to optimality, see Tab. 12.9.

Lagrangean pricing is a good idea to solve the large degenerate LPs that come up in solving multiple-depot instances with branch-and-cut. Our initial code used the well known standard reduced cost pricing techniques. However, this did not work at all because of stalling, see Chap. 7. To cure stalling, we introduced (what we call) Lagrangean pricing. We propose it as one of the basic ingredients of an effective method to solve multiple-depot vehicle scheduling problems. Similar positive results have been observed by Fischetti and Toth [1996] and Fischetti and Vigo [1996] also dealing with large degenerate LPs. We believe that variable pricing based on Lagrangean relaxation is a useful tool that can help to solve many combinatorial optimization problems.

Dantzig-Wolfe decomposition seems not to be a useful approach for large MDVSPs. We have experimented with various possibilities to employ Dantzig-Wolfe decomposition for the solution of MDVSPs. At first glance, this seems to be a quite natural and obvious approach. The computational results, however, are very disappointing, see Tabs. 12.17 and 12.18.

Computational breakthrough: to our knowledge, at present no other implementation is able to solve MDVSPs with more than 1,000 timetabled trips to optimality. Our code has successfully produced optimal solutions of various real-world problem instances with up to 25 thousand timetabled trips, see Tab. 12.9. The integer multicommodity flow problems arising this way are orders of magnitude larger than what other codes are able to handle. The largest real instance we encountered gave rise to an integral multicommodity flow problem with about 125 thousand equations and 70 million integer variables. We could not produce an optimal solution, but found a solution with a fleet size gap of less than 0.5 %.

Possible savings indicated by our test runs are immense. Compared with a manual planning process, the SCR heuristic indicates savings of about 15 % – 20 % of the vehicles and about 10 % – 15 % of the operational costs. Compared with an assignment heuristic, our branch-and-cut method indicates savings of several vehicles and about 10 % cost reduction. However, the final evaluations of the SCR generated solutions have not been finished by BVG, HHA, and VHH yet. It still has to be checked whether our vehicle schedules provide a useful input for duty scheduling, the next step in the hierarchical planning process. It is therefore not clear how much of these indicated savings can be obtained in practice. Nonetheless, our methods can solve large problems optimally. The Berliner Verkehrsbetriebe, for instance, expect to save about DM 100 million per year with our SCR heuristic, see Schmidt [1997].

There is a high demand within industry for efficient methods for the MDVSP. Parts of our system have been purchased by BVG for their planning system BERTA, by IVU for MICROBUS II, and by the research department of the SIEMENS AG in Munich.

There are further challenges in the field of vehicle scheduling in public transit. The main bottleneck of our branch-and-cut method was the solution of the LP relaxations. The solution process of the branch-and-cut software could be accelerated significantly by employing faster (e. g., parallelized) LP solvers.

A second task is to investigate whether further savings can be obtained by flexible departure times for the timetabled trips. Tests by Daduna and Völker [1997] show possible savings of several vehicles using the heuristic assignment approach of the HOT system.

In the long term, we aim at solving vehicle and duty scheduling by an integrated approach. Currently, the existing mathematical knowhow and the available computing power are insufficient to solve vehicle and duty scheduling in one step. Since most of the available planning systems do not include (sophisticated) mathematical methods for duty scheduling, transportation companies may well benefit from duty optimization software. Together with IVU and HanseCom, we have started a new project dealing with the optimization of duty scheduling.

List of Figures

List of Tables

Bibliography

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1989). *Network Flows.* In Nemhauser, Rinnooy Kan, and Todd [1989], chapter IV, pages 211–369.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Assad, A., Ball, M., Bodin, L., and Golden, B. (1983). Routing and scheduling of vehicles and crews. *Computers & Operations Research*, 10(2):63–211.

Balas, E. (1975). Facets of the knapsack polytope. *Mathematical Programming*, 8:146 – 164.

Ball, M. O., Magnanti, T. L., Monma, C. L., and Nemhauser, G. L., editors (1995b). *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science.* Elsevier Science B.V., Amsterdam.

Ball, M. O., Magnanti, T. L., Monma, C. L., and Nemhauser, G. L., editors (1995a). *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science.* Elsevier Science B.V., Amsterdam.

