FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

On Solving Constrained Tree Problems and an Adaptive Layers Framework

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der technischen Wissenschaften

by

Dipl.-Ing. Mario Ruthmair
Registration Number 9826157

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: a.o.Univ.-Prof. Dipl.-Ing. Dr.techn. Günther R. Raidl

The dissertation has been reviewed by:

(a.o.Univ.-Prof. Dipl.-Ing. Dr.techn. Günther R. Raidl)	(a.o.Univ.-Prof. Dipl.-Ing. Dr.techn. Ulrich Pferschy)

Wien, 27.05.2012

(Dipl.-Ing. Mario Ruthmair)

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Mario Ruthmair
Herbeckstraße 80/1, 1180 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quellen als Entlehnungen kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

First of all I want to greatly thank my supervisor Günther Raidl for the excellent working environment within the Algorithms and Data Structures Group and for giving me enough freedom to evolve while guiding me to meaningful directions. The last years have been one of the most exciting and valuable years of my life and the personal advancement in terms of knowledge and self-confidence within these times is incomparable.

Furthermore, thanks to my second supervisor Ulrich Pferschy for providing valuable comments on this thesis which helped to improve its quality. Thanks to my former colleagues Andy Chwatal, Martin Gruber, Sandro Pirkwieser, and Matthias Prandtstetter, for introducing me to the group's research fields and supporting me in my early teaching experiences. Especially Martin Gruber always had an open ear for technical, bureaucratic, and research problems of any kind. Thanks to Bin Hu for caring about many parts of our teaching responsibilities and for taking everything so easy without ever being in a bad mood. Thanks to Markus Leitner for numerous fruitful discussions about integer programming and a very efficient and enjoyable collaboration. Also my new colleagues Emir Causevic, programming guru Johannes Inführ, Marian Rainer-Harbach, and Christian Schauer deserve gratitude for heavily supporting and enriching our group. Without Johannes' tools to efficiently evaluate experimental results this thesis would have taken much longer. Not to forget about the people behind the scenes who care about organizational and technical stuff and thus keep everything going, namely Doris Dicklberger and Andi Müller (formerly Aksel Filipovic and Angela Schabel).

Special thanks go to my parents for supporting me in all decisions I ever made and always trusting in my self-reliance. If I needed any help I got it without discussion. And I can be sure that it will remain this way the rest of my life.

Last but not least I want to thank Daniela for her endless patience, support, understanding, and especially her power of endurance in the last years (I promise you that I will not append a second PhD!). Thanks for taking care of my social and culinary well-being, for extending my view on non-algorithmic (also relevant!) topics, and finally for bringing Lea and Archie into my life who numerously removed the stress by just being present.

Abstract

In this thesis we consider selected combinatorial optimization problems arising in the field of network design. In many of these problems there is a central server sending out information to a set of recipients. A common objective is then to choose connections in the network minimizing the total costs. Besides this, current applications, e.g. in multimedia, usually force additional quality-of-service constraints, e.g. limiting the communication delay between the central server and the clients. In general, these problems can be modeled on a graph and in many cases an optimal solution corresponds to a rooted tree with minimum costs satisfying all the given constraints. The most relevant of these optimization problems are $\mathcal{NP}$-hard making it necessary – provided that $\mathcal{P} \neq \mathcal{NP}$ – to develop sophisticated algorithmic approaches to obtain high quality or even optimal solutions.

Due to the complexity of these optimization problems it is usually not possible to obtain proven optimal solutions for medium- to large-sized problem instances in reasonable time. Therefore, heuristic approaches yielding high quality but in general sub-optimal solutions are of high practical interest. Metaheuristics and hybrid variants combining heuristic and exact solution techniques recently increased in popularity due to their successful application on many important optimization problems.

We present new state-of-the-art solution approaches for several of these optimization problems. Given a problem instance we first apply reduction rules identifying and removing nodes and edges in the graph which can only be part of infeasible or sub-optimal solutions. The more the input graph can be reduced in this way a priori the easier it is in general for an algorithm to find a feasible or optimal solution.

We designed several heuristic approaches for the *rooted delay-constrained minimum spanning tree (RDCMST) problem* in which all nodes in a graph have to be connected to a fixed root node while the total delay on the paths from the root to any other node has to be within a given delay-bound. For constructing a feasible solution we suggest a heuristic based on Kruskal's minimum spanning tree algorithm and another one utilizing the multilevel refinement paradigm. Improvements to these obtained solutions are achieved by applying a greedy randomized adaptive search procedure, local search in two different neighborhood structures, and embedding this local search in a general variable neighborhood search, an ant colony optimization approach, and a genetic algorithm. The appearance of duplicate solutions within the genetic algorithm is discussed and appropriate methods dealing with them are presented. Extensive computational results indicate the superiority of the evolutionary approach and the variable neighborhood search.

Additionally, we tackled small- to medium-sized problem instances with exact algorithms, mostly concentrating on mathematical programming methods since these turned out to perform

well on numerous related network design problems in the literature. Especially modeling these problems on so-called layered graphs has been shown to yield good results. We compared different modeling approaches for the *rooted delay-constrained Steiner tree (RDCST) problem* which is a generalization of the RDCMST problem where only a subset of the nodes is required to be connected to the root node. Computational results indicate that three methods dominate the comparison: a branch-and-price approach stabilized by using alternative dual-optimal solutions, a model on a corresponding layered graph, and a formulation based on an exponential number of subtour elimination and infeasible path inequalities.

In some situations, e.g. in Voice-over-IP applications, it is not only important that all recipients receive the information within a given delay-bound but also nearly at the same time. This additional constraint is modeled in the *rooted delay- and delay-variation-constrained Steiner tree (RDDVCST) problem*. For this problem we compare mixed integer programming formulations based on multi-commodity flows and again a transformation to a layered graph. The latter approach extended by some valid inequalities turned out to be clearly superior to the flow-based model.

Since the performance of layered graph approaches strongly depends on the sizes of the set of achievable path delay values and on given delay-bounds their practical applicability is limited. Thus, we extend these methods to a generally-applicable iterative *adaptive layers framework* (ALF) mitigating their disadvantages and emphasizing their benefits. Basically, ALF approximates the linear programming relaxation and the optimal integer solution of a complete layered graph formulation by solving a sequence of usually much smaller models and thus partly overcomes possible problems with huge layered graphs. The additional overhead of repeated model solving pays off in many cases, as experimental results indicate, especially on large sparse graphs ALF outperforms all other approaches for the RDCST problem. Additionally, we provide two case studies on applying ALF to further problems: For an extended variant of the RDCST problem with consideration of node prizes and a quota constraint ALF is clearly superior to other methods, in many cases even by orders of magnitudes. The second case study considers the *vehicle routing problem with time windows*: Here, we discuss a modeling approach on two separated layered graphs and another one on a three-dimensional layered graph. Preliminary results indicate that futher work in this direction is promising.

Kurzfassung

Die vorliegende Arbeit behandelt ausgewählte kombinatorische Optimierungsprobleme im Bereich des Netzwerkdesigns. In vielen dieser Probleme kommuniziert ein zentraler Server mit einer Gruppe von Clients, wobei es üblicherweise das Ziel ist, kostenminimale Wege im Netzwerk zu finden. Neben diesem Optimierungsziel erfordern aktuelle Anwendungen, z.B. im Multimedia-Bereich, die Einhaltung weiterer sogenannter Quality-of-Service Bedingungen, die unter anderem in der Beschränkung der Übertragungszeit zwischen Server und Clients bestehen. Diese Art von Problemen kann oft auf einem Graph modelliert werden, wobei eine optimale Lösung meistens einem Baum entspricht, der den Server und alle Clients beinhält und alle geforderten Nebenbedingungen erfüllt. Die wichtigsten dieser Probleme sind jedoch $\mathcal{NP}$-schwer, was dazu führt – vorausgesetzt $\mathcal{P} \neq \mathcal{NP}$ –, dass aufwendige und raffinierte Verfahren gefunden werden müssen, um gute bzw. optimale Lösungen zu erhalten.

Aufgrund der Komplexität dieser Optimierungsprobleme ist es üblicherweise nicht möglich beweisbar optimale Lösungen für größere Probleminstanzen in angemessener Zeit zu finden. Deshalb verwendet man in der Praxis oft heuristische Ansätze, die zwar im Allgemeinen nur zu suboptimalen aber dennoch zu sehr guten Lösungen führen. Metaheuristiken und hybride Varianten, die heuristische und exakte Verfahren kombinieren, gewannen in den letzten Jahren immer mehr an Beliebtheit, da sie für eine Vielzahl von wichtigen Optimierungsproblemen bereits überaus erfolgreiche Resultate erzielt haben.

Wir präsentieren neue State-of-the-Art Ansätze um einige dieser Probleme zu lösen, wobei wir zu allererst versuchen die gegebene Probleminstanz zu reduzieren, in dem wir Knoten und Kanten identifizieren, die entweder in keiner oder nur in einer suboptimalen Lösung enthalten sein können, und entfernen diese dann aus dem Graph. Je mehr der Graph in dieser Phase reduziert werden kann, desto einfacher ist es üblicherweise eine gültige oder optimale Lösung zu finden.

Für das sogenannte *Rooted Delay-Constrained Minimum Spanning Tree (RDCMST) Problem*, in dem alle Knoten in einem Graph mit dem vorgegebenen Wurzelknoten verbunden werden müssen und das Gesamtdelay jedes Pfads vom Server zu einem Client eine maximale Schranke nicht überschreiten darf, haben wir verschiedene heuristische Ansätze entwickelt. Um eine gültige Lösung zu konstruieren, wenden wir Heuristiken an, die auf Kruskal's Algorithmus zum Finden eines minimalen Spannbaums oder dem Multilevel-Refinement-Paradigma basieren. Weitere Verbesserungen dieser Lösungen werden durch folgende Verfahren erzielt: einer Greedy-Randomized-Adaptive-Search-Procedure, einer lokalen Suche in verschiedenen Nachbarschaftsstrukturen und der Einbettung dieser in einer variablen Nachbarschaftssuche, eines Ant-Colony-Optimization Ansatzes und eines genetischen Algorithmus. Das Vorkommen von

Duplikaten im genetischen Algorithmus wird diskutiert und entsprechende Verfahren werden vorgestellt, um mit diesen geeignet umzugehen. Experimentelle Ergebnisse haben schließlich die Überlegenheit des evolutionären Ansatzes und der variablen Nachbarschaftssuche gegenüber den restlichen Methoden gezeigt.

Zusätzlich zu diesen (meta-)heuristischen Ansätzen versuchen wir kleinere bis mittelgroße Probleminstanzen exakt zu lösen, wobei wir uns hier hauptsächlich auf Methoden der mathematischen Programmierung konzentrieren, die sich in einer Vielzahl von existierenden Arbeiten zu Netzwerkdesignproblemen als überaus erfolgreich gezeigt haben. Speziell die Modellierung dieser Probleme auf einem sogenannten Layered-Graph haben besonders gute Ergebnisse erzielt. Anhand des *Rooted Delay-Constrained Steiner Tree (RDCST) Problems*, das eine Generalisierung des RDCMST Problems darstellt, in der nur eine Untermenge der vorhandenen Knoten an den Wurzelknoten angeschlossen werden muss, vergleichen wir verschiedene Modellierungsansätze. Die experimentellen Resultate zeigen, dass drei Methoden die restlichen übertreffen: ein Branch-and-Price Ansatz, der durch die Verwendung von alternativen dual-optimalen Lösungen beschleunigt wird, ein Modell auf einem entsprechenden Layered-Graph und eine Formulierung, die eine exponentielle Anzahl von Subtour-Eliminations- und verbesserten Pfadungleichungen enthält.

In manchen Situation, z.B. in Voice-over-IP-Anwendungen, ist es nicht nur wichtig, dass alle Empfänger die Informationen innerhalb einer gewissen Zeitspanne erhalten, sondern auch ungefähr zur gleichen Zeit. Diese zusätzliche Bedingung wird im sogenannten *Rooted Delay- and Delay-Variation-Constrained Steiner Tree (RDDVCST) Problem* modelliert, wobei wir hier Integer-Programming-Formulierungen basierend auf Informationsflüssen bzw. einem Layered-Graph vergleichen. Der letztere der beiden Ansätze, erweitert durch stärkende Ungleichungen, erwies sich gegenüber dem Flussmodell als weit überlegen.

Die praktische Anwendbarkeit der Layered-Graph-Ansätze ist teilweise eingeschränkt, da deren Effizienz stark von der Menge der realisierbaren Pfaddelays und der gegebenen Zeitschranken abhängt. Deshalb haben wir diese Methoden zu einem generellen iterativen *Adaptive Layers Framework* (ALF) erweitert, das die Nachteile dieser Ansätze teilweise abschwächt und dennoch von deren Stärken profitiert. Im Grunde approximiert ALF eine optimale Lösung des ganzzahligen Modells und dessen fraktionaler Relaxierung auf dem kompletten Layered-Graph durch das Lösen einer Serie von üblicherweise viel kleineren Modellen, und kann dadurch teilweise die Probleme mit sehr großen Layered-Graphen vermeiden. Wie die Ergebnisse zeigen, lohnt sich der zusätzliche Aufwand für das wiederholte Lösen von Modellen in vielen Fällen, wobei speziell auf großen dünnen Graphen ALF alle anderen Ansätze für das RDCST Problem klar aussticht. Zusätzlich führen wir noch zwei Fallstudien auf anderen Problemen an: Für eine erweiterte Variante des RDCST Problems mit Berücksichtigung von Profiten auf Knoten und einer Quotenbedingung zeigte sich ALF in vielen Fällen sogar um Größenordnungen besser. In der zweiten Fallstudie betrachten wir das *Vehicle Routing Problem with Time Windows* und diskutieren ein Modell auf zwei getrennten Layered-Graphen und ein weiteres auf einem dreidimensionalen Layered-Graph. Erste Ergebnisse belegen, dass diese Ansätze durchaus vielversprechend erscheinen.

Contents

CHAPTER 1

Introduction

According to a recent analysis in 2011 by Cisco Systems [29] – a big player in networking business – nowadays streaming of video and audio over networks, e.g. in multimedia and Voice-over-IP (VoIP) applications, gets more and more popular and in several forecasts this trend is believed to hold on in future. Even with our current quickly increasing amount of available bandwidth we have to find more efficient ways of transmitting this information to all recipients. Repeated re-sending of the same data packets to each client within a network may not be possible anymore if the demand for video streaming further increases and television broadcast over internet overtakes common transmission by satellite or dedicated cable.

In this thesis we consider combinatorial optimization problems (COPs) arising in the field of network design which represent a highly important and practically relevant class of COPs. In many of these problems there is a central server sending out information to a set of recipients, possibly via optional intermediate nodes, respecting diverse resource and quality-of-service (QoS) constraints. One commonly desired QoS constraint is a limitation of the communication delay between the server and the clients. Additionally, in VoIP and video conferencing multicast scenarios it is not only important that all participants receive the information from the central server within a given time limit but also nearly at the same time. Otherwise upcoming race conditions possibly result in misunderstandings between the clients. In database replication scenarios it is necessary to guarantee the consistency of all mirroring databases. Thus, if updates have to be deployed the time interval between the first and the last client database applying the changes should be within a known limit. Buffering information at the server or intermediate nodes in the network shall be avoided as in general it would increase the total delay and requires the repeated sending of the same data, annihilating the advantage of distributing information over a multicast tree. Finally, buffering at the clients is not always a choice since in some online applications, e.g. gaming and stocktrading, competing users may benefit from receiving information earlier than others and thus may circumvent the local data retention.

If considering a problem variant in which all terminals need to be connected obligatorily one usually aims to identify a solution yielding overall minimal costs. These usually non-negative costs often depend on the effort to establish a particular network node or link, on used technolo-

gies, and the utilization of the corresponding resources. On the contrary, in many real world applications the primary goal is to maximize the net profit, which is the profit earned by connecting customers reduced by the investment to build the network. Such scenarios are frequently called prize collecting network design problems.

In a completely different application we may consider a package shipment organization with a central depot and a distribution network possibly consisting of several intermediate storage facilities. This company might guarantee its customers a delivery of certain commodities within a specified time horizon, e.g. because of perishable products. Naturally, the organization wants to minimize the transportation costs but at the same time wants to hold its promise of being in time. Also this type of problems can be seen as a network design problem and modeled by similar COPs as the previous applications.

1.1 Combinatorial Optimization Problems

In general, a COP is defined as follows [190]:

Definition 1.1.1. *Let S be a set of base elements, $c : S \to \mathbb{R}$ be a cost function assigning each element a cost value, and $X \subseteq 2^S$ be the set of feasible subsets of S. The problem of finding a minimum cost feasible subset is a combinatorial optimization problem (COP)*

$$\min_{x \in X} \sum_{s \in x} c_s. \tag{1.1}$$

A similar definition can be provided for maximization problems in an obvious way.

Usually, network design problems can be modeled on a graph $G = (V, E)$ with several properties and resources assigned to nodes V and edges E. In case of positive cost values assigned to edges $e \in E$ a subgraph with minimum costs connecting all required nodes corresponds to a tree [124]. In the simplest case when all nodes need to be connected, such a problem can be modeled as a spanning tree problem efficiently solvable by Kruskal's [107] or Prim's [145] algorithms, but additional options like possibly includable intermediate nodes, delay, length and/or more general resource constraints, and different objectives make these kind of problems most of the time $\mathcal{NP}$-hard. Thus, provided that $\mathcal{P} \neq \mathcal{NP}$ in general there is no algorithm which obtains a proven optimal solution in polynomial time, and therefore moderate to large instances of a given COP are frequently difficult to solve to optimality in practice. As long as aspects like redundant connections to terminals in order to achieve higher connectivity and robustness to failure are excluded, solutions have tree structure, and such problems can be modeled as extensions of the *Steiner tree problem on a graph* [44].

Exact approaches for $\mathcal{NP}$-hard COPs often incorporate (mixed) integer programming (MIP) techniques [134] since they proved to be quite successful for numerous problems in literature. Here we also focus on applying these concepts to the considered network design problems on small- to moderately-sized instances. Additionally, due to the complexity of these optimization problems heuristic approaches yielding high quality but in general sub-optimal solutions are of strong practical interest especially for large-scale problem instances. Metaheuristics [53] and hybrid variants [125, 154] combining heuristic and exact solution techniques recently increased in

popularity due to their successful application on many important optimization problems. Therefore, we consider these kinds of approaches for problem instances where our exact methods are not able to provide any useful results within reasonable time and memory limits.

1.2 Considered Problems

We consider the following three network design problems modeling the previously mentioned application scenarios, ordered from most specialized to most general:

1. *Rooted Delay-Constrained Minimum Spanning Tree (RDCMST) Problem*: This problem models the situations when a central server $s \in V$ needs to broadcast information to all other nodes $V \setminus \{s\}$ in the network while minimizing the total costs of establishing the network and satisfying a pre-defined global upper delay-bound on the paths from the server to any other client.

2. *Rooted Delay-Constrained Steiner Tree (RDCST) Problem*: This problem is a generalization of the RDCMST problem since it requires only a subset of the nodes in the network denoted as terminal nodes $R \subseteq V \setminus \{s\}$ to be connected to the server. The remaining potential Steiner nodes $V \setminus (R \cup \{s\})$ which e.g. represent routers can be optionally used as intermediate relay nodes to further decrease connection costs or delays.

3. *Rooted Delay- and Delay-Variation-Constrained Steiner Tree (RDDVCST) Problem*: This problem is a generalization of the RDCST problem since it additionally considers a so-called delay-variation constraint: Here, the overall delays of the paths from the server to the required clients are not allowed to differ too much which as already mentioned is important for VoIP, database replication, and other applications where all participants should receive information nearly simultaneously.

In all three problems cost and delay values are in general uncorrelated properties assigned to the edges. Typically, considered problem graphs are undirected allowing only symmetric links between nodes. However, the problems can easily be extended to directed networks allowing asymmetric connections with different costs and/or delays for opposite arcs. For some applications directed graphs may be more realistic e.g. “because of the asymmetric nature of communication networks” [55].

Clearly, different constraints may have significant impact on the structure and especially the overall cost of a solution which can be easily observed in Fig. 1.1 where optimal solutions to different problems on the same network are shown.