Barnhart, C., Hane, C. A., and Vance, P. H. (1996). Integer multicommodity flow problems. In Cunningham, McCormick, and Queyranne [1996], pages 58–71.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P. H. (1994). Branch-and-price: Column generation for solving huge integer programs. In Birge, J. R. and Murty, K. G., editors, *Mathematical Programming: State of the Art 1994*, pages 186–207. The University of Michigan.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., and Vance, P. H. (1997). Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 45(2):188–200.

Bazaraa, M. S., Jarvis, J. J., and Sherali, H. D. (1990). *Linear programming and network flows.* John Wiley & Sons, Inc., 2nd edition.

Becker, J., Roß, J., and Schemczyk, O. (1996). BERTA – EDV-gestützte Betriebseinsatzplanung bei den Berliner Verkehrsbetrieben, Sachstand und Ausblick. *Verkehr und Technik (V+T).* Heft 2+3. In German.

Beguelin, A., Dongarra, J., Geist, A., Jiang, W., Manchek, R., and Sunderam, V. (1994). *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing.* The MIT Press, Cambride, Massachusetts, London, England. Information available via WWW at www.epm.ornl.gov/pvm/pvm_home.html.

Berge, C. (1973). *Graphs and Hypergraphs.* North-Holland, Amsterdam.

Bertossi, A. A., Carraresi, P., and Gallo, G. (1987). On some matching problems arising in vehicle scheduling models. *Networks*, 17:271–181.

Bertsekas, D. P. and Tseng, P. (1994). RELAX-IV: A faster version of the RELAX code for solving minimum cost flow problems. Technical report. Available via WWW at ftp://lids.mit.edu/pub/bertsekas/RELAX/.

Bianco, L., Mingozzi, A., and Ricciardelli, S. (1994). A set partitioning approach to the multiple-depot vehicle scheduling problem. *Optimization Methods and Software*, 3:163–194.

Bixby, R. E. (1992). Implementing the simplex method: The initial basis. *ORSA Journal on Computing*, 4(3):267–284.

Bixby, R. E. (1994). Progress in linear programming. *ORSA Journal on Computing*, 6(1):15–22.

Blais, J.-Y. and Rousseau, J.-M. (1988). Overview of HASTUS current and future versions. In Daduna and Wren [1988], pages 175–187.

Bodin, L. and Golden, B. (1981). Classification in vehicle routing and scheduling. *Networks*, 11:97–108.

Bodin, L., Kydes, A., and Rosenfield, D. (1978). UCOST: a micro approach to a transportation planning problem. *Urban Analysis*, 5:47–69.

Böhmig, T. and Wolter, M. (1997). Kostenträgerrechnung für ÖPNV-Betriebe. *Der Nahverkehr*, 1–2/97:14–19. In German.

Bokinge, U. and Hasselström, D. (1980). Improved vehicle scheduling in public transport through systematic changes in the time-table. *European Journal of Operational Research*, 5:388–395.

Bollobás, B. (1978). *Extremal Graph Theory.* Academic Press, London.

Bondy, J. A. and Murty, U. S. R. (1976). *Graph Theory with Applications.* American Elsevier, New York, and Macmillan, London.

Borger, J. M., Kang, T. S., and Klein, P. N. (1993). Approximating concurrent flow with unit demands and capacities: an implementation. In Johnson and McGeoch [1993].

Borndörfer, R. (1998). *Packing, Partitioning, and Covering of Sets.* PhD thesis, Technical University of Berlin.

Borndörfer, R., Grötschel, M., and Löbel, A. (1995). Alcuin's transportation problems and integer programming. Preprint SC 95-27, Konrad-Zuse-Zentrum Berlin. Available via WWW at www.zib.de. To appear in Butzer, P. L., Jongen, H. T., and Oberschelp, W., editors, *Charlemagne and his Heritage: 1200 Years of Civilization and Science in Europe, Volume II: The Mathematical Arts*, Brepols Publishers.

Branco, I., Costa, A., and Paixão, J. M. P. (1995). Vehicle scheduling problem with multiple type of vehicles and a single depot. In Daduna, Branco, and Paixão [1995], pages 115–129.