Costs and delays may not only incur on links but also on intermediate or terminal nodes. However, in case of directed networks all node costs and delays can be added to incoming and outgoing arcs, respectively, without modifying the set of feasible and optimal solutions: If particular costs and delays incur as soon as the node is visited we add them to the costs and delays of all incoming arcs, respectively. If costs or delays are only raised when the corresponding node is utilized as relay node then we add the values to all outgoing arcs. Therefore, when considering client-server-networks after appropriate transformation an arc delay may include the delays e.g.

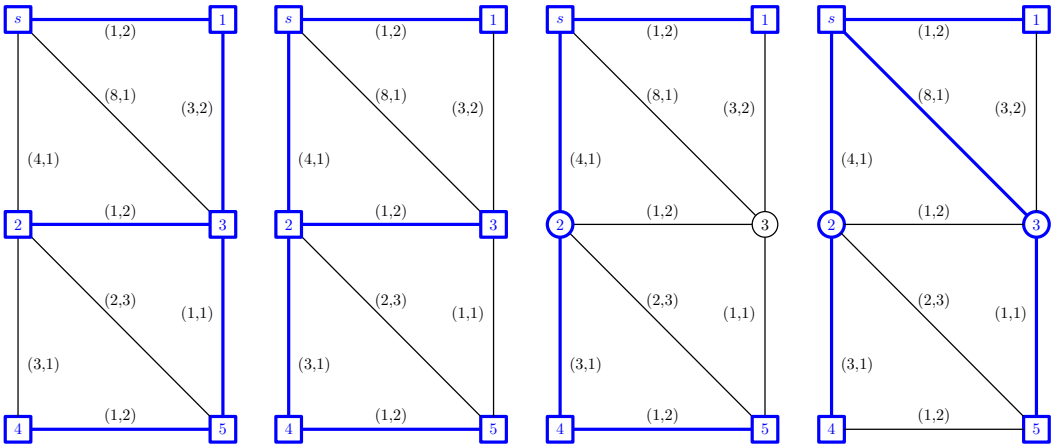


Figure 1.1: Edge labels denote $(cost, delay)$. (a) Optimal solution to the minimum spanning tree problem with total costs 7. (b) Optimal solution to the RDCMST problem with delay-bound 4 on the paths from server s to each client, and with total costs 10. (c) Optimal solution to the RDCST problem with delay-bound 4 and total costs 9 (squared nodes denote terminal nodes and circles represent optional relay nodes). (d) Optimal solution to the RDDVCST problem with delay-bound 4 and variation-bound 1 (the path delays from server s to the required clients are not allowed to differ by more than 1), and with total costs 17.

for switching, queuing, transmission, and propagation. In undirected graphs not all kinds of node costs and delays can be moved to the edges. In these situations usually edges are replaced by two oppositely directed arcs.

1.3 Structure of the Thesis

The remainder of this thesis is structured in the following way: Chapter 2 briefly discusses the methodology used as base for the solution approaches in the next parts. Exact methods for COPs mainly focusing on integer programming, several (meta-)heuristics, and finally hybrid approaches combining different concepts are described.

The next three chapters discuss methods solving the previously introduced problems: Chapter 3 is devoted to the RDCMST problem and presents two construction heuristics and several metaheuristics: a greedy randomized adaptive search procedure, a variable neighborhood descent, a general variable neighborhood search, an ant colony optimization approach, and a genetic algorithm. Most parts of this chapter have been published in

> Mario Ruthmair and Günther R. Raidl. A Kruskal-Based Heuristic for the Rooted Delay-Constrained Minimum Spanning Tree Problem. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, Proceedings of the 12th International Con-

ference on Computer Aided Systems Theory, volume 5717 of LNCS, pages 713-720. Springer, 2009.

Martin Berlakovich, Mario Ruthmair, and Günther R. Raidl. A Multilevel Heuristic for the Rooted Delay-Constrained Minimum Spanning Tree Problem. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I, volume 6927 of LNCS, pages 256-263. Springer, 2012.

Mario Ruthmair and Günther R. Raidl. Variable Neighborhood Search and Ant Colony Optimization for the Rooted Delay-Constrained Minimum Spanning Tree Problem. In R. Schaefer et al., editors, Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part II, volume 6239 of LNCS, pages 391-400. Springer, 2010.

Mario Ruthmair and Günther R. Raidl. A Memetic Algorithm and a Solution Archive for the Rooted Delay-Constrained Minimum Spanning Tree Problem. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I, volume 6927 of LNCS, pages 351-358. Springer, 2012.

Chapter 4 discusses exact methods based on integer programming for the RDCST problem. Several modeling approaches are compared to previously proposed ones: a branch-and-price stabilized by using alternative dual-optimal solutions, a path-cut formulation with directed connection cut and infeasible path inequalities, and a model based on a transformation to a layered graph strengthened by additional valid inequalities. Some parts of this chapter are published in

Mario Ruthmair and Günther R. Raidl. A Layered Graph Model and an Adaptive Layers Framework to Solve Delay-Constrained Minimum Tree Problems. In O. Günlük and G.J. Woeginger, editors, Proceedings of the 15th Conference on Integer Programming and Combinatorial Optimization (IPCO XV), volume 6655 of LNCS, pages 376-388. Springer, 2011.

Markus Leitner, Mario Ruthmair, and Günther R. Raidl. Stabilized Column Generation for the Rooted Delay-Constrained Steiner Tree Problem. In Proceedings of the VII ALIO/EURO - Workshop on Applied Combinatorial Optimization, pages 250-253, Porto, Portugal, 2011.

Markus Leitner, Mario Ruthmair, and Günther R. Raidl. Stabilized Branch-and-Price for the Rooted Delay-Constrained Steiner Tree Problem. In J. Pahl, T. Reiners, and S. Voß, editors, Network Optimization: 5th International Conference, INOC 2011, volume 6701 of LNCS, pages 124-138, Hamburg, Germany, 2011. Springer.

Markus Leitner, Mario Ruthmair, and Günther R. Raidl. On Stabilized Branch-and-Price for Constrained Tree Problems. Technical Report TR 186-1-11-01, Vienna

University of Technology, Vienna, Austria, 2011. accepted with revisions to Networks (INOC 2011 special issue).

Chapter 5 proposes two MIP approaches for solving the RDDVCST problem: a multi-commodity flow model and a layered graph formulation similarly to the one for the RDCST problem but additionally considering the delay-variation-constraint and extended by a new set of valid inequalities. Most parts of this chapter are published in

Mario Ruthmair and Günther R. Raidl. On Solving the Rooted Delay- and Delay-Variation-Constrained Steiner Tree Problem. In Proceedings of the 2nd International Symposium on Combinatorial Optimization, LNCS. Springer, 2012 (to appear).

Chapter 6 introduces the so-called Adaptive Layers Framework (ALF) which tries to partly overcome major computational issues of layered graph approaches. We describe basics of the generally-applicable ALF by illustration on the RDCST problem and then present two more specific case studies on different problems: the quota-constrained rooted delay-constrained Steiner tree problem which is a generalization of the RDCST problem and the vehicle routing problem with time windows. The basic parts of this chapter have been published in

Mario Ruthmair and Günther R. Raidl. A Layered Graph Model and an Adaptive Layers Framework to Solve Delay-Constrained Minimum Tree Problems. In O. Günlük and G.J. Woeginger, editors, Proceedings of the 15th Conference on Integer Programming and Combinatorial Optimization (IPCO XV), volume 6655 of LNCS, pages 376-388. Springer, 2011.

Furthermore, a talk on an extension of ALF and some further preliminary results has been given at the INFORMS Telecommunications Conference:

Mario Ruthmair. An Adaptive Layers Framework for Resource-Constrained Network Design Problems. 11th INFORMS Telecommunications Conference, Boca Raton, Florida, USA, 2012.

Finally, Chapter 7 concludes the thesis by summarizing the major results.

CHAPTER 2

Methodology

This chapter discusses basic methods and general principles used to solve combinatorial optimization problems (COPs). Usually, solution methods are classified into two domains: exact approaches (Section 2.1) aim at providing solutions with a certificate of optimality whereas heuristic ones (Section 2.2) only try to find solutions as good as possible in many cases without knowledge of the "distance" to optimality. Since both approaches have their benefits and disadvantages it seems to be quite natural to combine successful elements from both domains to form so-called hybrid methods briefly discussed in Section 2.3.

Furthermore, we will only concentrate on COPs with a single objective and deterministic input data. However, multi-objective [38], stochastic [84] and robust [16–18] optimization are highly relevant and upcoming fields of research since in practical applications we often have to deal with multiple objectives and uncertain data.

Our objective is not to give a complete overview of existing methods in literature but to only discuss those in more detail which are relevant for our approaches in the following Chapters 3–6. The structure of this chapter follows in some parts the corresponding presentation in the PhD thesis of Markus Leitner [112].

2.1 Exact Methods

If one is faced with an optimization problem the natural approach is to search for a best possible, i.e. an optimal, solution to this problem. In most cases in practice it is also sufficient to find one optimal solution even when there are multiple optima with the same objective value. If considering an $\mathcal{NP}$-hard problem – as it is the case for many relevant applications – there is no polynomial time algorithm to solve it to proven optimality, unless $\mathcal{P} = \mathcal{NP}$ [52, 138]. Thus, an exact algorithm for such problems in general requires exponential time to find an optimal solution which makes it hard or even impossible to solve large instances of a given COP in reasonable time.

One of the most promising solution approaches for a wide range of COPs is to model the problem as (mixed) integer linear program (MIP) and solve it by appropriate mathematical pro-

gramming methods. Following successful MIP approaches in literature we adopted and applied these techniques to our problems, see Chapters 4–6. Therefore, in the remainder of this section we will present the basics of these prominent methods based on the contents of well-known books on this topic [19, 20, 34, 134, 168, 190].

2.1.1 Linear Programming

Linear programs (LPs) commonly appear as subproblems within MIP approaches and the theoretical concepts and results related to LPs build the foundation of integer programming and further extensions. Thus, we briefly discuss how LPs are defined and how we can find feasible and optimal solutions to them. In general, an LP defines a set of feasible solutions by a set of linear (in-)equalities and evaluates these solutions by using a linear objective function. A solution with minimal (or maximal) objective value is then called optimal solution.

More formally, we are given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and vectors $\boldsymbol{c} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^m$ with real-valued elements. Vector $\boldsymbol{c}'$ denotes the transposed vector $\boldsymbol{c}$. A general formulation of an LP is defined as follows:

$$z_{\mathrm{LP}} = \min \quad \boldsymbol{c}'\boldsymbol{x} \tag{2.1}$$
$$\text{subject to} \quad \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \tag{2.2}$$
$$\boldsymbol{x} \in \mathbb{R}^n_+ \tag{2.3}$$

Note that LPs are not restricted to inequalities as side constraints since equalities obviously can be represented by two corresponding inequalities with opposite signs. Moreover, any LP can be transformed to an equivalent formulation only using equalities with additional slack and surplus variables. Furthermore, maximization problems can be easily converted to minimization problems by inverting the sign of the objective function.

Alternatively, a linear program P can be written in the following way:

$$z_{\mathrm{LP}} = \min\{\boldsymbol{c}'\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b},\ \boldsymbol{x} \in \mathbb{R}^n_+\}. \tag{2.4}$$

Duality

Duality is a fundamental and important concept in LP theory and utilized in many solution methods for LPs and MIPs. We can formulate a corresponding *dual* linear program D for each *primal* LP (2.1)–(2.3) in the following way:

$$w_{\mathrm{LP}} = \max \quad \boldsymbol{u}'\boldsymbol{b} \tag{2.5}$$
$$\text{subject to} \quad \boldsymbol{u}'\boldsymbol{A} \leq \boldsymbol{c}' \tag{2.6}$$
$$\boldsymbol{u} \in \mathbb{R}^m_+, \tag{2.7}$$

or alternatively:

$$w_{\mathrm{LP}} = \max\{\boldsymbol{u}'\boldsymbol{b} \mid \boldsymbol{u}'\boldsymbol{A} \leq \boldsymbol{c}',\ \boldsymbol{u} \in \mathbb{R}^m_+\}. \tag{2.8}$$

It can be easily seen that by transforming the dual LP to its dual using the same conversion rules we again obtain the primal LP.

A vector $\hat{\boldsymbol{x}} \in \mathbb{R}^n_+$ is *primal feasible* if all constraints in the primal LP are satisfied, i.e. if $\boldsymbol{A}\hat{\boldsymbol{x}} \geq \boldsymbol{b}$ holds. Similarly, $\hat{\boldsymbol{u}} \in \mathbb{R}^m_+$ is *dual feasible* if $\hat{\boldsymbol{u}}'\boldsymbol{A} \leq \boldsymbol{c}'$. The weak duality theorem is stated as follows.

Theorem 2.1.1 (Weak Duality)**.** *Given are a primal linear program P and its corresponding dual linear program D. Then, $\boldsymbol{c}'\hat{\boldsymbol{x}} \geq z_{\mathrm{LP}} \geq w_{\mathrm{LP}} \geq \hat{\boldsymbol{u}}'\boldsymbol{b}$ holds if $\hat{\boldsymbol{x}}$ is primal feasible and $\hat{\boldsymbol{u}}$ is dual feasible.*

Weak duality can be extended to the even more important and fundamental concept of strong duality:

Theorem 2.1.2 (Strong Duality)**.** *Given are a primal linear program P and its corresponding dual linear program D. If either z_{LP} or w_{LP} is finite, then both the primal and dual LPs have finite optimal solution values, and $z_{\mathrm{LP}} = w_{\mathrm{LP}}$.*

The previous two duality theorems imply exactly four possible results for a pair of primal and dual LPs P and D, respectively:

1. Optimal solutions $\boldsymbol{x}^*$ and $\boldsymbol{u}^*$ for both P and D exist and have finite and equal objective values, i.e. $\boldsymbol{c}'\boldsymbol{x}^* = z_{\mathrm{LP}} = w_{\mathrm{LP}} = (\boldsymbol{u}^*)'\boldsymbol{b}$.
2. P is unbounded, i.e. $z_{\mathrm{LP}} = -\infty$, and thus D is infeasible
3. D is unbounded, i.e. $w_{\mathrm{LP}} = \infty$, and thus P is infeasible
4. both P and D are infeasible

Furthermore, the *complementary slackness* conditions are a consequence of strong duality:

Proposition 2.1.3. *Let $\boldsymbol{x}^*$ and $\boldsymbol{u}^*$ be optimal solutions to P and D, respectively, then*

$$x_j^*((\boldsymbol{u}^*)'\boldsymbol{A} - \boldsymbol{c}')_j = 0 \quad \forall j \in \{1, ..., n\}, \tag{2.9}$$

$$u_i^*(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^*)_i = 0 \quad \forall i \in \{1, ..., m\}. \tag{2.10}$$

Note that the *maximum flow minimum cut* theorem [5], which states that the maximum flow from a source node s to a target node t in a directed graph with given arc capacities is equivalent to an s-t-cut with minimum capacity, can be shown by application of LP duality and complementary slackness.

Polyhedral Theory

The possibility to interpret a linear program in a geometric way opened the doors to further developments, e.g. the well-known simplex algorithm which currently is the in average best-performing method to solve LPs. The structure of this section mainly follows Nemhauser and Wolsey [134].

Definition 2.1.4. *A* polyhedron *$\mathcal{P} \subseteq \mathbb{R}^n$ is a set of points that satisfy a finite number of linear inequalities, i.e. $\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}\}$ where $\boldsymbol{A}$ is an $m \times n$ matrix and $\boldsymbol{b}$ is vector in $\mathbb{R}^m$.*

The relation to LPs can be easily seen: The set of feasible solutions of an LP defined by its linear constraints corresponds to a polyhedron.

Definition 2.1.5. *A polyhedron $\mathcal{P} \subseteq \mathbb{R}^n$ is bounded if there exists a scalar $\omega \in \mathbb{R}_+$ such that $\mathcal{P} \subseteq \{\boldsymbol{x} \in \mathbb{R}^n : -\omega \leq x_j \leq \omega,\ \forall j \in \{1, ..., n\}\}$. A bounded polyhedron is called* polytope.

The fact that a polyhedron is a convex set plays an important role, especially for the simplex algorithm described later.

Definition 2.1.6. *A set $S \subseteq \mathbb{R}^n$ is a* convex set *if $\boldsymbol{x}, \boldsymbol{y} \in S$ implies that $\lambda\boldsymbol{x} + (1 - \lambda)\boldsymbol{y} \in S,\ \forall\lambda \in [0, 1]$.*

Without loss of generality, we are given an LP

$$\min\{\boldsymbol{c}'\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \in \mathbb{R}^n_+\}. \tag{2.11}$$

Note that as already mentioned before any set of constraints can be represented by an equivalent set of equalities by introducing further variables. We further assume that the LP does not contain redundant equations, i.e. $rank(\boldsymbol{A}) = m \leq n$, by eliminating all linearly dependent rows in matrix $\boldsymbol{A}$. Let $\boldsymbol{a}_j,\ j \in \{1, ..., n\}$, be the j-th column vector of matrix $\boldsymbol{A}$. Then, $\boldsymbol{A}$ contains a non-singular (invertible) sub-matrix $\boldsymbol{A}_B = (\boldsymbol{a}_{B_1}, ..., \boldsymbol{a}_{B_m}) \in \mathbb{R}^{m\times m}$. Let $B = (B_1, ..., B_m)$ and $N = \{1, ..., n\} \setminus B$. By appropriately permuting columns in matrix $\boldsymbol{A}$ we obtain $\boldsymbol{A} = (\boldsymbol{A}_B, \boldsymbol{A}_N)$ such that $\boldsymbol{A}_B\boldsymbol{x}_B + \boldsymbol{A}_N\boldsymbol{x}_N = \boldsymbol{b}$ with $\boldsymbol{x} = (\boldsymbol{x}_B, \boldsymbol{x}_N)$. A solution to LP (2.11) is then given by $\boldsymbol{x}_B = \boldsymbol{A}_B^{-1}\boldsymbol{b}$ and $\boldsymbol{x}_N = 0$.

Definition 2.1.7. *A non-singular matrix $\boldsymbol{A}_B \in \mathbb{R}^{m\times m}$ is called* basis. *Then, $\boldsymbol{x} = (\boldsymbol{x}_B, \boldsymbol{x}_N)$ with $\boldsymbol{x}_B = \boldsymbol{A}_B^{-1}\boldsymbol{b},\ \boldsymbol{x}_N = 0$, is a* basic solution *of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{x}_B$ is the vector of* basic variables *and $\boldsymbol{x}_N$ the vector of* non-basic variables. *If $\boldsymbol{A}_B^{-1}\boldsymbol{b} \geq \boldsymbol{0}$, $(\boldsymbol{x}_B, \boldsymbol{x}_N)$ is called a* basic primal feasible solution *and $\boldsymbol{A}_B$ is called a* primal feasible basis.

To understand the simplex algorithm we discuss the concepts of adjacent basic solutions and degeneracy:

Definition 2.1.8. *Two bases $\boldsymbol{A}_B,\ \boldsymbol{A}_{B'}$ are* adjacent *if only one column is different. If $\boldsymbol{A}_B$ and $\boldsymbol{A}_{B'}$ are adjacent the corresponding two basic solutions are also denoted adjacent.*

Definition 2.1.9. *A primal basic feasible solution $\boldsymbol{x} = (\boldsymbol{x}_B, \boldsymbol{x}_N),\ \boldsymbol{x}_N = \boldsymbol{0}$, is* degenerate *if $\exists i$ with $(\boldsymbol{x}_B)_i = 0$.*

Further definitions are necessary to prove that the set of basic feasible solutions of an LP corresponds to the set of vertices of its associated polyhedron.