Branco, I. M. and Paixão, J. P. (1987). A quasi-assignment algorithm for bus scheduling. *Networks*, 17:249–269.

Branco, I. M. and Paixão, J. P. (1988). Bus scheduling with a fixed number of vehicles. In Daduna and Wren [1988], pages 28–40.

Bussieck, M., Winter, T., and Zimmermann, U. T. (1997). Discrete optimization in public rail transport. In Liebling, T. M. and de Werra, D., editors, *Mathematical Programming: A Publication of the Mathematical Programming Society*, pages 415–444. Elsevier Science B.V.

BVG. *Berliner Verkehrsbetriebe.* Information available via WWW at www.bvg.de.

Carpaneto, G., Dell'Amico, M., Fischetti, M., and Toth, P. (1989). A branch and bound algorithm for the multiple depot vehicle scheduling problem. *Networks*, 19:531–548.

Carraresi, P. and Gallo, G. (1984). Network models for vehicle and crew scheduling. *European Journal of Operations Research*, 16:139–151.

Ceder, A. and Stern, H. I. (1981). Deficit function bus scheduling with deadheading trip insertions for fleet size reduction. *Transportation Science*, 15(4):338–363.

Chamberlain, M. and Wren, A. (1988). The development of Micro-BUSMAN: Scheduling on micro-computers. In Daduna and Wren [1988], pages 160–174.

Chamberlain, M. and Wren, A. (1992). Developments and recent experience with the BUSMAN and BUSMAN II systems. In Desrochers and Rousseau [1992], pages 1–15.

Christof, T. (1994). PORTA – A Polyhedron Representation Transformation Algorithm, version 1.2.1. Written by T. Christof, revised by A. Löbel and M. Stoer. PORTA source code available via WWW at www.iwr.uni-heidelberg.de/iwr/comopt or www.zib.de.

Chvátal, V. (1980). *Linear programming.* W. H. Freeman and Company, New York.

CPLEX (1997). *Using the CPLEX Callable Library.* ILOG CPLEX Division, 889 Alder Avenue, Suite 200, Incline Village, NV 89451, USA. Information about CPLEX available via WWW at www.cplex.com.

Crowder, H. (1976). Computational improvements for subgradient optimization. In *Symposia Mathematica*, volume 19. Istituto Nazionale di Alta Matematica, Academic Press London and New York.

Cunningham, W. H. (1976). A network simplex method. *Math. Programming*, 11:105–116.

Cunningham, W. H., McCormick, S. T., and Queyranne, M., editors (1996). *Integer Programming and Combinatorial Optimization*, 5th International IPCO Conference, Vancouver, British Columbia, Canada.

Daduna, J. R., Branco, I., and Paixão, J. M. P., editors (1995). *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems. Springer Verlag.

Daduna, J. R. and Mojsilovic, M. (1988). Computer-aided vehicle and duty scheduling using the HOT programme system. In Daduna and Wren [1988], pages 133–146.

Daduna, J. R., Mojsilovic, M., and Schütze, P. (1993). Practical experiences using an interactive optimization procedure for vehicle scheduling. In Du and Pardalos [1993], pages 37–52.

Daduna, J. R. and Paixão, J. M. P. (1995). Vehicle scheduling for public mass transit – an overview. In Daduna, Branco, and Paixão [1995].

Daduna, J. R. and Völker, M. (1997). Vehicle scheduling with not exactly specified departure times. In *7th International Workshop on Computer-Aided Scheduling of Public Transport*, pages 2–12. Center for Transportation Studies, MIT, Cambridge, USA. A German version entitled *Fahrzeugumlaufbildung im ÖPNV mit unscharfen Abfahrtzeiten* has been publised in *Der Nahverkehr*, 11/97:39–43.

Daduna, J. R. and Wren, A., editors (1988). *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems. Springer Verlag.

Dantzig, G. B. (1963). *Linear Programming and Extensions.* Princeton University Press, Princeton.

Dantzig, G. B. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, 8:101–111.

Dell'Amico, M., Fischetti, M., and Toth, P. (1993). Heuristic algorithms for the multiple depot vehicle scheduling problem. *Management Science*, 39(1):115–125.

Desrochers, M., Desrosiers, J., and Soumis, F. (1985). Optimal urban bus routing with scheduling flexibilities. In Thoft, P., editor, *Lecture Notes in Control and Information Sciences*, pages 155–165.

Desrochers, M. and Rousseau, J.-M., editors (1992). *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems. Springer Verlag.

Desrosiers, J., Dumas, Y., Solomon, M. M., and Soumis, F. (1995). *Time Constrained Routing and Scheduling.* In Ball, Magnanti, Monma, and Nemhauser [1995a], chapter 2, pages 35–139.

Du, D.-Z. and Pardalos, P. M., editors (1993). *Network Optimization Problems: Algorithms, Applications and Complexity*, volume 2 of *Series on Applied Mathematics*, Singapore, New York, London. World Scientific Publishing Co. Pte. Ltd.

Fischetti, M. and Toth, P. (1988). A new dominance procedure for combinatorial optimization problems. *Operations Research Letters*, 7(4):181–187.

Fischetti, M. and Toth, P. (1996). A polyhedral approach to the asymmetric traveling salesman problem. Technical report, University of Bologna. To appear in *Management Science.*

Fischetti, M. and Vigo, D. (1996). A branch-and-cut algorithm for the resource-constrained arborescence problem. *Networks*, 29:55–67.

Forbes, M. A., Holt, J. N., and Watts, A. M. (1994). An exact algorithm for multiple depot bus scheduling. *European Journal of Operational Research*, 72(1):115–124.

Freling, R. and Paixão, J. M. P. (1995). Vehicle scheduling with time constraint. In Daduna, Branco, and Paixão [1995].

Fuchs, C. (1992). Optimization in bus scheduling – driver constraints, deadheading estimation and size advantage. In Desrochers and Rousseau [1992].

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York.

Gavish, B., Schweitzer, P., and Shlifer, E. (1978). Assigning buses to schedules in a metropolitan area. *Computers and Operations Research*, 5:129–138.

Gavish, B. and Shlifer, E. (1978). An approach for solving a class of transportation scheduling problems. *European Journal of Operational Research*, 3:122–134.

Goldberg, A. V. (1992). An efficient implementation of a scaling minimum-cost flow algorithm. Report No. STAN-CS-92-1439, Department of Computer Science, Standford University, Stanford, California 94305. Available via WWW at www.neci.nj.nec.com/homepages/avg/soft/soft.html.

Goldfarb, D. and Reid, J. K. (1977). A practicable steepest-edge simplex algorithm. *Mathematical Programming*, 12:361–371.

Grötschel, M., Löbel, A., and Völker, M. (1997). Optimierung des Fahrzeugumlaufs im Öffentlichen Nahverkehr. In Hoffmann, K. H., Jäger, W., Lohmann, T., and Schunck, H., editors, *Mathematik – Schlüsseltechnologie für die Zukunft*, pages 609–624. Springer Verlag. In German. Available as ZIB preprint SC 96-7 via WWW at www.zib.de.

Grötschel, M., Lovász, L., and Schrijver, A. (1988). *Geometric algorithms and combinatorial optimization.* Springer Verlag, Berlin.

Hamer, N. and Séguin, L. (1992). The HASTUS system: New algorithms and modules for the 90s. In Desrochers and Rousseau [1992], pages 17–29.

Hammer, P. L., Johnson, E. L., and Peled, U. N. (1975). Facets of regular 0-1 polytopes. *Mathematical Programming*, pages 179 – 206.

HanseCom. *HanseCom GmbH.* Information available via WWW at www.hansecom.com.

Harris, P. M. J. (1973). Pivot selection methods of the Devex LP code. *Mathematical Programming*, 15:1–28.

Hartley, T. (1981). A glossary of terms in bus and crew scheduling. In Wren [1981], pages 353–359.

Helgason, R. V. and Kennington, J. L. (1995). *Primal Simplex Algorithms for Minimum Cost Network Flows.* In Ball, Magnanti, Monma, and Nemhauser [1995b], chapter 2, pages 85–133.

HHA. *Hamburger Hochbahn AG.* Information available via WWW at www.hochbahn.com.