Definition 2.1.10. *A polyhedron $\mathcal{P}$ has dimension k if the number of affinely independent points in $\mathcal{P}$ is $k + 1$, i.e. $\dim(\mathcal{P}) = k$.*

Definition 2.1.11. *The inequality $\boldsymbol{a}'\boldsymbol{x} \geq b_j$ is called a* valid inequality *for a set $\mathcal{P}$ if it is satisfied by all points in $\boldsymbol{x} \in \mathcal{P}$.*

Definition 2.1.12. *If $\boldsymbol{a}'\boldsymbol{x} \geq b_j$ is a valid inequality for $\mathcal{P}$ and $F = \{\boldsymbol{x} \in \mathcal{P} \mid \boldsymbol{a}'\boldsymbol{x} = b_j\}$, F is called a* face *of $\mathcal{P}$.*

Definition 2.1.13. *A face F of $\mathcal{P}$ is a* facet *of $\mathcal{P}$ if $\dim(F) = \dim(\mathcal{P}) - 1$.*

Definition 2.1.14. *Let $\mathcal{P}$ be a polyhedron. A vector $\boldsymbol{x} \in \mathcal{P}$ is an* extreme point *of $\mathcal{P}$ if $\nexists \boldsymbol{y}, \boldsymbol{z} \in \mathcal{P},\ \boldsymbol{x} \neq \boldsymbol{y},\ \boldsymbol{x} \neq \boldsymbol{z}$, with a scalar $\lambda \in [0, 1]$, such that $\boldsymbol{x} = \lambda\boldsymbol{y} + (1 - \lambda)\boldsymbol{z}$.*

Note that an extreme point of $\mathcal{P}$ can be seen as a face F with $\dim(F) = 0$.

Corollary 2.1.15. *Each polyhedron has only a finite number of extreme points.*

Definition 2.1.16. *Let $\mathcal{P}$ be a polyhedron. A vector $\boldsymbol{x} \in \mathcal{P}$ is a* vertex *of $\mathcal{P}$ if $\exists \boldsymbol{c} \in \mathbb{R}^n$ such that $\boldsymbol{c}'\boldsymbol{x} \leq \boldsymbol{c}'\boldsymbol{y},\ \forall \boldsymbol{y} \in \mathcal{P},\ \boldsymbol{y} \neq \boldsymbol{x}$.*

Theorem 2.1.17. *Let $\mathcal{P}$ be a non-empty polyhedron and let $\boldsymbol{x} \in \mathcal{P}$. Then, the following statements are equivalent:*

- *Vector $\boldsymbol{x}$ is a vertex.*
- *Vector $\boldsymbol{x}$ is an extreme point.*
- *Vector $\boldsymbol{x}$ is a basic feasible solution.*

Corollary 2.1.15 and Theorem 2.1.17 imply that the number of basic feasible solutions of any LP is finite and due to the following theorem at least one of them is an optimal solution.

Theorem 2.1.18. *We consider an LP minimizing $\boldsymbol{c}'\boldsymbol{x}$ over a set of feasible solutions defined by polyhedron $\mathcal{P}$. Furthermore, we assume that $\mathcal{P}$ contains at least one extreme point and there exists an optimal solution. Then, there exists an optimal solution which is an extreme point of $\mathcal{P}$.*

Theorem 2.1.19. *A non-empty and bounded polyhedron is the convex hull of its extreme points.*

The Simplex Algorithm

The ellipsoid method by Khachiyan [99] or interior point methods introduced by Karmakar [97] are able to solve LPs in polynomial time. In contrast, the simplex algorithm presented by Dantzig [33] decades earlier has exponential runtime in the worst case [19]. However, in practice the simplex method is widely favored due to its far higher performance on average. Thus, we will focus on this LP algorithm, here, presenting the main ingredients and argumentation. Further details of the simplex method can be found in [19].

In principle, the simplex algorithm starts from an arbitrary vertex of the polyhedron associated to an LP and iteratively moves to adjacent vertices with lower objective function values (in case of minimization problems). If there is no adjacent vertex with better objective value then the current vertex represents an optimal solution. This stopping criterion only works because of the convexity of the polyhedron since then a local optimum equals a global optimum.

To return to the context of solution vectors, the move from one basic feasible solution to an adjacent one is called *pivoting step*: Beginning with solution $\boldsymbol{x} = (\boldsymbol{x}_B, \boldsymbol{x}_N)$ exactly one basic

variable $x_i \in \boldsymbol{x}_B$ leaves the basis and non-basic variable $x_j \in \boldsymbol{x}_N$ enters it, cf. Definition 2.1.8. The decision which variable is replaced by which one bases on the *reduced costs* $\bar{c}_j$ of variables $x_j \in \boldsymbol{x}$:

Definition 2.1.20. *Let $\boldsymbol{x}$ be a basic feasible solution, $\boldsymbol{A}_B$ its associated basis matrix, and $\mathbf{c}_B$ the cost vector of basic variables. The* reduced cost $\bar{c}_j$ *of variable $x_j,\ j \in \{1, ..., n\}$, is defined as $\bar{c}_j = c_j - \boldsymbol{c}'_B \boldsymbol{A}_B^{-1} \boldsymbol{a}_j$.*

The reduced costs vector $\bar{\mathbf{c}}_B$ of basic variables is obviously the zero vector since $\bar{\boldsymbol{c}}'_B = \boldsymbol{c}'_B - \boldsymbol{c}'_B \boldsymbol{A}_B^{-1} \boldsymbol{A}_B = \mathbf{0}$. The following theorem defines optimality conditions for a basic feasible solution:

Theorem 2.1.21. *Let $\bar{\mathbf{c}}$ be the reduced costs vector of a basic feasible solution $\boldsymbol{x}$ and its associated basis matrix $\boldsymbol{A}_B$.*

- *If $\bar{c}_j \geq 0,\ \forall j \in \{1, ..., n\}$, then $\boldsymbol{x}$ is optimal.*
- *If $\boldsymbol{x}$ is optimal and non-degenerate, then $\bar{c}_j \geq 0,\ \forall j \in \{1, ..., n\}$.*

Provided the current basic feasible solution $\boldsymbol{x}$ is non-degenerate, after the next pivoting step bringing in a non-basic variable with negative reduced costs we obtain a new solution with lower objective value. Thus, in case of non-degeneracy and according to Corollary 2.1.15 the simplex algorithm terminates after a finite number of iterations.

According to Theorem 2.1.21 in an optimal but degenerated solution some variables may have negative reduced costs. Thus, in case of degeneracy as per Definition 2.1.9 situations can arise in which a pivoting step does not change the solution and the simplex possibly runs into a cycle. To guarantee termination cycling can be prevented by using special *pivoting rules*, e.g. the lexicographic ordering or the smallest subscript rule (Bland's rule).

To summarize, given an initial basic feasible solution the simplex method obtains an optimal solution within a finite number of iterations. However, sometimes an initial feasible solution may not be trivial to find. In these cases the so-called two-phase simplex algorithm is applied: By solving an additional linear program introducing appropriate artificial variables at first we are able to find a basic feasible solution if one exists. Equipped with this solution we can proceed with the usual simplex on the original LP to search for an optimal solution.

Finally, we want to mention the so-called *dual simplex* which is executed in the same way but on the dual LP instead of the primal LP. Because of strong duality we know that the optimal solutions of P and D are equivalent. However, in some cases it might be beneficial to work on the dual LP: Consider an LP and a corresponding optimal solution. By adding further constraints – as it is the case for branch-and-bound (Section 2.1.3) and cutting plane methods (Section 2.1.4) – the previously primal feasible solution may become infeasible. Thus, the *primal simplex algorithm* again has to find a basic feasible solution to start from. In contrast, adding a row in the primal LP corresponds to adding a column (or variable) in the dual LP. Since the rest of the dual LP does not change, the previous solution stays dual feasible by simply setting the newly added variable to zero. The dual simplex can now "hot-start" and it may be expected to need less pivoting steps to obtain a new optimal solution than the primal simplex which has to start from scratch.

2.1.2 Integer Linear Programming

According to the general definition 1.1.1 of COPs one has to choose a feasible subset of a given set of elements which yields the best objective value. The decision whether an element is selected or not usually is modeled by a binary variable instead of a continuous one since intermediate states do not make sense. More generally, if the base set contains several equivalent elements we may be interested to decide how many elements of a particular type should be chosen. Thus, we could use non-negative integer variables to model these problems. More formally, an integer linear program (IP) is defined as

$$z = \min \quad \boldsymbol{c}'\boldsymbol{x} \tag{2.12}$$

$$\text{subject to} \quad \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \tag{2.13}$$

$$\boldsymbol{x} \in \mathbb{Z}^n_+. \tag{2.14}$$

Matrix $\boldsymbol{A}$ and vectors $\boldsymbol{c}$ and $\boldsymbol{b}$ are defined exactly the same as for the LP (2.1)–(2.3). From a polyhedral point of view, the set of feasible solutions $X = \mathcal{P} \cap \mathbb{Z}^n_+$ is the intersection of polyhedron $\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A}x \geq \boldsymbol{b}\}$ with the integer space $\mathbb{Z}^n_+$. Usually, the set of variables within an IP is not restricted to integer ones: In practical applications we often have both continuous and integer variables to model a problem resulting in a so-called mixed integer linear program (MIP).

Definition 2.1.22 (LP relaxation)**.** *Given is an IP* (2.12)–(2.14)*. If we replace the integrality constraints* (2.14) *by* (2.3) *we obtain the* linear programming relaxation *of IP.*

Since constraints (2.3) are weaker restrictions on the set of feasible solutions than (2.14) all feasible solutions of IP are also feasible for the corresponding LP relaxation, and the optimal solution of the LP relaxation provides a lower bound to the optimal integer solution of IP, i.e. $z_{\text{LP}} \leq z$.

Theorem 2.1.23. *Let $\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^n_+ \mid \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}\}$, where $\boldsymbol{A} \in R^{m \times n}$ is a rational matrix and $\boldsymbol{b} \in \mathbb{R}^m$ a rational vector, and $X = \mathcal{P} \cap \mathbb{Z}^n_+$. Then the convex hull of the set of integral vectors X denoted as $conv(X)$ is a rational polyhedron.*

This theorem shows us a way to solve IP: We determine the convex hull of set X of feasible solutions of IP and solve the LP $\min\{\boldsymbol{c}'\boldsymbol{x} \mid \boldsymbol{x} \in conv(X)\}$. The obtained solution corresponds to a vertex on the convex hull of X and thus represents a feasible and also optimal solution to IP. However, the convex hull in general may consist of an exponential number of constraints and thus it usually is not possible to describe it in polynomial time. In contrast to LPs, solving IPs is in general $\mathcal{NP}$-hard. Usually we are able to perform "intelligent" enumeration using sophisticated rules to prune subsets of feasible solutions but in worst case we have to examine all of them.

Comparing Formulations

In principle, an unlimited number of feasible IP formulations exist for a set of integral solutions X which all describe a polyhedron whose intersection with space $\mathbb{Z}^n_+$ results in set X. However,

some of them may provide a "tighter" description of the convex hull of X than others, i.e. the optimal solution value to the corresponding LP relaxation is "nearer" to the optimal integer value.

Sometimes, different IPs for the same problem may involve different sets of variables. Thus, comparing these formulations in terms of their corresponding polyhedra seems to be not obvious. In fact, we need to first project the different polyhedra on a common subspace usually defined by the variables which directly correspond to the set of elements in the problem description.

Definition 2.1.24. *Let $\mathcal{P} = \{(\boldsymbol{x}, \boldsymbol{y}) \mid \boldsymbol{D}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} \geq \boldsymbol{d}\}$ be a polyhedron. The* projection *of $\mathcal{P}$ on the set of variables $\boldsymbol{x}$ is defined as $proj_{\boldsymbol{x}}(\mathcal{P}) = \{\boldsymbol{x} \mid \exists \boldsymbol{y},\ (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{P}\}$.*

Now we are able to compare the polyhedra associated to different IP formulations:

Definition 2.1.25. *Given are two integer programming formulations P and P' with associated polyhedra $\mathcal{P}$ and $\mathcal{P}'$, respectively. Assume that variables $\boldsymbol{x}$ are used in both P and P'. Then P* dominates *P' if $proj_{\boldsymbol{x}}(\mathcal{P}) \subseteq proj_{\boldsymbol{x}}(\mathcal{P}')$ and* strictly dominates *P' if $proj_{\boldsymbol{x}}(\mathcal{P}) \subset proj_{\boldsymbol{x}}(\mathcal{P}')$.*

Also in cases where two formulations P and P' do not have a common set of variables the concept of dominance is applicable: If each feasible solution of P can be transformed to a feasible solution of P' then P dominates P'. On the other hand, if there is no solution mapping from P' to P, then P strictly dominates P'.

2.1.3 LP-based Branch-and-Bound

This section discusses the most common approach for solving IPs and the presentation mainly follows Wolsey [190]. In principle, branch-and-bound is a divide and conquer approach which first breaks the problem into smaller and easier problems, then solves these smaller problems, and finally puts obtained information together to determine a solution to the original problem.

Proposition 2.1.26. *We are given a problem $z = \min\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in X\}$. Let $X = X_1 \cup ... \cup X_K$ be a decomposition of X into smaller not necessarily disjoint sets, and let $z^k = \min\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in X_k\},\ \forall k \in \{1, ..., K\}$. Then $z = \min_k z^k$.*

In LP-based branch-and-bound approaches a problem is usually split into two subproblems which is called *binary branching* but also a decomposition into more than two subproblems is possible (*multi-way branching*).

Such a divide and conquer method can be represented by an enumeration tree also called *branch-and-bound tree*. For example, we could enumerate all possible values for one particular integer variable of an IP model on one level of the tree and for each subtree fix this variable to the corresponding value. However, complete enumeration is usually computationally impossible for more than 20 variables for many problems. So, how can we benefit from the information obtained in one node to possibly prune complete subtrees without explicitly enumerating them?

Proposition 2.1.27. *Let $X = X_1 \cup ... \cup X_K$ be a decomposition of X into smaller sets and $z^k = \min\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in X_k\},\ \forall k \in \{1, ..., K\}$. Let $\overline{z}^k$ be an upper bound and $\underline{z}^k$ be a lower bound on z^k. Then $\overline{z} = \min_k \overline{z}^k$ is an upper bound and $\underline{z} = \min_k \underline{z}^k$ a lower bound on z.*

Lower bounds for minimization problems can be obtained by solving relaxations of a problem, e.g. the LP relaxation from Section 2.1.2, Lagrangian relaxation [50], or problem-specific combinatorial relaxations, whereas upper bounds are usually provided by heuristics yielding feasible solutions.

In general, subtrees are pruned in three ways:

1. If a subproblem $z^k = \min\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in X_k\}$ can be solved to optimality then we do not need to further split up X_k and thus can prune the current subtree.

2. If a lower bound of a subproblem is at least as high as the best-known upper bound, i.e. $\underline{z}^k \geq \overline{z}$, then clearly there cannot be a better solution in the current subtree and thus it can be pruned.

3. If the set of feasible solutions of a subproblem is empty, i.e. $X_k = \emptyset$, this branch can also be pruned.

If none of these rules can be applied to a node of the branch-and-bound tree then the subproblem has to be further split up.

The LP-based branch-and-bound is frequently used in practice to solve IPs since it is generally applicable without need for problem-specific adaptations. As its name suggests it uses the LP relaxation to compute lower bounds (for minimization problems) and performs binary branching in the following way: Let $\boldsymbol{x}_{\mathrm{LP}}$ be the optimal solution to the LP relaxation on set X. If all integer variables have integral values then the obtained lower bound is also an upper bound and thus the current subproblem is solved. On the other hand, if $\boldsymbol{x}_{\mathrm{LP}}$ contains some fractional values for integer variables, i.e. $x_j^{\mathrm{LP}} \notin \mathbb{Z}$ for some j, then set X is split into

$$X_1 = X \cap \{\boldsymbol{x} \mid x_j \leq \lfloor x_j^{\mathrm{LP}} \rfloor\} \qquad \text{and} \qquad X_2 = X \cap \{\boldsymbol{x} \mid x_j \geq \lceil x_j^{\mathrm{LP}} \rceil\}. \tag{2.15}$$

By applying this branching we can be sure that $X_1 \cup X_2 = X,\ X_1 \cap X_2 = \emptyset,\ \boldsymbol{x}_{\mathrm{LP}} \notin LP(X_1),\ \boldsymbol{x}_{\mathrm{LP}} \notin LP(X_2)$. Thus, the combined lower bound $\min\{\underline{z}^1, \underline{z}^2\} \geq \underline{z}$ monotonically increases. After adding the constraints created by branching to the models of the corresponding subproblems we select one of them and resolve the corresponding LP by using the dual simplex algorithm, see Section 2.1.1. The complete LP-based branch-and-bound algorithm is shown in Alg. 2.1.

If there are more than one integer variables with fractional values in an LP solution there has to be chosen one of them to branch on: We could pick a random one, the "most fractional" one, apply *strong branching* or special branching for generalized upper bounds, etc. Several branching strategies are discussed e.g. in [3].

Finally, we also have to make a decision which subproblem from the list of open problems to examine next: Again we may select a random one or apply more sophisticated strategies, e.g. *depth-first-search* or *best-node-first*. Since strong primal bounds are important for pruning subtrees, in the depth-first-search strategy we prioritize to go down the branch-and-bound tree to quickly find primal bounds, which is especially meaningful if we have no or weak heuristics. On the contrary, best-node-first chooses the subproblem with the smallest lower bound $\underline{z}^i = \min_k \underline{z}^k$ to minimize the total number of examined problems since here we can be sure to never split up

Algorithm 2.1: LP-based Branch-and-Bound

Input: IP $\min\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in X\}$
Output: Optimal solution $\boldsymbol{x}^*$

problem list $L : \min\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in X\}$
$\overline{z} = \infty$, incumbent $\boldsymbol{x}^* = \mathbf{0}$
while $L \neq \emptyset$ **do**
    choose set X_i and remove it from L
    solve $\underline{z}^i = LP(X_i)$
    $\boldsymbol{x}^i_{\mathrm{LP}}$ ... optimal LP solution
    if $X_i = \emptyset$ **then** prune X_i by infeasibility
    else if $\underline{z}^i \geq \overline{z}$ **then** prune X_i by bound
    else if $\boldsymbol{x}^i_{\mathrm{LP}} \in X$ **then**
        if $z^i \leq \overline{z}$ **then**
            update primal bound $\overline{z} = z^i$
            update incumbent $\boldsymbol{x}^* = \boldsymbol{x}^i_{\mathrm{LP}}$
        prune X_i by optimality
    else $L = L \cup \{X_{i,1}, X_{i,2}\}$
return $\boldsymbol{x}^*$

a subproblem with $\underline{z}^k > z$ which would be pruned later. In practice a combination of several strategies is typically used, see [20, 134] for a more detailed discussion.

2.1.4 Cutting Planes and Branch-and-Cut

Sometimes a feasible IP model for a problem contains an exponential number of constraints. Clearly, explicitly formulating all of them for a given instance and then solving the complete model usually makes no sense with respect to computability. Additionally, possibly not all of these constraints are needed to describe the polyhedron associated to the corresponding LP relaxation. Thus, we need a method to identify a reasonable subset of these constraints and only add this set to the model while ensuring feasibility of the finally obtained solution.

In other situations we may have a compact formulation for a problem, i.e. with a polynomial number of variables and constraints, which however provides a rather weak LP relaxation bound. Thus, the corresponding branch-and-bound tree might become quite large during the LP-based branch-and-bound process. The concept of *valid inequalities* described in Definition 2.1.11 provides a way to strengthen our LP bounds by adding further constraints which are valid for the convex hull of integer solutions but cut off parts of the LP relaxation polyhedron.

The so-called *cutting plane methods* first solve the LP relaxation of a usually small or incomplete model, then try to find at least one valid inequality which is violated by the current solution with respect to the set of feasible solutions to the given problem, add the new constraint(s) to the model, and resolve it. These steps are repeated until we obtain a feasible and thus optimal solution for our problem.

Definition 2.1.28. *Given an IP* $\min\{\boldsymbol{c}'\boldsymbol{x} : \boldsymbol{x} \in X\}$ *and a solution* $\hat{\boldsymbol{x}} \in \mathbb{R}^n_+$ *with* $\hat{\boldsymbol{x}} \notin conv(X)$*, the* separation problem *is to find a valid inequality* $\boldsymbol{a}'\boldsymbol{x} \geq b_j$ *which is violated by* $\hat{\boldsymbol{x}}$*.*

Generally speaking, cutting plane methods try to obtain the description of the convex hull of the set X of feasible solutions. However, since $conv(X)$ itself may consist of an exponential number of inequalities, it may be a good idea to terminate the cutting plane method at some point. Also if we only separate some particular sets of valid inequalities we may not be able to identify a violated valid inequality in some iteration and thus end up with a fractional solution.

Therefore, cutting plane methods are usually embedded in a branch-and-bound system resulting in a so-called branch-and-cut algorithm: After adding several cutting planes in a branch-and-bound node in case of a still fractional solution the usual branching takes place resulting in further subproblems to be examined. This quite general approach proved to be extremely successful for numerous problems in literature since it often dramatically reduces the size of the branch-and-bound tree.