Hiriart-Urruty, J.-B. and Lemaréchal, C. (1993). *Convex Analysis and Minimization Algorithms I: Fundamentals.* Springer Verlag.

Hoffstadt, J. (1981). Computerized vehicle and driver scheduling for the Hamburger Hochbahn Aktiengesellschaft. In Wren [1981], pages 35–52.

IVU. *IVU Gesellschaft für Informatik, Verkehrs- und Umweltplanung mbH.* Information available via WWW at www.ivu-berlin.de.

Johnson, D. S. and McGeoch, C. C., editors (1993). *Network Flows and Matching*, volume 12 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society.

Klein, P., Plotkin, S., Stein, C., and Tardos, É. (1994). Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM J. Comput.*, 23(3):446–487.

Kokott, A. (1996). Lagrange-Relaxierungen für das Fahrzeugumlaufplanungsproblem. Master's thesis, Technische Universität Berlin. In German.

Kokott, A. and Löbel, A. (1996). Lagrangean relaxations and subgradient methods for multiple-depot vehicle scheduling problems. Preprint SC 96-22, Konrad-Zuse-Zentrum für Informationstechnik Berlin. Available via WWW at www.zib.de.

Kretschmann, C. and Lawerentz, R. (1997). Produktmanagement im ÖPNV. *Der Nahverkehr*, 1–2/97:8–13. In German.

Lamatsch, A. (1988). *Wagenumlaufplanung bei begrenzten Betriebshofkapazitäten*. PhD thesis, Universität Fridericiana zu Karlsruhe (TH). In German.

Lamatsch, A. (1992). An approach to vehicle scheduling with depot capacity constraints. In Desrochers and Rousseau [1992].

Larsen, A. and Madsen, O. B. G. (1997). Solving the multiple vehicle scheduling problem in a major scandinavian city. Technical Report IMM-REP-1997-10, IMM Department of Mathematical Modelling, Technical University of Denmark.

Lawler, E. L. (1976). *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York.

Leighton, T., Makedon, F., Plotkin, S., Stein, C., Tardos, É., and Tragoudas, S. (1991). Fast approximation algorithms for multicommodity flow problems. In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*.

Leong, T., Shor, P., and Stein, C. (1993). Implementation of a combinatorial multicommodity flow algorithm. In Johnson and McGeoch [1993].

Löbel, A. (1996a). Solving large-scale real-world minimum-cost flow problems by a network simplex method. Preprint SC 96-7, Konrad-Zuse-Zentrum für Informationstechnik Berlin. Available via WWW at www.zib.de.

Löbel, A. (1997a). Experiments with a Dantzig-Wolfe decomposition for multiple-depot vehicle scheduling problems. Preprint SC 97-16, Konrad-Zuse-Zentrum für Informationstechnik Berlin. Available via WWW at www.zib.de.

Löbel, A. (1997b). *MCF Version 1.0 – A network simplex implementation*. Available for academic use free of charge via WWW at www.zib.de.

Löbel, A. (1997c). Recent computational developments for large-scale multiple-depot vehicle scheduling problems. Preprint SC 97-17, Konrad-Zuse-Zentrum für Informationstechnik Berlin. Available via WWW at www.zib.de.

Löbel, A. (1997d). Vehicle scheduling in public transit and Lagrangean pricing. Revised Preprint SC 96-26, Konrad-Zuse-Zentrum für Informationstechnik Berlin. Available via WWW at www.zib.de.

Löbel, A. and Strubbe, U. (1996). Wagenumlaufoptimierung – Methodischer Ansatz und praktische Anwendung. In *Heureka '96: Optimierung in Verkehr und Transport*, pages 341–355. Forschungsgesellschaft für Straßen- und Verkehrswesen, Köln. In German. Available as ZIB preprint SC 95-38 via WWW at www.zib.de.

Luenberger, D. G. (1989). *Linear and nonlinear programming*. Addison-Wesley, second edition.

Lustig, I. J. (1990). The influence of computer language on computational comparisons: An example from network optimization. *ORSA Journal on Computing*, 2(2):152–161.

Mesquita, M. and Paixão, J. (1992). Multiple depot vehicle scheduling problem: A new heuristic based on quasi-assignment algorithms. In Desrochers and Rousseau [1992].