2.1.5 Column Generation and Branch-and-Price

In contrast to cutting plane approaches column generation and branch-and-price provide an efficient way to solve formulations with exponential numbers of variables by adding them dynamically similarly to valid inequalities before. Such models arise e.g. when applying a reformulation technique called Dantzig-Wolfe decomposition [35] to improve the dual bound obtained by the LP relaxation.

In these situations delayed column generation is applied initially including only a small subset of all variables and iteratively adding further variables identified by the so-called *pricing subproblem*. This idea has been introduced by Gilmore et al. [57, 58] in 1961 and was used to solve numerous problems during the following decades, see e.g. [39] for a detailed description and survey.

We denote the following LP the (linear) master problem (MP):

$$\min \quad \sum_{j \in J} c_j x_j \tag{2.16}$$

$$\text{subject to} \quad \sum_{j \in J} \boldsymbol{A}_j x_j \geq \boldsymbol{b} \tag{2.17}$$

$$x_j \geq 0 \qquad \forall j \in J. \tag{2.18}$$

Sometimes, the set J of variables and thus the corresponding MP may be extremely large, e.g. exponentially-sized, which usually makes it intractable to solve directly. The approach is now to define a restricted master problem (RMP)

$$\min \quad \sum_{j \in \tilde{J}} c_j x_j \tag{2.19}$$

$$\text{subject to} \quad \sum_{j \in \tilde{J}} \boldsymbol{A}_j x_j \geq \boldsymbol{b} \tag{2.20}$$

$$x_j \geq 0 \qquad \forall j \in \tilde{J}, \tag{2.21}$$

where $\tilde{J} \subset J$ is a small subset of the original set of variables.

Let $\boldsymbol{u} \geq \mathbf{0}$ be the vector of corresponding dual variable values. Theorem 2.1.21 tells us that a variable $x_j,\ j \in J \setminus \tilde{J}$, with negative reduced costs

$$\bar{c}_j = c_j - \boldsymbol{u}' \boldsymbol{A}_j < 0, \tag{2.22}$$

is able to decrease the objective value. In the pricing subproblem we need to either identify a not yet included variable with negative reduced costs and add it to the model or prove that no one exists. If at least one new variable has been added, we resolve the LP relaxation and then the pricing subproblem. This procedure is repeated until there is no variable $x_j,\ j \in J \setminus \tilde{J}$, with negative reduced costs left.

However, computational problems within column generation may arise in practice: Vanderbeck [178] describes several issues leading to possibly poor performance, e.g. adding irrelevant variables in the beginning (*heading-in effect*), primal simplex degeneracy (*plateau effect*), and slow convergence (*tailing-off effect*). So-called *stabilization* approaches try to partly avoid these issues [122].

Similarly to branch-and-cut in the previous section, column generation can be embedded in a branch-and-bound system to solve the LP relaxation in each branch-and-bound node in order to finally obtain a proven optimal solution. In this case it is usually beneficial to explicitly consider special branching rules: It can be easily seen that branching on one of the dynamically added variables possibly leads to extremely asymmetric subtrees often resulting in a huge number of examined branch-and-bound nodes.

2.2 Heuristic Methods

Whenever exact algorithms are not applicable to a COP because of too strict time and/or memory requirements heuristic approaches come into play. This is often the case for $\mathcal{NP}$-hard problems on large-scale instances. Here, usually a problem-specific algorithm tries to find a solution for a given problem with "good" objective value. In most cases no proof of optimality and no information about the quality of a solution with respect to the optimal objective value can be provided. Thus, even if a heuristic actually finds an optimal solution for an instance it usually does not know about it and may continue its search for improvements. However, the fundamental advantage of heuristics is the in general much higher performance in terms of runtime and memory consumption making it possible to frequently obtain excellent results for large-sized instances of a COP. We structure the remainder of this part in the following way: Section 2.2.1 discusses heuristics which just aim at constructing one feasible solution, Section 2.2.2 is devoted to approximation algorithms which are able to provide a worst-case bound on the quality of a solution with respect to the optimal value, Section 2.2.3 briefly describes local search methods, and Section 2.2.4 discusses the large class of metaheuristics.

2.2.1 Construction Heuristics

The aim of construction heuristics is to find a solution to a COP by iteratively adding elements to a partial solution until a complete, feasible solution is obtained. In most cases already made

decisions in earlier iterations are not allowed to be withdrawn in later steps. The criteria on which the selection of elements is based are often greedy-like which means that always the one of the remaining candidates is chosen which does not violate feasibility and is the best choice from the current point of view with respect to some cost function. However, the simplicity and the "one-way-property" of these greedy heuristics may produce solutions which can be far from being optimal [23].

2.2.2 Approximation Algorithms

Most of the heuristic methods including metaheuristics in general have no tools to evaluate the quality of a solution with respect to the optimal objective value. However, one kind of heuristics – so-called approximation algorithms [92] – provide worst-case guarantees in the sense that a computed solution is not farther away from an optimal solution than a given absolute or relative bound. The reader is referred to the books by Kellerer et al. [98] and Vazirani [179] for extensive discussions on this topic.

We are given some instance I of a minimization problem. Let $c^*(I)$ and $c^A(I)$ be the objective values of an optimal solution and a solution obtained by some polynomial-time algorithm A, respectively.

Definition 2.2.1. *An algorithm A is an approximation algorithm with* absolute performance guarantee $k > 0$*, if $c^A(I) - c^*(I) \leq k$ for all instances I of a given minimization problem.*

Definition 2.2.2. *An algorithm A is an approximation algorithm with* relative performance guarantee $k > 1$*, if $\frac{c^A(I)}{c^*(I)} \leq k$ for all instances I of a given minimization problem.*

Definition 2.2.3. *APX is the class of polynomial-time approximation algorithms with constant performance guarantee.*

If the relative performance guarantee of an algorithm is not fixed to a value k but can be chosen arbitrarily close to the optimal value at the cost of runtime, then A is called an ϵ-approximation scheme.

Definition 2.2.4. *An algorithm A is an* ϵ-approximation scheme *if for every input $\epsilon \in (0, 1)$, $\frac{c^A(I)}{c^*(I)} \leq 1 + \epsilon$ holds for all instances I of a given minimization problem.*

ϵ-approximation schemes are further differentiated by their runtime with respect to the instance size and ϵ: If the runtime of A is polynomial in the instance size it is called a *polynomial-time approximation scheme* (PTAS). Even more restrictive (and useful in practice) are the so-called *fully polynomial time approximation schemes* (FPTAS) which require the runtime to be also polynomial in $\frac{1}{\epsilon}$.

2.2.3 Local Search

A so-called *local search* [1, 22, 138] requires a feasible starting solution $x \in X$ and then tries to find a solution x' in a pre-defined neighborhood of x which is at least as good as x. If such a solution x' can be found x is replaced by this new one and the neighborhood search restarts.

Algorithm 2.2: Local Search

Input: Feasible solution $x \in X$ for a minimization problem
Output: Possibly improved feasible solution x

```
while stopping criteria not met do
    choose x' ∈ N(x)
    if c(x') ≤ c(x) then
        x = x'
return x
```

Definition 2.2.5. *A* neighborhood structure *is a function $N : X \to 2^X$ which assigns a set of neighbors $N(x) \subseteq X$ to each feasible solution $x \in X$.*

Usually, neighborhoods are defined by feasible moves modifying small parts of a solution, e.g. replacing one edge in a solution to a graph problem.

Definition 2.2.6. *We are given an instance of a minimization problem, a corresponding feasible solution $x \in X$ with objective value $c(x)$, and a neighborhood structure N. Then, x is called a* local optimum *with respect to N iff $c(x) \leq c(x'),\ \forall x' \in N(x)$.*

Definition 2.2.7. *A global optimum to an optimization problem is a local optimum with respect to all possible neighborhood structures.*

The neighboring solution $x' \in N(x)$ commonly is selected in one of the following ways:

- Random selection: We simply pick a random element of $N(x)$ without considering objective values.
- Next improvement: We enumerate the solutions in $N(x)$ and select the first one which is better than x.
- Best improvement: We go through all elements of $N(x)$ and choose the best one.

If we perform next or best improvement a natural stopping criterion is when no improved solution could be obtained anymore, i.e. a local optimum has been reached. However, also different stopping criteria, e.g. time or iteration limits, are frequently used in practice. Algorithm 2.2 shows this basic local search while more successful extensions of this simple approach are discussed in the next section.

2.2.4 Metaheuristics

A more general concept are the so-called *metaheuristics* [53,59,182] which describe a problem-independent iterative guiding process controlling a set of subordinate mostly problem-specific heuristics. Applying metaheuristics to optimization problems has become quite popular in recent times especially in the last ten years. One of the reasons for this trend is that in many cases metaheuristics are able to obtain high-quality solutions while the implementation can often be

done with moderate effort: The main task is to provide the problem-specific subordinate operators and heuristics which are often quite simple-structured algorithms. The famous holistic saying can also be applied to metaheuristics: The combination of these simple components is more than the sum of its parts.

The secret of a metaheuristic's success usually lies in finding a balance between diversification, i.e. exploring unknown areas in solution space, and intensification, i.e. focusing on the close surrounding of promising solutions. It is also important to find the right time to switch between these two modes depending on the current progress: If the search stagnates it may be beneficial to move to new regions of the solution space. On the other hand, if new best solutions are found in recent iterations then it may be better to further concentrate on their neighborhoods.

The development of new metaheuristics has recently become an upcoming trend, possibly caused by the result of Wolpert et al. [189] who have shown with their *no free lunch theorems* that there is no (meta-)heuristic which performs better than another one with respect to all problem classes.

An extensive overview of the most widespread metaheuristics can be found in the books [53, 59]. Here, we only discuss the variants used in our approaches in the following chapters.

Multilevel Refinement

In [184] Walshaw brings the nature of this metaheuristic to the point:

> "The multilevel paradigm as applied to combinatorial optimisation problems is a simple one, which at its most basic involves recursive coarsening to create a hierarchy of approximations to the original problem. An initial solution is found, usually at the coarsest level, and then iteratively refined at each level, coarsest to finest, typically by using some kind of heuristic optimisation algorithm (either a problem-specific local search scheme or a metaheuristic). Solution extension (or projection) operators can transfer the solution from one level to another."

Algorithm 2.3 shows the general structure of a multilevel refinement heuristic. Additional information and exemplary applications to COPs can be found in [183].

The detailed procedures for coarsening, obtaining the solution at the coarsest level, and refinement are in most cases problem-specific and can hardly be generalized. If considering for example problems which can be represented on graphs possible coarsening steps could be to merge adjacent nodes or edges by applying some greedy criterion. An important decision to be made when designing a reasonable coarsening scheme is how much information should be preserved on the transition from one level to the next. If discarding too many details of a problem instance the solution obtained on the coarsest level may be highly infeasible for the original instance and thus has to be repaired in the refinement phase with possibly high computational overhead. On the other hand, if too much information is preserved the multilevel principle might not be able to unfold its power by allowing a more abstract view on the problem catching the essential elements of an instance.

In contrast to this solution construction an iterated multilevel refinement approach starts with a given feasible solution and tries to exploit the solution structure within the coarsening phase. Thus, different starting solutions result in different coarsening steps.

Algorithm 2.3: Multilevel Refinement Heuristic

Input: Instance I_0 of an optimization problem
Output: Feasible solution x_0

$l = 0$
while *stopping criteria not met* **do**
    $I_{l+1} = coarsen(I_l)$
    $l = l + 1$
$x_l = solve(I_l)$
while $l > 0$ **do**
    $l = l - 1$
    $x_l = extend(x_{l+1}, I_l)$
    $x_l = refine(x_l, I_l)$
return x_0

Algorithm 2.4: Greedy Randomized Adaptive Search Procedure

Input: Instance of a minimization problem with element set S
Output: Feasible solution x

$x = \emptyset$
while *stopping criteria not met* **do**
    candidate list $CL = S$
    $x' = \emptyset$
    while x' *is not a complete solution* **do**
        build restricted candidate list RCL from CL
        randomly select an element x_i from RCL
        $x' = x' \cup \{x_i\}$
        $CL = CL \setminus \{x_i\}$
    (locally) improve x'
    if $x == \emptyset \vee c(x') < c(x)$ **then**
        $x = x'$
return x

Greedy Randomized Adaptive Search Procedure

The greedy randomized adaptive search procedure (GRASP) has been introduced by Feo et al. [47, 48] and provides a way to extend the simple greedy construction principle by use of randomness and local search. While iteratively constructing a solution a greedy approach would always choose the best element with respect to the used criterion. Here, we usually restrict the remaining set of elements to a smaller subset containing the best candidates and then randomly choose one of them. The size of this restricted candidate list usually is controlled by some parameter.

Algorithm 2.5: Variable Neighborhood Descent

Input: Feasible solution x to a minimization problem
Output: Feasible solution x

```
l = 1
while l ≤ l_max do
    find a neighbor x' ∈ N_l(x) by next- or best-improvement
    if c(x') < c(x) then
        x = x'
        l = 1
    else
        l = l + 1
return x
```

After finishing the solution construction Feo et al. suggest a subsequent improvement phase, e.g. a local search as described in Section 2.2.3. These steps, i.e. the construction and improvement, are repeated as long as some stopping criteria are not met and the overall best solution is finally returned. Algorithm 2.4 sketches the GRASP approach. A major disadvantage of this basic variant of GRASP is the fact that it cannot learn from past iterations. It always starts from scratch without utilizing already collected information about the problem instance. Ribeiro et al. [157] extend the basic GRASP in a way that the solution construction profits from past iterations by choosing elements which have been part of good solutions with higher probabilities.

Variable Neighborhood Search

The concept of variable neighborhood search (VNS) [78–80] is based on the fact that a local optimum with respect to one particular neighborhood structure is not necessarily a local optimum with respect to another neighborhood structure.

A VNS variant called variable neighborhood descent (VND) utilizes this property and systematically switches between several neighborhood structures to finally obtain a solution which is locally optimal with respect to all considered neighborhoods. Algorithm 2.5 shows the VND.

Choosing appropriate neighborhoods and defining the order of examination is still a difficult problem-specific task. However, at least for finding successful neighborhood orderings different quite general concepts have been presented recently: Puchinger et al. [148] suggest a dynamic approach by using neighborhood relaxations to a priori evaluate the success rate of applying a particular neighborhood and then obtain an appropriate ordering. Hu et al. [88] propose a so-called self-adaptive VND which determines a promising ordering depending on the successes and runtimes of neighborhoods in past iterations.

Since VND only provides mechanisms for intensification, the general VNS (GVNS) additionally incorporates a set of usually larger neighborhoods for diversification and randomly "shakes" solutions within these neighborhoods to escape from local optima. After shaking a VND is applied to intensify the solution again. The GVNS is based on the principle that differ-

Algorithm 2.6: General Variable Neighborhood Search

Input: Feasible solution x to a minimization problem
Output: Feasible solution x

```
while stopping criteria not met do
    k = 1
    while k ≤ k_max do
        randomly select x' ∈ N_k(x) // diversification
        x' = VND(x') // intensification
        if c(x') < c(x) then
            x = x'
            k = 1
        else
            k = k + 1
return x
```

ent starting solutions for a VND result in different local optima which clearly is only the case for multi-modal solution spaces. The general structure of a GVNS can be seen in Algorithm 2.6.

Genetic Algorithms

Holland [85] introduced the term *genetic algorithms* (GA) for an in the meanwhile broad class of heuristic frameworks inspired by nature and especially by the concept of evolution described by Darwin [36] and Mendel [127]. Basically, a GA works on a set of solutions represented by strings of some alphabet and denoted as *population of chromosomes*, and iteratively applies the common evolutionary processes of selection, recombination, and mutation on it.

The representation (genotype) of a solution (phenotype) within a GA heavily influences the performance: A direct representation where each element of the instance's base set corresponds to a decision bit within a vector may not be the best choice if either the base set is quite large resulting in long strings or the set of feasible solutions is rather small in relation to the set of possible bit strings. Thus, usually a more sophisticated (favorably bijective) mapping function converts a feasible solution to a chromosome and the other way round. In the best case each possible chromosome corresponds to a feasible solution. In the simplest forms of GAs this mapping function is the only problem-specific part in an implementation.

Different operators for selection (e.g. randomized, fitness proportional, or tournament selection), recombination (e.g. one-point or uniform crossover), and mutation are imaginable which can either be general or in rare cases problem-specific. Algorithm 2.7 shows a quite general GA template [155]. Further information about GAs can be found in numerous articles and books, e.g. [128, 155, 188].

A promising extension also for many other metaheuristics is the embedding of local search methods to improve individual solutions by exploiting the particular problem structure. In case of GAs this concept is called memetic algorithm [130, 131] where usually both the initial so-

Algorithm 2.7: Genetic Algorithm

Input: Instance of a COP
Output: Feasible solution x

```
P ... initial set of chromosomes
while stopping criteria not met do
    O ... empty set of offsprings
    while offsprings O not sufficient do
        if crossover condition satisfied then
            select parent chromosomes P' from P
            select crossover parameter
            o = crossover(P')
        if mutation condition satisfied then
            select mutation parameters
            o = mutate(o)
        evaluate fitness of offspring o
        O = O ∪ {o}
    P = select(P, O)
return best solution x ∈ P
```

lutions and new offsprings are locally improved before continuing the GA. In contrast to basic GAs, these more complex memetic algorithms frequently belong to the leading state-of-the-art algorithms for many COPs.

Ant Colony Optimization

Similar to genetic algorithms ant colony optimization is inspired by nature [43]:

> "Ant Colony Optimization (ACO) is a metaheuristic that is inspired by the pheromone trail laying and following behavior of some ant species. Artificial ants in ACO are stochastic solution construction procedures that build candidate solutions for the problem instance under concern by exploiting (artificial) pheromone information that is adapted based on the ants' search experience and possibly available heuristic information."

Algorithm 2.8 describes ant colony optimization in a rather generic way: In every iteration each "ant" step by step constructs a solution with respect to probabilities defined by the currently available pheromones and usually other heuristic information e.g. greedy criteria. After that, a solution optionally is improved by some local search methods or other metaheuristics. Among the final solutions the best ones are chosen to "deposit" pheromones according to the solutions' properties. In the end of an iteration a small amount of each pheromone value "evaporates". The pheromones can be seen as memory incorporating the information of all solutions obtained in past iterations.

Algorithm 2.8: Ant Colony Optimization

```
Input: Instance of a minimization problem
Output: Feasible solution x
initialize pheromones
while stopping criteria not met do
    forall the ants do
        construct solution x' based on pheromone and heuristic information
        optionally improve solution x'
        if c(x') < c(x) then
            x = x'
    deposit pheromones of best ants
    evaporate pheromones
return x
```

One of the most successful extensions to the original ant system from Dorigo et al. [42] is the $\mathcal{MAX} - \mathcal{MIN}$ ant system (MMAS) by Stützle et al. [176] which includes the following features: The pheromone values are limited by predefined lower and upper bounds preventing a stagnation of the search if some values nearly reach one and others converge to zero. Additionally, the pheromones are initially set to the upper bounds focusing on diversification at the beginning of the search. Furthermore, MMAS follows an elitist strategy which only allows the best ant of an iteration or even the globally best ant to deposit pheromones.

2.3 Hybrid Methods

As already mentioned, both exact and heuristic methods have their advantages and disadvantages. Exact methods provide proven optimal solutions but only if they have the time and memory to reach the state of optimality. Especially for large-scale problem instances sometimes no useful information in terms of bounds or feasible solutions may be obtained within reasonable time. On the other hand, heuristic approaches are frequently good at quickly providing feasible solutions but usually without information about the corresponding quality w.r.t. an optimal solution. Thus, it seems to be obvious to combine the benefits of both approaches to hybrid methods, see [21, 125, 154] for detailed overviews.

There are two main criteria for classifying different hybrids: By the type of used methods and by the way they are combined. *Collaborative combinations* denote hybrids where the participating methods stay quite independent but communicate with each other to share obtained information. This can be done by executing the methods sequentially, in parallel, or in an intertwined way, see [151,153] for further details. On the contrary, *integrative combinations* describe hybridizations where one method calls another e.g. to solve some subproblem.

Considering the type of used methods, we can distinguish between combining different metaheuristics and heuristic and exact approaches. We already mentioned some heuristic combinations in previous section, e.g. embedding local search methods in other metaheuristics,

see [21, 154] for a comprehensive survey on this topic.

In the second case we can once again differentiate between metaheuristic hybrids which use exact algorithms to solve some subproblems to improve the solution quality, and exact approaches which utilize (meta-)heuristics to accelerate the solution process by providing stronger primal bounds. For the promising hybridizations of metaheuristics and mathematical programming techniques even a special term – *matheuristics* – came up, see [125, 149] for an overview and applications.

For example when searching large-scale possibly exponentially-sized neighborhood structures [4] a simple enumeration of all neighboring solutions is not reasonable anymore. Thus, it makes sense to search for the best neighbor by sophisticated exact methods being able to prune parts of the search space, e.g. [30, 87, 144]. On the other hand, the heuristic solution of $\mathcal{NP}$-hard separation problems within cutting plane algorithms [74] and difficult pricing subproblems in column generation [141, 147] proved to be a successful approaches in literature.

CHAPTER 3

Rooted Delay-Constrained Minimum Spanning Tree Problem

This chapter discusses several heuristic approaches for solving the *rooted delay-constrained minimum spanning tree (RDCMST) problem*. Section 3.1 defines the considered problem and Section 3.2 mentions previous related work. After describing preprocessing techniques to reduce the size of input graphs in Section 3.3, we present two construction heuristics based on Kruskal's minimum spanning tree algorithm in Section 3.4 and the multilevel refinement paradigm in Section 3.5, respectively. The first heuristic is embedded in a GRASP in Section 3.6 combined with a variable neighborhood descent described in Section 3.8 based on two efficient neighborhood structures from Section 3.7. These neighborhoods are further utilized in a general variable neighborhood search in Section 3.9, an ant colony optimization approach in Section 3.10, and a memetic algorithm in Section 3.11. How duplicates in our genetic algorithm can be tackled is discussed in Section 3.12. Experimental results in Section 3.13 show the potential of our reduction techniques and compare the proposed heuristic methods. Finally, Section 3.14 mentions open problems and possible future research directions. The content mainly bases on the published articles [15, 161, 162, 164].

3.1 Problem Definition

First, we formally define the RDCMST problem. We are given an undirected graph $G = (V, E)$ with node set V, edge set E, a cost function $c : E \to \mathbb{Z}_0^+$, and a delay function $d : E \to \mathbb{Z}^+$ assigning cost and delay values to all edges, respectively. Furthermore, a fixed root node $s \in V$ and a delay-bound $B \in \mathbb{Z}^+$ is given. A feasible solution to the RDCMST problem is a spanning tree $T = (V, E'),\ E' \subseteq E$, satisfying the delay-constraints

$$d_v^T = \sum_{e \in P_T(s,v)} d_e \leq B,\ \forall v \in V \setminus \{s\}. \tag{3.1}$$

$P_T(s,v)$ denotes the unique path from the specified root node s to node $v \in V \setminus \{s\}$ in the spanning tree T and d_v^T the corresponding total delay on this path. Further, we define the cost function

$$c_T = \sum_{e \in E'} c_e, \tag{3.2}$$

summing up the cost values of all edges in a solution T. An optimal solution T^* to the RDCMST problem is a feasible solution with minimal total edge costs, i.e. $c_{T^*} \leq c_T,\ \forall T$.

The limitation to integer cost and delay values does not restrict the set of input graphs in practice since rational values can be transformed to integers by scaling and irrational values do not occur in most real world applications and usually cannot be handled efficiently in computer programs. Furthermore, cost and delay values do not have to be Euclidian, not even in practice. In some situations it may be cheaper to make a detour since the direct way may contain some obstacles which are expensive to overcome. If considering transportation networks the delay on the direct way may also take longer than alternative paths because of different carriers.

Additionally, we define a directed variant of this problem on graph $G' = (V, A)$ with arc set $A = \{(s,v) : \{s,v\} \in E\} \cup \{(u,v),(v,u) : \{u,v\} \in E,\ u,v \neq s\}$ consisting of two opposite arcs for each edge in graph G except for edges incident to root node s, for which we include only the corresponding arc going out from s. A feasible solution to the directed variant is an arborescence $T_s = (V, A'),\ A' \subset A$, directed out of root node s. It can be easily seen that each feasible spanning tree T bijectively corresponds to a feasible arborescence T_s.

The RDCMST problem is $\mathcal{NP}$-hard because a special case called *hop-constrained minimum spanning tree (HCMST) problem*, where $d_e = 1,\ \forall e \in E$, is shown to be $\mathcal{NP}$-hard in [32], so all more general variants of this problem are $\mathcal{NP}$-hard, too.

A trivial lower bound to the optimal cost value is provided by a minimum spanning tree T^{l} with respect to the edge costs and without considering the delay values at all. If such a tree T^{l} is feasible for the RDCMST problem, i.e. it satisfies the delay-constraints, then T^{l} is an optimal solution. Further, we are able to construct a trivial feasible spanning tree T^{u} defined by the shortest-delay-paths from root s to all nodes $v \in V$ without considering edge costs. T^{u} can be computed e.g. by Dijkstra's polynomial time algorithm for the single-source shortest path problem [41]. Finally, T^{u} helps to decide if a feasible solution to the RDCMST problem exists or not because if even the shortest-delay-path exceeds the specified delay-bound for some node then there is no feasible tree.

Furthermore, instead of using $c_{\{u,v\}}$ and $d_{\{u,v\}}$ to denote cost and delay values assigned to edge $\{u,v\} \in E$, we use the more readable notation c_{uv} and d_{uv}, respectively. The same holds for arcs $(u,v) \in A$ in directed graph G'. Variable $d_v,\ v \in V$, refers to the node delay with respect to one specific tree T. In case of multiple solutions the considered tree is explicitely included in the notation, i.e. $d_v^T,\ v \in V$.

3.2 Related Work

Salama et al. [166, 167] introduced the RDCMST problem and proved its $\mathcal{NP}$-hardness by reduction from the *exact cover by 3-sets problem* [52]. Furthermore, they presented the first heuristic approach, a construction method based on Prim's algorithm to find a minimum span-

ning tree [145]. This Prim-based heuristic starts from the root node and iteratively connects a node which can be reached in the cheapest way without violating the delay-constraint. If at some point no node can be connected anymore, the delays in the existing tree are reduced by replacing edges. These steps are repeated until a feasible RDCMST is obtained. A second phase improves the solution by local search using the edge-exchange neighborhood structure.

Manyem et al. [126] showed that the problem is not in APX, the class of $\mathcal{NP}$-hard problems for which there exist polynomial-time algorithms with constant approximation factor.

Exact approaches to the RDCMST problem have been examined by Gouveia et al. in [68] based on the concept of constrained shortest paths utilized in a MIP formulation. The LP relaxation of this formulation is solved by unstabilized delayed column generation and Lagrangian relaxation methods. To tackle the subproblem of finding constrained shortest paths the authors use an efficient dynamic programming approach extending the basic method by Dumitrescu et al. [46]. The column generation method suffers from degeneracy issues, cf. [178], and therefore is in most cases outperformed by the Lagrangian relaxation approach dualizing the constraints linking path and edge variables combined with an efficient primal heuristic. Similarly to [66], a third approach reformulates the constrained shortest path subproblem for each node on a layered graph and solves them using a multi commodity flow (MCF) formulation. Each layer in this extended graph corresponds to a specific path delay from the root node. Original nodes are then duplicated on each layer modeling the visit of a node exactly at the corresponding path delay. Each edge in G is copied in a similar way skipping a number of layers that corresponds to the edge's delay. Thus, delays are implicitly encoded in the layered structure and therefore do not have to be considered explicitly anymore in a solution approach. In Section 4.8 this transformation is described in detail. Obviously, the size of the layered graph and therefore the efficiency of the according model heavily depend on the number of achievable delay values. Hence, this approach can in practice only be used for instances with a reasonably small set of delay values and rather low bounds. Additionally, MCF models usually suffer from the huge amount of flow variables used in the MIP formulation altogether leading to a slow and memory-intensive solving process. Nevertheless solving these layered graph models turned out to be very effective on certain classes of instances, see Section 4.9, not only for RDCMST problems, but e.g. for the *hop-constrained connected facility location problem* as well, see [118].

To the best of the author's knowledge there are so far no other publications dedicated to the RDCMST problem specifically. However, many recent publications exist for the *rooted delay-constrained Steiner tree (RDCST) problem*, see Chapter 4, which is a generalization of the RDCMST problem. In this variant only a subset of the nodes has to be reached within the given delay-bound, the other nodes can optionally be used as intermediate (Steiner) nodes. In principle most methods for the RDCST problem can also be applied to the RDCMST problem but often are rather inefficient since they usually provide special considerations of the optional nodes in the input graph which do not exist here. Salama et al. [166] compared their Prim-based heuristic to two successful heuristics for the RDCST problem [104, 196], see Section 4.2, and showed that their approach outperforms the other methods on most of the rather small instance graphs with 20 to 200 nodes. Therefore, it makes sense to tackle the RDCMST problem explicitly using methods exploiting the spanning tree structure of this problem.

The *bounded-diameter minimum spanning tree (BDMST) problem* is another related problem

variant generalizing the HCMST problem. Here, we aim for a spanning tree with minimum cost in which the number of edges between any pair of nodes is limited by a given diameter. Much work has been done for this problem: Gruber et al. [75] propose several neighborhood structures embedded in a variable neighborhood search, an ant colony optimization approach, and a genetic algorithm. Furthermore, the same authors present in [73] a sophisticated construction heuristic based on hierarchical clustering, and in [74] a hybrid approach using (meta-)heuristics to solve a separation problem arising in an integer programming formulation based on so-called jump cuts. All these methods are also described in more detail in [72]. Finally, Gouveia et al. [70] present an efficient branch-and-cut method working on a layered graph.

3.3 Preprocessing

Applying preprocessing techniques to reduce the input graph is highly important for most of the solution approaches since these techniques are able to dramatically reduce the search space and therefore the runtime of a solution method. First, we clearly can remove edges that cannot be part of a feasible solution. Second, edges can be discarded that may not be part of all optimal solutions. Additional to the graph reductions presented in the rest of this section, we could apply preprocessing rules based on reduced costs, e.g. as Leggieri et al. proposed in [111] for the RDCST problem. However, for successfully adopting these methods we need strong lower and upper bounds to the optimal costs which in turn have to be obtained by some dedicated approaches. Therefore, these kind of techniques are more commonly used in exact approaches where both bounds are available and tightened throughout the solution process.

3.3.1 Infeasible Edges

In the following cases an edge $\{u, v\} \in E$ cannot be part of a feasible solution so discarding it safely reduces the search space.

- Obviously if delay d_{uv} is larger than the bound B edge $\{u, v\}$ can be discarded immediately.
- Edges $\{u, v\} \in E$ that would exceed the bound in all possible trees can also be removed from the graph [111] if satisfying these conditions:

$$d_u^{\min} + d_{uv} > B \quad \wedge \quad d_v^{\min} + d_{uv} > B, \tag{3.3}$$

  whereas minimum delays $d_v^{\min} := \min_{P(s,v)} \sum_{e \in P} d_e,\ \forall v \in V$, are calculated a priori by Dijkstra's shortest path algorithm [41] applied on the delays and starting from root s. If we consider an arc (u, v) in the corresponding directed graph G' only the first condition has to hold to discard this arc.

Both preprocessing tests can be done on all edges E in $\mathcal{O}(|E| + |V| \log |V|)$ time including the calculation of $d^{\min}$ values if using an efficient implementation of Dijkstra's algorithm based on Fibonacci heaps [51].

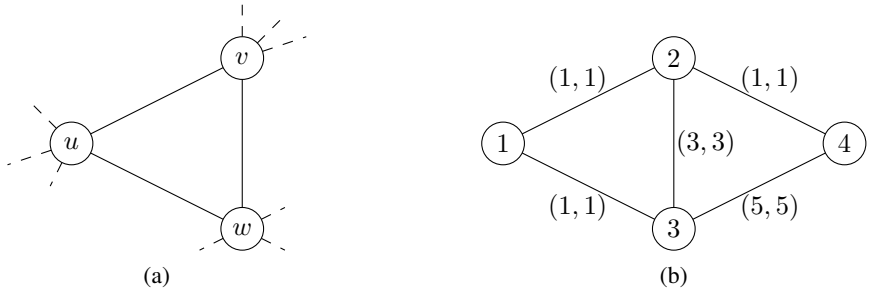


Figure 3.1: (a) Subgraph of input graph $G = (V, E)$ forming a triangle. (b) Example for the importance of edge examination order. Edge labels (c_e, d_e) denote corresponding cost and delay values.

3.3.2 Suboptimal Edges

Suboptimal edges can be part of a feasible solution but may not appear in an optimal solution, so discarding them safely prunes the solution space. Since we just aim to find one optimal solution and not the complete set of optimal solutions we additionally are allowed to remove an edge which is part of an optimal solution as long as there is another optimal solution which does not include this edge.

Comparison to root edges

Applying the arc elimination test presented in [65] for the HCMST problem to the RDCMST problem results in the removal of edge $\{u, v\} \in E$ if edges $\{s, i\}$ and $\{s, j\}$ exist and the following conditions hold:

$$c_{sv} \leq c_{uv} \quad \wedge \quad d_{sv} \leq d_u^{\min} + d_{uv} \quad \wedge \quad c_{su} \leq c_{uv} \quad \wedge \quad d_{su} \leq d_v^{\min} + d_{uv} \tag{3.4}$$

This preprocessing rule can be helpful for rather dense or complete graphs but in sparse graphs only few edges are typically discarded, mainly because of the small out-degree of the root node. Searching those edges takes $\mathcal{O}(|E|)$ time if values $d_v^{\min},\ v \in V$, are already known. Similar to infeasible arcs, if we consider an arc (u, v) in the corresponding directed graph G' only the first two conditions have to hold to discard this arc.

Extension to arbitrary triangles

The following two preprocessing methods are related to the special distance test for the classical Steiner tree problem in graphs, cf. [100].

Theorem 3.3.1. *We consider a triangle consisting of edges $\{u, v\}, \{u, w\}, \{v, w\}$ in graph $G = (V, E)$, see Fig. 3.1. If*

$$c_{uv} \geq c_{uw} + c_{vw} \quad \wedge \quad d_{uv} \geq d_{uw} + d_{vw}, \tag{3.5}$$

then edge $\{u, v\}$ may appear in an optimal solution $T^* = (V, E^*)$ but there is at least one other optimal solution with $\{u, v\} \notin E^*$. Therefore, we can remove edge $\{u, v\}$ from graph G.

Proof. We prove this theorem by contradiction and assume that edge $\{u, v\}$ satisfying the conditions above is part of any optimal solution T^*. Without loss of generality, node u is nearer to the root s in tree T^*, i.e. $d_u < d_v$. According to the definition of a feasible solution to the RDCMST problem, $d_v = d_u + d_{uv} \leq B$. All three edges of the triangle clearly cannot be part of a solution since a cycle would violate the tree property. Next, we distinguish three cases and show that in any of these cases we are able to replace edge $\{u, v\}$ by another edge without increasing the tree costs.

1. $\{u, v\}, \{u, w\} \in E^*,\ \{v, w\} \notin E^*$: Again, we consider two different cases:

   a) $d_w < d_u \Rightarrow d_v = d_w + d_{wu} + d_{uv}$

   b) $d_w > d_u \Rightarrow d_v = d_u + d_{uv}$

   If we define a tree T' by replacing edge $\{u, v\}$ by $\{v, w\}$, d_u, d_w stay the same and d_v may change. By applying the delay condition $d_{uv} \geq d_{uw} + d_{vw}$ we obtain:

   a) $d'_v = d_w + d_{wv} \leq d_v$

   b) $d'_v = d_u + d_{uw} + d_{wv} \leq d_v$

   So all node delays $d_v,\ v \in V$, either remain unchanged or are decreased, maintaining solution feasibility. Applying the cost condition $c_{uv} \geq c_{uw} + c_{vw}$ we obtain new tree costs $c_{T'} = c_{T^*} - c_{uv} + c_{vw} \leq c_{T^*}$. This contradicts the assumption that there is no optimal tree without edge $\{u, v\}$. ↯

2. $\{u, v\}, \{v, w\} \in E^*,\ \{u, w\} \notin E^*$:

   a) $d_w < d_v$: Since $d_u < d_v$ the solution does not form a feasible tree because of two different paths from root s to node v. ↯

   b) $d_w > d_v$: $d_w = d_u + d_{uv} + d_{vw}$. Replacing $\{u, v\}$ by $\{u, w\}$ leaves d_u unchanged and leads to:

   - $d'_v = d_u + d_{uw} + d_{wv} \leq d_v$
   - $d'_w = d_u + d_{uw} \leq d_w$

   $\Rightarrow c_{T'} = c_{T^*} - c_{uv} + c_{uw} \leq c_{T^*}$. ↯

3. $\{u, v\} \in E^*,\ \{u, w\}, \{v, w\} \notin E^*$: node w is not directly connected to u or v, so we discuss the following two cases:

   a) $d_w \leq d_v - d_{vw}$: replacing $\{u, v\}$ by $\{v, w\}$ leaves d_u, d_w unchanged and $d'_v = d_w + d_{wv} \leq d_v \Rightarrow c_{T'} = c_{T^*} - c_{uv} + c_{vw} \leq c_{T^*}$. ↯

   b) $d_w > d_v - d_{vw}$: let $\{p, w\} \in E^*$ be the unique edge with $d_p < d_w$. We replace $\{u, v\}, \{p, w\}$ by $\{u, w\}, \{v, w\}$, leading to an unchanged d_u and:

   - $d'_v = d_u + d_{uw} + d_{wv} \leq d_v$
   - $d'_w = d_u + d_{uw} = d_v - d_{uv} + d_{uw} \leq d_v - d_{vw} < d_w$

   $\Rightarrow c_{T'} = c_{T^*} - c_{uv} - c_{pw} + c_{uw} + c_{wv} \leq c_{T^*}$. ↯

All cases lead to a contradiction of the assumption that edge $\{u, v\}$ has to be in any optimal solution. Therefore, we can safely discard this edge and be sure that there is still at least one optimal solution left. □ □

Considering all triangles including one specific edge $\{u, v\}$ takes $\mathcal{O}(|V|)$ time so the whole preprocessing test can be done in $\mathcal{O}(|E| \cdot |V|)$ time. When iterating over all edges it is important not to immediately discard found suboptimal edges since this would reduce the set of triangles for incident edges examined later and therefore would possibly prevent a removal, see Fig. 3.1b for an example: Examining edge $\{2, 3\}$ first would result in a removal because of edges $\{1, 2\}$ and $\{1, 3\}$. However, then no more triangles exist in this graph. Otherwise, if we first examine edge $\{3, 4\}$ then finally this one and $\{2, 3\}$ can be discarded. Therefore, if edges are first sorted by decreasing costs then we are not faced with this problem and can safely remove an edge at the time of discovery. In this case we examine the most expensive edge of each triangle before the two other ones, and this is the only one which potentially can be discarded within this triangle.