Mesquita, M. and Paixão, J. M. P. (1997). Exact algorithms for the multi-depot vehicle scheduling problem based on multicommodity network flow type formulations. In *7th International Workshop on Computer-Aided Scheduling of Public Transport*, pages 269–290. Center for Transportation Studies, MIT, Cambridge, USA.

Mojsilovic, M. (1983). Verfahren für die Bildung von Fahrzeugumläufen, Dienstplänen und Dienstreihenfolgeplänen. In *Heureka 83 – Optimierung in Transport und Verkehr*, pages 178–191. Forschungsgesellschaft für Straßen- und Verkehrswesen, Köln.

Nemhauser, G. L., Rinnooy Kan, A. H. G., and Todd, M. J., editors (1989). *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science*. Elsevier Science B.V., Amsterdam.

Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc.

Padberg, M. W. (1973). On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215.

Papadimitriou, C. H. and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Petzold, P. and Schütze, P. (1995). Integrated data processing for public transport in Hamburg. In Daduna, Branco, and Paixão [1995].

Plotkin, S., Shmoys, D. B., and Tardos, É. (1991). Fast approximation algorithms for fractional packing and covering problems. In *32nd Annual Symposium on Foundations of Computer Science*, pages 495–504. IEEE Computer Society Press.

Polyak, B. T. (1967). A general method of solving extremum problems. *Soviet Mathematics Doklady*, 8:593–597. English translation.

Ribeiro, C. C. and Soumis, F. (1994). A column generation approach to the multiple-depot vehicle scheduling problem. *Operations Research*, 42(1):41–52.

Ryan, D. M. and Foster, B. A. (1981). An integer programming approach to scheduling. In Wren [1981], pages 269–280.

Schmidt, V. A. (1997). *Auf Sparkurs zum Ziel*. Rheinischer Merkur, number 39, page 37, 26th September 1997. In German.

Schrijver, A. (1989). *Theory of Linear and Integer Programming*. John Wiley & Sons Ltd., Chichester.

Schütze, P. and Völker, M. (1995). Recent developments of HOT II. In Daduna, Branco, and Paixão [1995].

Sol, M. (1994). *Column Generation Techniques for Pickup and Delivery Problems.* PhD thesis, Technische Universiteit Eindhoven.

Soumis, F. (1997). *Decomposition and Column Generation.* Chapter 8 in Dell'Amico, M., Maffioli, F., and Martello, S., editors (1997), *Annotated Bibliographies in Combinatorial Optimization.* John Wiley & Sons Ltd, Chichester.

Thienel, S. (1995). *ABACUS A Branch-And-CUt System.* PhD thesis, Universität zu Köln.

VDV (1992). Das Fachwort im Verkehr: Wirtschaftliche Begriffe des ÖPNV, Wörterbuch deutsch, englisch, fanzösisch. Alba Fachverlag GmbH & Co. KG, Düsseldorf, Germany. Verband Deutscher Verkehrsunternehmen (VDV), Köln.

Veldhorst, M. (1993). A bibliography on network flow problems. In Du and Pardalos [1993], pages 301–331.

VHH. *Verkehrsbetriebe Hamburg-Holstein AG.* Information available via WWW at www.oepnv.de/vhh.

Wolsey, L. A. (1975). Faces of linear inequalities in 0-1 variables. *Mathematical Programming*, 8:165 – 178.

Wren, A., editor (1981). *Computer scheduling of public transport: Urban passenger vehicle and crew scheduling.* North-Holland Publishing Company.

Index

Curriculum Vitae

Andreas Löbel

geboren am 28. August 1967 in Memmingen.

1973-1979:	Grundschule in Memmingen
1979-1983:	Wirtschaftschule Memmingen
Juni 1983:	Mittlere Reife
1983-1985:	Berufsoberschule Memmingen
Juni 1985:	Fachgebundene Hochschulreife
1985-1987:	Wehrdienst
1987-1992:	Studium der Wirtschaftsmathematik an der Universität Augusburg
Januar 1992:	Diplom
seit 1992:	Wissenschaftlicher Angestellter am Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Abteilung Optimierung.