To further analyze the possible practical impact of this preprocessing rule we consider a complete graph $G = (V, E)$ with $n = |V|$ nodes and random real-valued edge costs and delays uniformly distributed in $[0, 1]$. The density function of the sum of two uniformly distributed random variables in $[0, 1]$ [180] is given by

$$f(x) = \begin{cases} x & : \quad 0 \leq x \leq 1 \\ -x + 2 & : \quad 1 < x \leq 2 \end{cases} . \tag{3.6}$$

Then, the probability that the cost condition in Theorem 3.3.1 holds for edge $\{u, v\}$ with cost $c_{uv} \in [0, 1]$ and delay $d_{uv} \in [0, 1]$ with respect to one triangle is

$$\Pr(C_{uw} + C_{wv} \leq c_{uv}) = \int_0^{c_{uv}} f(x)\, dx = \int_0^{c_{uv}} x\, dx = \frac{{c_{uv}}^2}{2}, \tag{3.7}$$

so the probability of discarding edge $\{u, v\}$ with respect to one triangle is

$$\Pr(C_{uw} + C_{wv} \leq c_{uv} \wedge D_{uw} + D_{wv} \leq d_{uv}) = \frac{{c_{uv}}^2}{2} \cdot \frac{{d_{uv}}^2}{2} = \frac{(c_{uv} d_{uv})^2}{4}. \tag{3.8}$$

The expected value for the removal of edge $\{u, v\}$ over all possible cost and delay values in one triangle is further given by

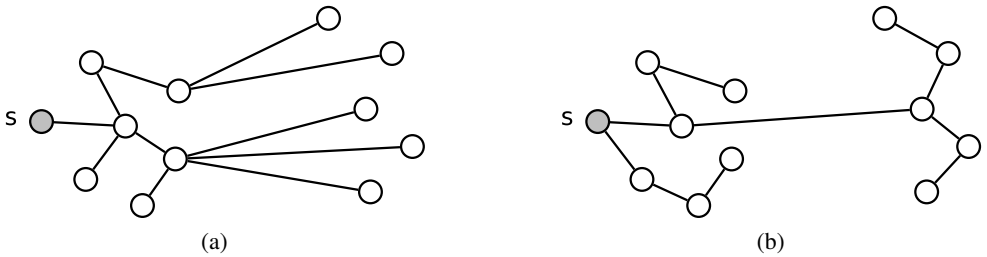
$$\mathbb{E} = \iint_0^1 \frac{(xy)^2}{4}\, dx\, dy = \frac{1}{36}. \tag{3.9}$$

In a complete graph there are exactly $n - 2$ triangles including edge $\{u, v\}$, so the overall probability of discarding this edge in a complete graph is

$$\Pr(\{u, v\} \text{ discarded}) = 1 - \Pr(\{u, v\} \text{ not discarded}) = 1 - (1 - \frac{(c_{uv} d_{uv})^2}{4})^{n-2}. \tag{3.10}$$

Again, the expected value for the removal of edge $\{u, v\}$ over all possible cost and delay values and triangles can be determined by

$$\mathbb{E} = \iint_0^1 1 - (1 - \frac{(xy)^2}{4})^{n-2}\, dx\, dy, \tag{3.11}$$



(a) (b)

Figure 3.2: Prim-based heuristic (a) compared to Kruskal-based heuristic (b).

which cannot be simplified to an explicit function of n in a trivial way. However, the percentage of expected edge removals for specific values of n can be numerically approximated, e.g. $\mathbb{E} \approx 0.54$ for $n = 100$, $\mathbb{E} \approx 0.73$ for $n = 500$, $\mathbb{E} \approx 0.79$ for $n = 1000$, and $\mathbb{E} \approx 0.89$ for $n = 5000$, resulting in a significant reduction of the edge set in complete graphs with random cost and delay values. We can easily see: The more nodes in a complete graph, the more triangles exist, and the higher the probability of an edge to be discarded.

Extension to arbitrary paths

The previous preprocessing rule can be generalized in the following way: Edge $\{u, v\}$ may appear in an optimal solution but there is at least one optimal solution without this edge if there exists a path $P(u, v) \subseteq E \setminus \{\{u, v\}\}$ satisfying the following conditions:

$$c_{uv} \geq \sum_{e' \in P(u,v)} c_{e'} \quad \wedge \quad d_{uv} \geq \sum_{e' \in P(u,v)} d_{e'} \tag{3.12}$$

Since the proof of Theorem 3.3.1 can be extended in a rather trivial way to show the feasibility of this preprocessing rule, we skip it here.

Applying this preprocessing test reduces to solving the resource-constrained shortest path problem. This problem is $\mathcal{NP}$-hard in the weak sense, cf. [52], and approximable by an FPTAS which implies the existence of an exact pseudo-polynomial algorithm. Recently, algorithms based on dynamic programming have been described for solving practical instances of this problem quite efficiently, see e.g. [46] for a general approach and [68] for a variant customized to the RDCMST problem by Gouveia et al. Nevertheless, finding a delay-constrained shortest path runs in time $\mathcal{O}(|E| \cdot B)$ which extends to $\mathcal{O}(|E|^2 \cdot B)$ for the complete preprocessing test.

Especially the last two preprocessing steps must be used with caution. When having a limited runtime (as in some of our tests in Section 3.13) the preprocessing phase could dominate or even use up the whole time for large instances. So preprocessing can also be counterproductive in practice.

Algorithm 3.1: Kruskal-Based Heuristic

Input: graph $G = (V, E)$
Output: feasible solution $T = (V, E')$ to the RDCMST problem

`// Stage 1: merge components`

L_E ... list of edges E sorted by ascending costs
$C = \{C_1, ..., C_{|V|}\}$ ... set of components initally consisting of single nodes
$\delta_v = d_v^{\min},\ p_v = v,\ \delta_v^{\max} = 0,\ v_{C_v} = v,\ \forall v \in V$
while $|C| > 1 \wedge L_E \neq \emptyset$ **do**
    remove first edge $e = \{u, v\}$ from L_E
    if $C_u \neq C_v$ **then**
        $\Delta_u = B - (\delta_u + d_{uv} + \delta_v^{\max})$
        $\Delta_v = B - (\delta_v + d_{vu} + \delta_u^{\max})$
        if $\Delta_u \geq 0 \vee \Delta_v \geq 0$ **then**
            $E' = E' \cup \{e\}$
            $C_{u+v} = C_u \cup C_v$
            if $\Delta_u \geq \Delta_v$ **then** $v_{C_{u+v}} = v_{C_u}$
            else $v_{C_{u+v}} = v_{C_v}$
            update $\delta_w,\ p_w,\ \delta_w^{\max},\ \forall w \in C_{u+v}$

`// Stage 2: repair by adding shortest-delay-paths`

if $|C| > 1$ **then**
    forall the $C_i \in C \setminus \{C_s\}$ **do**
        $P(s, v_{C_i})$ ... shortest-delay-path from s to v_{C_i}
        $u \in V$ ... latest node on $P(s, v_{C_i})$ with $\delta_u = d_u^{\min}$
        forall the $\{v, w\} \in P(u, v_{C_i})$ **do** `//` $d_v^{\min} < d_w^{\min}$
            if $p_w \neq w, v$ **then** $E' = E' \setminus \{\{p_w, w\}\} \cup \{\{v, w\}\}$

return T

3.4 Kruskal-Based Construction Heuristic

A general problem of the Prim-based heuristic by Salama et al. [166], summarized in Section 3.2, especially on Euclidian instances is the fact that the nodes in the close surrounding of the root node are typically connected rather cheaply, but at the same time delay is "wasted", and many distant nodes can later only be linked by rather expensive edges, see Fig. 3.2. The stricter the delay-bound the more this drawback will affect the costs negatively. This fact led us to a more de-centralized approach by applying the basic concept of Kruskal's minimum spanning tree algorithm [107] to the RDCMST problem, see Algorithm 3.1.

3.4.1 Stage 1: Merging components

In the beginning of stage one of the construction heuristic all edges are sorted by ascending costs and then iteratively added to the solution preventing cycles until a feasible spanning tree is

formed. In other words, components initially consisting of single nodes are merged by adding edges to result in one connected tree. The challenge is to maintain the feasibility of the partial solutions, i.e. to satisfy the delay-constraint to the root node throughout the whole merging process. In the Prim-based approach in [166] checking the feasibility of adding an edge to the existing tree naturally runs in constant time whereas our de-centralized algorithm needs more effort to achieve this. We have to store and update additional information for each node $v \in V$:

- the delay δ_v on the path from s to v
- the maximum delay $\delta_v^{\max}$ to any other node in the same component
- the predecessor p_v on the path $P(s, v)$, initialized with node v

To initialize δ_v Dijkstra's algorithm [41] calculates the path $P(s, v),\ \forall\ v \in V$, with the shortest path delay. The paths themselves are not added to the solution, we just keep them stored as a backup to always have a possible feasible connection to the root node available. These fallback paths will become essential for stage two of the heuristic.

Initially we have a set of components $C = \{C_1, ..., C_k\},\ k = |V|$. Every time we add an edge to the solution two components are merged and thereby k is decreased by 1 until set C only contains one component. For each component $C_i,\ i = 1, ..., k$, we specify one node v_{C_i} which is assumed to be nearest to the root node – it can be seen as the local root node of the subtree C_i. As mentioned above the path $P(s, v_{C_i}),\ v_{C_i} \neq s$, is not part of the tree, we just use it for testing the feasibility of a partial solution.

Now we start iterating over the sorted edge-list. Let $e = \{u, v\} \in E$ be the next edge on the list and $C_u \ni u,\ C_v \ni v$ be the components incident to e, respectively. Clearly, if $C_u = C_v$, edge e would create a cycle and thus can be skipped. If $C_u \neq C_v$, the decision of adding e to the tree and thereby merging the two components C_u and C_v is based upon fulfilling at least one of the following two conditions:

$$\delta_u + d_{uv} + \delta_v^{\max} \leq B \tag{3.13}$$

$$\delta_v + d_{vu} + \delta_u^{\max} \leq B \tag{3.14}$$

So if it is allowed to add edge e to the solution in this sense, the node information of all nodes in the newly created component $C_{u+v} = C_u \cup C_v$ has to be updated. First of all we have to specify the new $v_{C_{u+v}}$. There are many possibilities of choosing this node with the only constraint that $\delta_{v_{C_{u+v}}}$ plus the delay of path $P(v_{C_{u+v}}, w)$ has to satisfy the delay-bound for all $w \in C_{u+v}$. A very simple and fast method turned out to be the most successful one: if only condition (3.13) is met then $v_{C_{u+v}} = v_{C_u}$, when only condition (3.14) holds, we choose v_{C_v}, and if both conditions are satisfied we prefer the v_{C_i} where the corresponding inequality has a larger gap to the delay-bound.

Beginning from this chosen local root node for component C_{u+v} we perform a depth-first search to update p_w and $\delta_w,\ \forall\ w \in C_{u+v}$, using $\delta_{v_{C_{u+v}}}$ as the starting delay. The maximal extents $\delta_w^{\max}$ can be determined in linear time profiting from the tree structure of the component: Basically we perform a depth-first-search from an arbitrary node in the subtree to obtain the delay-heights of all nodes. In a second depth-first-search we use these heights to finally calculate the maximal extents.

The iterations stop if the solution only consists of one component, which at the same time means that it is feasible, or there are more than one components but no more edges left in the list. The latter case is handled in stage two.

To conclude, stage one consists of sorting all edges of the graph in $\mathcal{O}(|E| \log |E|)$ time, testing each one for feasibility in constant time and updating the node information in $\mathcal{O}(|V|)$ time if an edge is added which can happen at most $|V| - 1$ times due to the properties of a tree. So the total runtime is in $\mathcal{O}(|E| \log |E| + |V|^2)$.

3.4.2 Stage 2: Extension to a feasible solution

At the end of stage one the graph is not necessarily connected, so in stage two the remaining subtrees are attached to the component which contains the root node by adding the shortest-delay-path $P(s, v_{C_i}),\ \forall\ C_i \in C,\ C_i \neq C_s$. At least one of the edges of a path $P(s, v_{C_i})$ creates a cycle when adding it to the solution, otherwise all edges of $P(s, v_{C_i})$ would have been included in stage one. Consequently, the main task in this stage is to dissolve resulting cycles to form a tree without violating the delay-constraint.

Paths are added by backtracking the shortest-delay-path starting from node v_{C_i} until a node u with minimal delay δ_u is reached. We can be sure that path $P(s, u)$ is already the shortest-delay-path and do not have to go further – in the worst case, however, we end up at the root node. Now we add the missing edges along path $P(u, v_{C_i})$ until we are back at v_{C_i}. Cycles can occur if edge $e = \{v, w\}$ is added and $p_w \neq w, v$, indicating that two different paths $P(s, w)$ exist. Removing edge $\{p_w, w\}$ dissolves this cycle and at the same time maintains feasibility because the delay δ of any node in component C_w can only get smaller or stay equal since δ_w now is the smallest possible delay and all other nodes depend on that. In C_{p_w} no delays are affected by the removal of edge $\{p_w, w\}$ since all nodes are connected to the root node through path $P(s, v_{C_{p_w}})$.

Since the dissolving of cycles can be done in constant time and each node is examined at most once, stage two runs in $\mathcal{O}(|V|)$.

3.4.3 Example

Figure 3.3 shows a detailed example of applying the Kruskal-based heuristic, see Algorithm 3.1, on instance graph G from Fig. 3.3a. Edges included in the solution and the components' representatives $v_{C_i},\ \forall C_i \in C$, are colored blue. The shortest delays to nodes $v \in V$ are: $d_s^{\min} = 0,\ d_1^{\min} = 2,\ d_2^{\min} = 2,\ d_3^{\min} = 1,\ d_4^{\min} = 4,\ d_5^{\min} = 2$. Only relevant changes in variable values will be mentioned in this example description. After sorting the edge set firstly by ascending costs, secondly by ascending delays, and thirdly by ascending node indices, stage one examines the edges in the following order:

1. $\{2, 3\}$ (Fig. 3.3b): $\Delta_2 = 1,\ \Delta_3 = 2 \Rightarrow E' = \{\{2, 3\}\},\ v_{C_{2+3}} = 3,\ \delta_2 = 2,\ \delta_3 = 1,\ p_2 = p_3 = 3,\ \delta_2^{\max} = \delta_3^{\max} = 1$.

2. $\{3, 5\}$ (Fig. 3.3c): $\Delta_3 = 2,\ \Delta_5 = 0 \Rightarrow E' = \{\{2, 3\}, \{3, 5\}\},\ v_{C_{3+5}} = 3,\ \delta_2 = \delta_5 = 2,\ \delta_3 = 1,\ p_2 = p_3 = p_5 = 3,\ \delta_2^{\max} = \delta_5^{\max} = 2,\ \delta_3^{\max} = 1$.

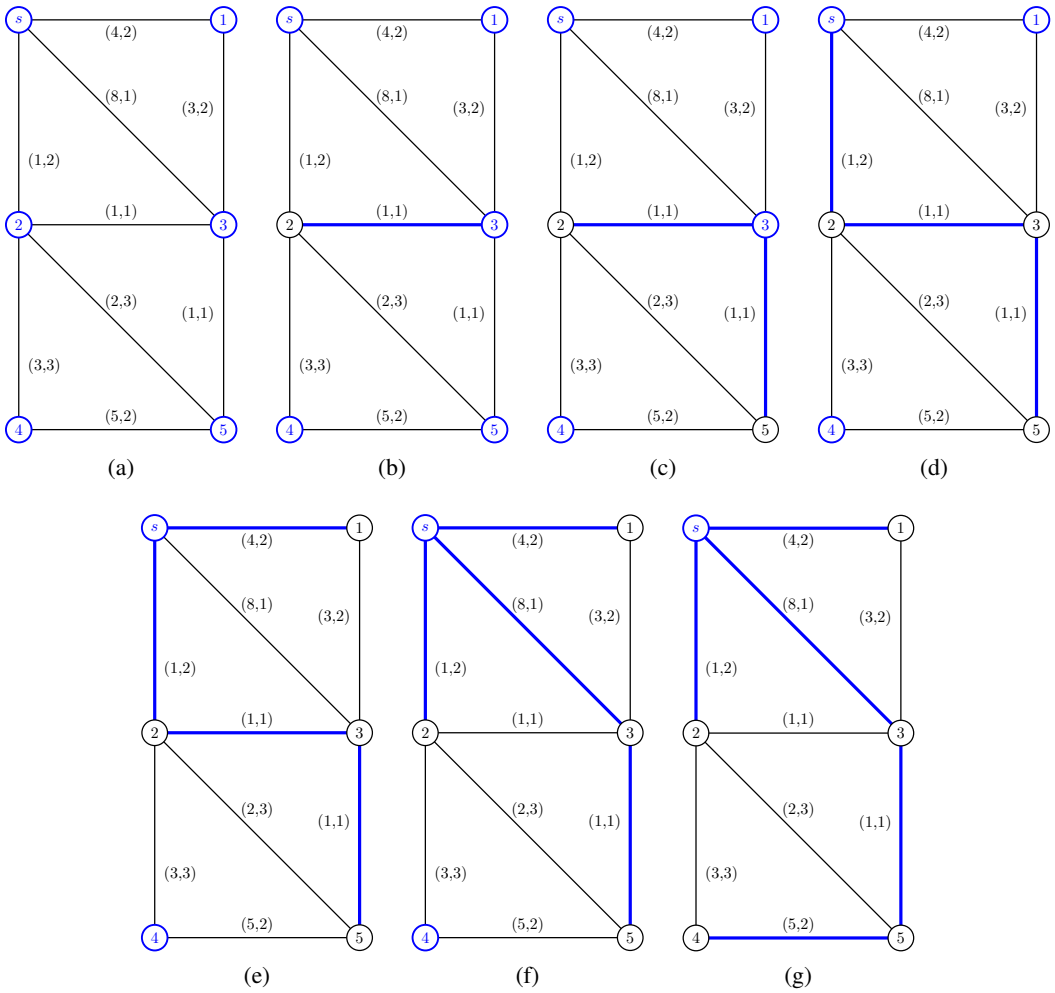


Figure 3.3: Application of Kruskal-based heuristic on input graph in (a) with $B = 4$: Stage one in (b)–(e) and stage 2 in (f)–(g).

3. $\{s, 2\}$ (Fig. 3.3d): $\Delta_s = 0,\ \Delta_2 = 0 \Rightarrow E' = \{\{2, 3\}, \{3, 5\}, \{s, 2\}\},\ v_{C_{s+2}} = s,\ \delta_s = 0,\ \delta_2 = 2,\ \delta_3 = 3,\ \delta_5 = 4,\ p_s = s,\ p_2 = s,\ p_3 = 2,\ p_5 = 3,\ \delta_s^{\max} = 4,\ \delta_2^{\max} = 2,\ \delta_3^{\max} = 3,\ \delta_5^{\max} = 4.$

4. $\{2, 5\}$: creates a cycle $\Rightarrow$ skip.

5. $\{1, 3\}$: $\Delta_1 = -3,\ \Delta_3 = -1 \Rightarrow$ skip.

6. $\{2, 4\}$: $\Delta_2 = -1,\ \Delta_4 = -5 \Rightarrow$ skip.

7. $\{s, 1\}$: (Fig. 3.3e): $\Delta_s = 2,\ \Delta_1 = -4 \Rightarrow E' = \{\{2, 3\}, \{3, 5\}, \{s, 2\}, \{s, 1\}\},\ v_{C_{s+1}} = s,\ \delta_s = 0,\ \delta_1 = \delta_2 = 2,\ \delta_3 = 3,\ \delta_5 = 4,\ p_s = p_1 = p_2 = s,\ p_3 = 2,\ p_5 = 3,\ \delta_s^{\max} = \delta_2^{\max} = 4,\ \delta_1^{\max} = \delta_5^{\max} = 6,\ \delta_3^{\max} = 5.$

8. $\{4, 5\}$: $\Delta_4 = -8,\ \Delta_5 = -2 \Rightarrow$ skip.

9. $\{s, 3\}$: creates a cycle $\Rightarrow$ skip.

We can clearly see in Fig. 3.3e that after stage one there are still two components left, so we need to apply stage 2 to repair our partial solution. The shortest-delay-path to node 4 is defined by $P(s, 4) = \{\{s, 3\}, \{3, 5\}, \{5, 4\}\}$. Adding edge $\{s, 3\}$ creates a cycle which is dissolved by removing edge $\{p_3, 3\} = \{2, 3\}$. The next edge $\{3, 5\}$ can be skipped since it is already contained in our solution. Due to the decrease of the delays of nodes 3 and 5 to values $\delta_3 = 1,\ \delta_5 = 2$, it is now feasible to add edge $\{5, 3\}$ finally resulting in a feasible tree T with costs $c_T = 19$. For comparison, an optimal solution T^* has costs $c_{T^*} = 18$ and edge set $E' = \{\{s, 2\}, \{s, 3\}, \{1, 3\}, \{3, 5\}, \{4, 5\}\}$.

3.4.4 Modifications

Two modifications in stage one usually lead to better results when applying a subsequent improvement method, such as those described in Section 3.8:

1. A delay-factor $F_d \geq 1$ is introduced and multiplied with the left side of inequalities (3.13) and (3.14) when checking the feasibility of adding an edge. In other words, the delay-bound is lowered by the factor $\frac{1}{F_d}$.

2. If stage one has added a predefined number of edges less than $|V| - 1$ it is aborted and stage two uses shortest-delay-paths to attach the left components.

Both modifications provide a solution where the gap between the node-delays d_v and the delay-bound is usually larger than in the spanning tree of the standard implementation. This higher "residual delay" leads to more possibilities in a following improvement phase and therefore often results in final solutions with less total cost.

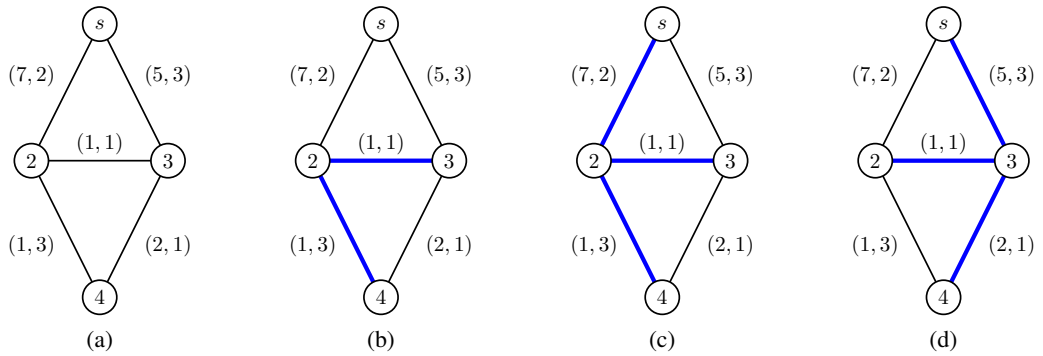


Figure 3.4: A small example graph (a) with delay-bound $B = 5$. The edge description is read (c_e, d_e). Adding the cheapest edges to the solution in (b) forces the use of an expensive edge in (c). By also considering the delay a better solution (d) can be created.

3.5 Multilevel Construction Heuristic

In the previously discussed construction heuristics, see Sections 3.2 and 3.4, the inclusion of an edge with low costs is not necessarily cheap w.r.t. the overall solution. If an edge with low costs but high delay is used it can affect the further construction of the solution negatively. The high delay can force a heuristic to use very expensive edges with low delay in order to not violate the delay-constraint. Such decisions sometimes create weak solutions corresponding to poor local optima which even good improvement procedures are not able to overcome. An example is given in Fig. 3.4.

3.5.1 Ranking Score

In an attempt to estimate how promising an edge is, the ranking score is introduced. It is more likely that an edge with comparatively low costs and low delay is part of an optimal solution than an edge with very low costs but high delay. The ranking score

$$\sigma_e = \left(1 - \frac{r_e^c - 1}{|E|}\right) \cdot \left(1 - \frac{r_e^d - 1}{|E|}\right) \tag{3.15}$$

describes the relative cost in relation to the delay of an edge $e \in E$ in comparison to other edges; $r_e^c \in \{1, \ldots, |E|\}$ and $r_e^d \in \{1, \ldots, |E|\}$ represent the cost and delay ranks of edge e obtained by sorting the edges according to costs and delays, respectively. After normalizing the ranking and subtracting from 1 in order to ensure that lower ranks result in higher scores the partial cost and delay scores are multiplied. The resulting ranking score $\sigma_e \in [0, 1]$ is an indicator for the quality of an edge $e \in E$.

The ranking score is further extended to vertices. To calculate the ranking score σ_v of a node $v \in V$ we sum up the ranking scores of all incident edges. That way the ranking score of a node is high if high quality or a high number of edges are connected to that node. For example the

ranking score of an outlying node with few, possibly bad, connections is lower than the ranking score of a central node with many connections.

3.5.2 Ranking-Based Multilevel Heuristic

The previous construction heuristics referred to in Section 3.2 and 3.4 are based on adding edges to a partial solution trying to minimize the costs in each step. However, the delay is ignored as long as no constraint violation occurs. This can sometimes lead to relatively poor solutions with a rather low potential for further improvement by local search methods. This motivates a heuristic that uses the above described ranking score to decide which edges should be part of a solution.

Our approach is based on the multilevel refinement paradigm [183], firstly creating a hierarchy of approximations of the original problem by recursive coarsening. After an initial solution has been found on the coarsest level it is iteratively refined in each level obtaining a feasible solution for the original problem in the end. In our case the vertices are iteratively merged to components until only one component is left. The key difference to the Kruskal-based construction heuristic from Section 3.4 is the iterative merge process. In each level a number of vertices, including the source node, is selected as so-called super-vertices. The remaining vertices are connected directly to these super-vertices creating multiple subtrees in each level. These subtrees are contracted to vertices in the next level and the process continues until only the source node remains. The resulting tree is a spanning tree and due to checks during the merge process it is guaranteed that the delay-constraints are not violated. The construction heuristic is shown in detail in Algorithm 3.2.

Selecting Super-vertices

In each level the ranking-based multilevel heuristic has to choose a number of vertices to become super-vertices. These super-vertices act as root nodes to which the remaining vertices can be connected. For a practical application, i.e. a shipment organization, this can be compared to choosing the site of a regional distribution center and creating a hierarchical network of transportation. The two major questions concerning super-vertices are how many vertices should become super-vertices and which vertices should be chosen.

The number of super-vertices chosen during each level is determined by a user parameter called $superrate$, a simple percentage. A low superrate leads to a low number of super-vertices, therefore to a high number of remaining vertices which have to be connected. The advantage of a low superrate is comparatively fast coarsening since the number of levels will be low, too. However since the number of super-vertices is directly related to the number of possible connections for each node the search space is smaller. A low superrate is a promising choice if the solution is expected to be a star-like network. In constrast a higher superrate leads to a slower coarsening since more levels can be expected. Note that the superrate is not directly related to the number of levels due to a mechanism ensuring a feasible solution which will be introduced later. The obvious advantage of a high superrate is that more and maybe better connections are available for each non-super-vertex.

Algorithm 3.2: Ranking-Based Multilevel Heuristic

Input: graph $G = (V, E)$

Output: feasible solution $T = (V, E')$ to the RDCMST problem

calculate ranking scores $\sigma_e,\ \forall e \in E$, and $\sigma_v,\ \forall v \in V$

L_E ... list of edges E sorted by decreasing ranking scores σ_e

$V' = V$

$p_v = v,\ \delta_v^{\max} = 0,\ \forall v \in V$ `// predecessors, subtree delays`

while $\exists v \in V \setminus \{s\} : p_v = v$ **do**

  choose super-vertices $S \subseteq V'$ depending on scores σ_v and parameter *superrate*

  $V' = V' \setminus S$

  forall the $e = \{u, v\} \in L_E$ **do**

    if $u \in S\ \wedge\ v \in V'\ \wedge\ p_v = v\ \wedge\ d_u^{\min} + d_{uv} + \delta_v^{\max} \leq B$ **then**

      $p_v = u,\ E' = E' \cup \{e\},\ V' = V' \setminus \{v\}$

      if $\delta_u^{\max} < \delta_v^{\max} + d_{uv}$ **then** $\delta_u^{\max} = \delta_v^{\max} + d_{uv}$

    if $u \notin S \vee v \notin S$ **then** $L_E = L_E \setminus \{e\}$

  `// connect rest of nodes via shortest-delay-paths`

  forall the $v \in V'$ **do**

    u ... predecessor of v in shortest-delay-path $P(s, v)$

    $S = S \cup \{u\},\ V' = V' \setminus \{u, v\}$

    $p_v = u,\ E' = E' \cup \{\{u, v\}\}$

    if $p_u \neq u$ **then**

      if $\delta_{p_u}^{\max} = \delta_u^{\max} + d_{p_u u}$ **then** $\delta_{p_u}^{\max} = \max_{w \in V,\ p_w = p_u} \{\delta_w^{\max} + d_{p_u w}\}$

      $E' = E' \setminus \{\{p_u, u\}\},\ p_u = u$

    if $\delta_u^{\max} < \delta_v^{\max} + d_{uv}$ **then** $\delta_u^{\max} = \delta_v^{\max} + d_{uv}$

    forall the $e = \{u, w\} \in E \setminus L_E,\ w \in S$ **do** $L_E = L_E \cup \{e\}$

  $V' = S$

  update vertex ranking scores $\sigma_v,\ \forall v \in V'$, considering only edges $e \in L_E$

return T

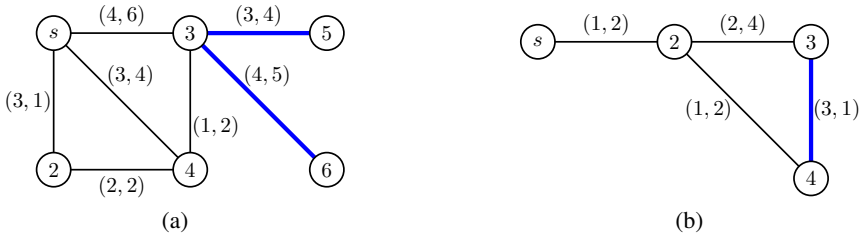


Figure 3.5: (a) An example graph with $B = 10$. We consider edge $\{3, 4\}$: $d_4^{\min} = 3, d_3^t = 5$. (b) An example graph with $B = 5$. Node 3 can only be connected via node 4.

The second question is which vertices should become super-vertices. Here we apply the ranking score for vertices. The vertices with the highest ranking scores are those with either a high number of connections, thus ensuring a high number of possibilities, or very promising connections. In case of equal ranking scores super-vertices are randomly selected making the selection process non-deterministic. Finally, root node s is always selected as super-vertex.

Merge Process

After the selection of super-vertices the next step is to connect the remaining vertices. Sorted by ascending ranking scores, the only edges considered in the merging process are those between super-vertices and other nodes. If

$$d_u^{\min} + d_{uv} + \delta_v^{\max} \leq B \tag{3.16}$$

is satisfied for an edge $\{u, v\}$ we know that its use would not violate the delay-constraint. $\delta_v^{\max}$ represents the delay caused by the current subtree of node v, see Fig. 3.5a. If node 4 is chosen as super-vertex and we want to use edge $\{3, 4\}$ to connect node 3, we have to consider $d_4^{\min}$ and $\delta_3^{\max}$. Summing up these delays plus the edge delay results in an overall delay equal to B. Therefore, this edge can safely be used to connect node 3.

However, there is no guarantee that all non-super-vertices can be connected this way. Figure 3.5b illustrates the problem. For a delay-bound of 5 the only possible path to connect node 3 is via node 4. In case node 4 is not a super-vertex there is no possibility to connect node 3. Therefore, a repair strategy for these problematic vertices is required. If an instance is solvable a feasible path to connect a node to the source is given by the shortest-delay-path, cf. Section 3.4.2. For each problem node the immediate predecessor in the shortest-delay-path becomes a super-vertex in the current level. Additionally, a connection between the new super-vertex and a possibly already assigned predecessor is removed. This way a new subtree is created.

After this merge process all non-super-vertices are connected and a set of subtrees with super-vertices as their root remains. These subtrees are contracted and represent the vertices in the next level, whereas only edges connecting two super-vertices are now considered anymore. This process is continued until only the source node remains, corresponding to a feasible solution for the original problem. The RBMH runs in $\mathcal{O}(|E| \log |E| + |V|^2)$ time, see [14] for a detailed runtime analysis.

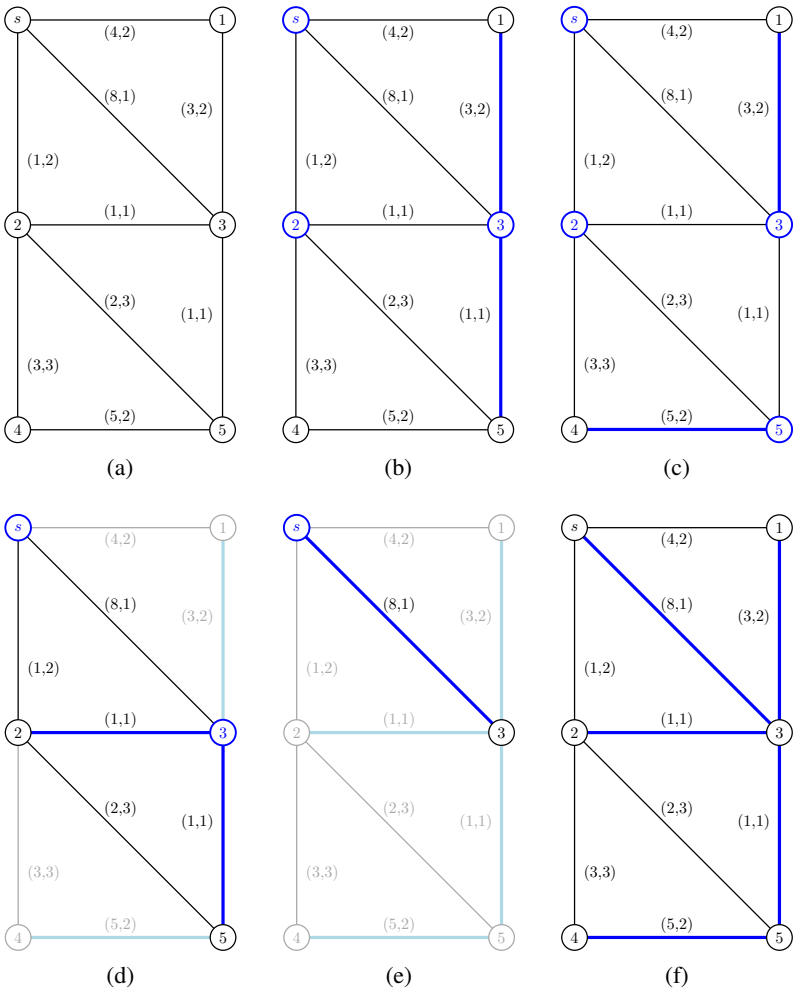


Figure 3.6: Applicaton of ranking-based multilevel heuristic on input graph in (a) with $B = 4$: Level 1 in (b)–(c), level 2 in (d), level 3 in (e), and the final solution in (f).

3.5.3 Example

Applying the ranking-based multilevel heuristic on the input graph from Section 3.4.3 with delay-bound $B = 4$ results in the solution construction shown in Fig. 3.6. According to Formula 3.15 the edge ranking scores are: $\sigma_{\{s,1\}} = 0.22$, $\sigma_{\{s,2\}} = 0.67$, $\sigma_{\{s,3\}} = 0.11$, $\sigma_{\{1,3\}} = 0.37$, $\sigma_{\{2,3\}} = 1$, $\sigma_{\{2,4\}} = 0.12$, $\sigma_{\{2,5\}} = 0.15$, $\sigma_{\{3,5\}} = 1$, $\sigma_{\{4,5\}} = 0.15$. Corresponding node ranking scores are: $\sigma_s = 1$, $\sigma_1 = 0.59$, $\sigma_2 = 1.94$, $\sigma_3 = 2.48$, $\sigma_4 = 0.27$, $\sigma_5 = 1.3$. The parameter controlling the selection of super-vertices is set to $superrate = 50\%$, here. Super-vertices are colored blue in Fig. 3.6. The following coarsening steps are performed:

- Level 1 (Fig. 3.6b): $S = \{s, 2, 3\}$, $V' = \{1, 4, 5\}$,
  $L_E = \{\{2,3\}, \{3,5\}, \{s,2\}, \{1,3\}, \{s,1\}, \{2,5\}, \{4,5\}, \{2,4\}, \{s,3\}\}$:

1. $\{2,3\}$: connects two super-vertices $\Rightarrow$ skip.
2. $\{3,5\}$: $1+1+0 \leq B \Rightarrow p_5 = 3,\ E' = \{\{3,5\}\},\ \delta_3^{\max} = 1.$
3. $\{s,2\}$: connects two super-vertices $\Rightarrow$ skip.
4. $\{1,3\}$: $1+2+0 \leq B \Rightarrow p_1 = 3,\ E' = \{\{3,5\},\{1,3\}\},\ \delta_3^{\max} = 2.$
5. $\{s,1\}$: node 1 is already connected $\Rightarrow$ skip.
6. $\{2,5\}$: node 5 is already connected $\Rightarrow$ skip.
7. $\{4,5\}$: connects two non-super-vertices $\Rightarrow$ skip.
8. $\{2,4\}$: $2+3+0 > B \Rightarrow$ skip.
9. $\{s,3\}$: connects two super-vertices $\Rightarrow$ skip.

Set $V' = \{4\}$ is still non-empty. Therefore, we connect node 4 using the last edge of the shortest-delay-path $P(s,4) = \{\{s,3\},\{3,5\},\{5,4\}\}$: $S = \{s,2,3,5\},\ p_4 = 5,\ E' = \{\{3,5\},\{1,3\},\{4,5\}\}$. Node 5 is now a super-vertex and thus its connection to node 3 is removed again: $E' = \{\{1,3\},\{4,5\}\},\ p_5 = 5,\ \delta_5^{\max} = 2$, see Fig. 3.6c. Relevant vertex ranking scores are now: $\sigma_s = 0.78,\ \sigma_2 = 1.82,\ \sigma_3 = 2.11,\ \sigma_5 = 1.15$.

- Level 2 (Fig. 3.6d): $S = \{s,3\},\ V' = \{2,5\}$,
  $L_E = \{\{2,3\},\{3,5\},\{s,2\},\{2,5\},\{s,3\}\}$:
  1. $\{2,3\}$: $1+1+0 \leq B \Rightarrow p_2 = 3,\ E' = \{\{1,3\},\{4,5\},\{2,3\}\},\ \delta_3^{\max} = 2.$
  2. $\{3,5\}$: $1+1+2 \leq B \Rightarrow p_5 = 3,\ E' = \{\{1,3\},\{4,5\},\{2,3\},\{3,5\}\},\ \delta_3^{\max} = 3.$
  3. $\{s,2\}$: node 2 is already connected $\Rightarrow$ skip.
  4. $\{2,5\}$: connects two non-super-vertices $\Rightarrow$ skip.
  5. $\{s,3\}$: connects two super-vertices $\Rightarrow$ skip.

  Set V' is now empty. Thus, we only have to update the vertex ranking scores needed for the next level: $\sigma_s = \sigma_3 = 0.11$.

- Level 3 (Fig. 3.6e): $L_E = \{\{s,3\}\},\ S = \{s\},\ V' = \{3\}$. Edge $\{2,3\}$ can be added since $0+1+3 \leq B$: $p_3 = s,\ E' = \{\{1,3\},\{4,5\},\{2,3\},\{3,5\},\{s,3\}\},\ \delta_s^{\max} = 4$.

Since all nodes $v \in V \setminus \{s\}$ now have a valid predecessor, edges E' form a feasible solution, see Fig. 3.6f. In this case we even obtain an optimal solution with $c_T = 18$.

3.6 Greedy Randomized Adaptive Search Procedure

To provide many different feasible starting solutions for a subsequent improvement phase we extended stage one of the Kruskal-based construction heuristic towards a *greedy randomized adaptive search procedure* (GRASP) [48]. In each iteration of stage one we do:

1. store all feasible edges in a candidate list (CL),

2. select a subset of least-cost edges of CL with

$$c_e \leq \min_{e \in CL} c_e + \alpha \cdot (\max_{e \in CL} c_e - \min_{e \in CL} c_e), \tag{3.17}$$

   for a predefined parameter $\alpha \in [0, 1]$ and insert them into a restricted candidate list (RCL),

3. randomly choose an edge from the RCL, and

4. merge components by adding this edge, see Section 3.4.1.

3.7 Neighborhood Structures

Local search methods are commonly used to heuristically improve existing feasible solutions and can easily be integrated in many other metaheuristics and exact methods, see 2.2 and 2.3. However, their efficiency heavily depends on the used neighborhood structure and the according search algorithm. Too small neighborhoods possibly restrict the search space too much while too large ones might not be completely searched in reasonable time. Additionally, in many cases it is not trivial to provide an efficient feasibility check to quickly sort out infeasible neighboring solutions. In preliminary tests we experimented with different neighborhood structures for the RDCMST problem, but here we only discuss the two most relevant and successful ones, the Edge-Replace and Component-Renew neighborhoods.

3.7.1 Edge-Replace Neighborhood

A move in the Edge-Replace neighborhood simply removes an edge and connects the resulting two components in the cheapest feasible way. To efficiently find a new feasible edge after removing one from the solution, we first compute the maximal delay extents $d_v^{\max}$ for all nodes v in the component which is now separated from the root node. The determination of these values takes linear time, see Section 3.4.1, but then allows us to check the feasibility of a candidate edge $\{u, v\}$ in constant time by testing the condition

$$d_u + d_{uv} + d_v^{\max} \leq B, \tag{3.18}$$

where node u is part of the root component and node v is included in the separated component. By examining all candidates in the order of ascending costs we therefore are able to find the cheapest feasible replacement edge in $\mathcal{O}(|E| \log |E|)$ time.

Since we use additional supporting data structures, it turned out to be practically beneficial w.r.t. runtime to always perform a complete neighborhood search until a local optimum is reached before turning to an alternative neighborhood structure. Furthermore, we apply a next improvement step function considering the removal edges in decreasing cost order since the more time-consuming best improvement provided no significant advantage, here. Both the set of edges in the current solution and the set of candidate edges for replacement are managed by a priority queue based on heaps [31] to allow constant time access to the most expensive and cheapest edge, respectively, and insertions and deletions in logarithmic time.

Due to the required cost decrease in each move it can easily be seen that we can perform at most $\mathcal{O}(|E|)$ moves, leading to the whole local search in the Edge-Replace neighborhood running in $\mathcal{O}(|E|^2 \log |E|)$ time.

3.7.2 Component-Renew Neighborhood

A Component-Renew move also deletes an edge, but completely dissolves the component which is now separated from the root node. It then re-adds the individual nodes by applying Prim's algorithm [145] always respecting the delay-bound. A feasibility check for a candidate edge $\{u, v\}$ within Prim's algorithm can be done in constant time by testing

$$d_u + d_{uv} \leq B, \tag{3.19}$$

where node u is part of the root component and node v is one of the singletons. In some cases not all nodes can be added due to the delay-bound, cf. [166]. These remaining nodes are again joined to the root component by shortest-delay-paths, dissolving created cycles, see Section 3.4.2. Altogether, one move takes $\mathcal{O}(|E| \log |E|)$ time which is dominated by the management of candidate edges in priority queues for Prim's algorithm.

The cost gain of one specific move cannot be determined until the end of this move which may comprise many modifications to the tree. In the beginnings we used to work on a solution copy which turned out to be inefficient in practice because of repeated creations and deletions of complete trees. Thus, we introduced some kind of snapshot mechanism recording all modifications within a move. If the total tree cost finally increases we just undo all operations, else we do not need to do anything but deleting all records. Additionally, in some cases we are able to decide a priori whether a move can improve a solution or not: Assume that we previously performed an improving move by removing edge $\{u, v\}$ and renewing the subtree of v by Prim's algorithm. Then it makes no sense to consider an edge part of this new subtree of v as long as node delay d_v is not decreased by subsequent moves or adding shortest-delay-paths, since we would obtain exactly the same solution again.

A neighborhood search is done similarly as for the Edge-Replace neighborhood following a next improvement strategy considering the edges in decreasing cost order until a local optimum is reached, running in $\mathcal{O}(|E|^2 \log |E|)$ time, too.

3.8 Variable Neighborhood Descent

We introduce a *variable neighborhood descent* (VND) [79] for improving a constructed solution by performing a local search switching between the two previously described neighborhood structures: Edge-Replace and Component-Renew. The standard implementation of a VND as it is described in [79] was modified to provide here better results in shorter runtime: A neighborhood structure is searched by next improvement until a local optimum is reached, see Section 3.7 for details; then we switch to the other one continuing until no better solution can be found anymore. However, a VND only provides mechanisms for intensification without a possibility to escape from a local optimum w.r.t. all used neighborhood structures.

3.9 General Variable Neighborhood Search

To further improve the quality of a constructed solution we apply the metaheuristic framework *general variable neighborhood search* (GVNS) as introduced by Hansen et al. [79]. Compared to the VND described before, a GVNS includes methods for diversification to possibly escape from local optima. In each iteration the so far best solution is perturbed by shaking and then improved by an embedded VND.

3.9.1 Shaking

Three different kinds of shaking moves are used in the GVNS framework. The algorithm always chooses one at random with equal probabilities:

1. Replacing a random edge by another randomly chosen edge not violating the delay-constraint and tree structure.
2. Adding the shortest-delay-path to a random node, see Section 3.4.2.
3. Adding the delay-constrained least-cost path to a random node, see Section 3.3.2 for details.

To ensure feasibility of the solution two issues have to be considered: Firstly the last two shaking moves could possibly cause cycles which have to be dissolved, and secondly the delay-bound in the third move can be set at most to the global delay-bound reduced by the maximal delay in the subtree of the randomly chosen node. The number of shaking moves performed in one iteration is $\lceil |V| * sr \rceil$, where $sr \in (0, 1]$ is called shaking rate. This shaking rate is either set to a fixed value or dynamically changed in the search process. In the latter case sr is initialized with 0.01, increased by 0.01 when no better solution could be found in an iteration, limited from above by 0.3, and reset again to 0.01 in case of an improvement.

3.10 Ant Colony Optimization

We apply the concept of the $\mathcal{MAX} - \mathcal{MIN}$ Ant System (MMAS) by Stützle et al. [176] to our problem implementing the following key features:

- Only a single ant is allowed to deposit pheromones at the end of an iteration, either the best ant of the iteration or the globally best one, focusing the search to the best solutions found.
- Pheromone values are limited to an interval $[\tau_{\min}, \tau_{\max}]$ preventing a stagnation of the search.
- The initial pheromone values are set to $\tau_{\max}$ leading to a high diversification at the beginning of the search.

Combining all these features provides both intensification and diversification throughout the search process, which is essential for a well-performing metaheuristic.

3.10.1 Pheromone Values

Pheromone values $\tau_{v,\delta} \in [\tau_{\min}, \tau_{\max}]$ are defined for each node $v \in V \setminus \{s\}$ in combination with any node delay $\delta_v \in [1, B]$ it might have. The root s by definition always has a node delay 0 and therefore has no pheromone values associated.

Notice that if we would consider real-valued delays the pheromone values do not form a classical finite matrix. However, when considering a specific instance graph the number of feasible node delays is finite (but possibly very large). Therefore, in an implementation one has to consider efficient techniques for handling large sparse matrices [45].

Limits $\tau_{\min}$ and $\tau_{\max}$ are initialized and updated according to [176] using parameter $p_{\text{best}} = 0.00005$. Preliminary tests have shown that these parameter settings work well in general.

3.10.2 Solution Construction

The method for constructing a solution based on the pheromone values is inspired by the level-based construction heuristic introduced in [75] for the BDMST problem and runs in time $\mathcal{O}(|V| \cdot B + |V| \log |V| + |E|)$:

1. For each node $v \in V \setminus \{s\}$ a delay value δ_v is selected with a probability proportional to the according pheromone value.

2. All nodes are then sorted by these delay values in ascending order.

3. The nodes are added in the specified order to the existing tree – initialized with the root node – always choosing the cheapest possible edge without causing a node delay higher than the selected delay. If there is no edge satisfying this constraint, the shortest-delay-path to the problematic node is added, overriding the given order but guaranteeing a feasible solution.

3.10.3 Local Improvement

After its construction, a solution is improved either by the VND or by a local search in one of the two neighborhoods Edge-Replace or Component-Renew, see Sections 3.7 and 3.8, depending on the instance size. In the latter case the neighborhood Edge-Replace is chosen with probability 0.8 because of its usually higher performance.

3.10.4 Depositing Pheromones

After each ant constructed and improved a solution the pheromone values are updated. Here the mixed strategy suggested in [176] is used: At the beginning of the search only the best ant of the current iteration is allowed to deposit its pheromones. Later in the search process intensification has higher priority in order to concentrate on the surrounding of the so far best solution. This leads to the following update strategy: The more iterations have been performed, the higher the frequency of reinforcing the pheromone trail of the so far best solution instead of the iteration best one. More precisely, an instance-dependent number of iterations $I = \frac{50000}{|V|}$ is defined. In iterations $[1, I]$ only the iteration best solution deposits pheromones, in $(I, 2I]$ the so far best

solution is chosen every fifth iteration, in $(2I, 3I]$ every third iteration, in $(3I, 6I]$ every other iteration and in $(6I, \infty)$ every time. A predefined pheromone decay coefficient p controls the evaporation and enforcement of the pheromone values.

3.11 Memetic Algorithm

Here, we describe a genetic algorithm for the RDCMST problem, using a special solution representation and incorporating an improvement method to finally form a memetic algorithm (MA).

3.11.1 Solution Representation

One of the key aspects in designing genetic algorithms is choosing a meaningful solution representation. This is not an easy task especially when dealing with solutions represented by complex graph structures, e.g. constrained trees. An obvious encoding of general graph structures is a binary array of length $|E|$ indicating which edges are part of the solution. However, only a small subset of all possible edge selections may correspond to feasible solutions, e.g. for the RDCMST problem, and naive standard variation operators are therefore unlikely to produce feasible solutions. There are various encodings intended to uniquely represent spanning trees, e.g. Prüfer codes [146], but here again we are faced with the problem that many trees may violate the delay-constraints.

Similar to [75] and our ant colony optimization approach, the genotype in our genetic algorithm consists of an array of length $|V| - 1$ with one delay value $\delta_v \in [1, B]$ assigned to each node $v \in V \setminus \{s\}$. Value δ_v here represents the maximal allowed delay of path $P(s, v)$ in the corresponding phenotype. To convert such a delay array to a feasible constrained spanning tree we use the following decoding method which is similar to the solution construction in the ant colony optimization approach, see Section 3.10.2:

1. Sort all nodes $v \in V \setminus \{s\}$ by delay values δ_v in ascending order.
2. Initialize tree with source s.
3. Add next node v in the given order to the tree by choosing the cheapest possible edge without causing a delay higher than δ_v on the path $P(s, v)$; if there is no such edge the shortest-delay-path to v is added and possibly introduced cycles are dissolved.
4. If the tree spans all nodes we obtain a feasible solution, else go to 3.

The decoding method runs in $\mathcal{O}(|V| \log |V| + |E|)$ time. Encoding a feasible tree T to a delay array runs in $\mathcal{O}(|V|)$ time by simply using the actual path delays $\delta_v = d_v^T$, $\forall v \in V \setminus \{s\}$. Important about this representation is that every delay array can be decoded to a feasible solution but there is no bijective mapping between delay array and tree: different delay arrays may decode to the same tree while different trees may be encoded by the same delay array. Even encoding and decoding in a row may not lead to the same tree but the resulting tree costs are guaranteed to be at least as low. Additionally, small changes of the delay array caused by genetic operators may result in completely different solutions leading to poor locality

Algorithm 3.3: Memetic algorithm with duplicate detection

```
1  initialize(P)
2  while time limit is not reached do
3      (p1, p2) = select(P)
4      o = recombine(p1, p2)
5      mutate(o)
6      if is_duplicate(o, P) then restart loop or transform(o)
7      T = decode(o)
8      improve(T)
9      o = encode(T)
10     if is_duplicate(o, P) then restart loop or transform(o)
11     replace(o, P)
```

and heritability, similar to the unique encoding of trees by Prüfer numbers [63]. However, in preliminary tests all these disadvantages turned out to be outweighed by the possibility to easily apply standard genetic operators without caring about solution feasibility. Furthermore, the poor locality of our encoding acts as additional diversification mechanism complementing the intensification methods in our MA.

3.11.2 Components and Operators

Our MA is based on a steady-state genetic algorithm [188] selecting only two parent individuals to produce one offspring per iteration or time step, see Algorithm 3.3. The main components and operators have been decided in preliminary tests:

- Population initialization: a random delay value $\delta_v \in [1, B]$ is assigned to each node $v \in V \setminus \{s\}$ of an individual
- Selection: parent individuals are selected by binary tournaments
- Recombination: an offspring is derived by uniform crossover proportional to the parents' solution quality
- Mutation (two different operators):
  - a different random delay is assigned to a node v with probability p_m
  - the delays of two different, randomly chosen nodes are swapped
- Replacement: an offspring randomly replaces one of the r worst individuals

3.11.3 Improvement

Additionally, offsprings are locally improved after mutation by local search methods previously described, see Sections 3.7 and 3.8. Depending on the instance size the individuals are either

v	1	2	3	4	5
δ_v	2	3	1	4	2

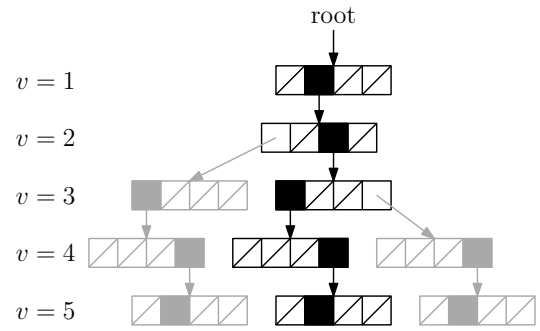


Figure 3.7: Given an instance with five nodes and delay-bound $B = 4$. The solution archive on the right contains three solutions where the black trie nodes correspond to the solution encoded by the delay array on the left.

improved by a local search in a single neighborhood or by a VND switching between the two neighborhood structures Edge-Replace and Component-Renew.

3.12 Tackling Duplicates

One of the basic problems of local search and population-based heuristics is the potentially repeated examination of already visited solutions. We exemplarily analyze this issue for our memetic algorithm. Duplicates decrease the diversity in a population and time is wasted by analyzing or trying to improve these solutions. In a first rather obvious approach to detect revisits hash values of all individuals are computed and maintained in a hash-table. We only store hash values of individuals in the current population, hashes of replaced solutions are discarded. In Section 3.13.6 we will see that this apparently artificial limitation is quite beneficial. However, an efficient transformation of duplicates to guaranteed unvisited solutions is not possible.

In a more sophisticated second approach a complete solution archive is built efficiently storing solutions and making it possible to derive new unvisited solutions as replacements of detected duplicates. Promising experiments with similar solution archives to enhance standard genetic algorithms for binary benchmark problems are presented in [152, 175]. Here we adopt and extend this concept for our MA. As in [152], our archive uses a *trie* data structure, which is mostly known from the domain of (language) dictionaries, where a huge number of words has to be stored in a compact way. In our trie, each node contains an array of B references to nodes at the next level, and at each level a dedicated node's delay in a given solution array decides which pointer to follow. Therefore, a single solution is uniquely represented by $|V| - 1$ trie nodes. An example is given in Fig. 3.7. In this way, the trie has maximum height $\mathcal{O}(|V|)$, and an insertion operation and a check whether or not a solution is already contained can always be done in time $\mathcal{O}(|V|)$ independently of the number of stored solutions. Some special adaptions are applied to the basic trie data structure in order to reduce the used space while at the same time not increasing access time too much. More specifically, not all delay values are feasible for a node, so the number of array elements of a trie node can be appropriately reduced. To maintain constant access time to an array element a global mapping between delay values and array indices is

stored. Furthermore, fully explored subtrees are pruned and replaced by an appropriate marker. The essential aspect which makes our archive approach different to more common simple solution caching strategies as e.g. described in [106], is the provision of a function that derives for each duplicate a typically similar but definitely not yet considered delay array. This operation can also be seen as a kind of "intelligent" mutation. In general finding an unvisited delay array in the archive takes $\mathcal{O}(|V|)$ time and the modification is done by assigning a randomly chosen unvisited delay value to a random node. An interesting, although more theoretical side effect of the extension of a metaheuristic by our archive is that the metaheuristic in principle becomes a complete, exact optimization approach with bounded runtime: In each iteration, (at least) one new delay array is evaluated, and by the archive it is also efficiently possible to detect when the whole search space has been covered and the search can be terminated.

An important question is where to integrate the archive in the (meta-)heuristic process and which metaheuristics can benefit from such an extension at all. At some points the solution diversity may be very high and the probabilities of revisits low, e.g. after shaking the solution randomly. Then, the archive would just grow very large possibly consuming too much space. At other points revisits typically occur more frequently, e.g. after applying local improvement methods, but due to the structure of the metaheuristic it cannot benefit much from consulting the archive. Generally speaking, the solution archive must be used with caution but has the potential to speed up a metaheuristic significantly, cf. [152]. We integrated the duplicate check at two different positions in our MA, see Algorithm 3.3. The first check is performed immediately before decoding the delay array and improving the solution to prevent wasting time on revisits, the second after encoding the solution again to preserve diversity in the population.

3.13 Computational Results

In this section we provide a computational comparison of our heuristic approaches to solve the RDCMST problem: Kruskal-based construction heuristic (KBH), ranking-based multilevel construction heuristic (RBMH), GRASP, variable neighborhood descent (VND), general variable neighborhood search (GVNS), ant colony optimization (MMAS), and memetic algorithm (MA). Since there is only one previously existing heuristic method – the Prim-based heuristic (PBH) by Salama et al. [166], see Section 3.2 –, we mainly compare our algorithms to each other. The state-of-the-art exact approaches for the RDCMST problem are considered in comparisons in Chapter 4 when discussing MIP methods for the RDCST problem.

3.13.1 Test Instances and Environment

Several benchmark instances for the RDCMST problem are used in the work by Salama et al. [166] comprising graphs with 20 to 200 nodes. However, it was not possible to get further information about these graph data. Gouveia et al. [68] apply their MIP approaches on graphs with 20 and 40 nodes, which are used for computational results in Section 4.11 when considering exact approaches, but are much too small to draw reasonable conclusions for heuristics with statistical significance. For example the GVNS, see Section 3.9, is able to find optimal solutions for most of these instances within seconds.

Therefore, we generated our own instance sets R100, R500, R1000, and R5000 each containing 30 complete instances with 100, 500, 1000, and 5000 nodes, respectively, and random integer edge costs and delays uniformly distributed in $[1, 99]$. The root node s is set to node 0 for all instances. All instances are available for download in the web[1].

Additionally, we used two sets EU500 and EU1000 each consisting of 15 Euclidian instances from the OR-Library [12] originally used for the Euclidian Steiner tree problem [13]. These instances consist of 500 and 1000 points, respectively, randomly distributed in the unit square and the edge costs correspond to the Euclidian distances between these points. We extended these input data by edge delays normally distributed around the associated costs and chose a point near the center as root node. To obtain integer cost and delays the according real values are scaled by 10^7 and rounded. The final objective values are then converted to the original domain back again. Because of these extremely large delay values the preprocessing test based on alternative paths, the MMAS, and the MA cannot successfully be applied to these instances since both rely on rather small sets of achievable delay values and delay-bounds of about $B \leq 1000$. Therefore, we used these instance sets only in Section 3.13.3.

The testing environment for comparing the Prim-based construction heuristic by Salama et al. [166] to our Kruskal-based heuristic in Tables 3.2 and 3.3 consists of Intel Core 2 Quad Q9550 processors with 2.83 GHz where four cores share 8 GB memory. All other tests are performed on Intel Xeon E5540 processors with 2.53 GHz where eight cores share 24 GB of memory. However, all implemented algorithms only run on single cores without utilizing multi-core architecture.

3.13.2 Preprocessing

Edges $e \in E$ with $d_e > B$ are already omitted when importing a graph instance, so we do not consider those edges here anymore. The rest of the preprocessing methods described in Section 3.3 are applied in the following order to an instance:

1. discard infeasible edges satisfying the shortest-delay-path conditions (3.3)
2. discard redundant edges satisfying the root edges conditions (3.4)
3. discard redundant edges based on alternative triangles, see Theorem 3.3.1
4. discard redundant edges satisfying the alternative path conditions (3.12)

The results of these preprocessing phases for instance sets R100, R500, and R1000 are shown in Table 3.1. Keep in mind that cost and delay values of these random instances are within the interval $[1, 99]$. The first phase is able to discard many infeasible edges when considering low delay-bounds and clearly is less successful for high delay-bounds. The third phase based on alternative triangles is the most successful method here whereupon the graph density is directly correlated with the percentage of discarded edges. Since the first phase already removes large delay edges and the second phase is kind of related to the third one, we can see that the theoretically derived expected number of discarded edges in the third phase is near our results, see

[1] `https://www.ads.tuwien.ac.at/w/Research/Problem_Instances`

Set	B	$\overline{\lvert E \rvert}$	$\overline{\lvert E_p \rvert}$	$\overline{\lvert E_r \rvert}$ %	$t[s]$	$\overline{\lvert E_r^1 \rvert}$ %	$t^1[s]$	$\overline{\lvert E_r^2 \rvert}$ %	$t^2[s]$	$\overline{\lvert E_r^3 \rvert}$ %	$t^3[s]$	$\overline{\lvert E_r^4 \rvert}$ %	$t^4[s]$
R100	16	798	490	39	0	35	0	0	0	2	0	1	0
	30	1501	932	38	0	19	0	1	0	11	0	8	0
	50	2498	1269	49	0	11	0	3	0	24	0	11	0
	100	4950	1695	66	0	5	0	12	0	39	0	10	0
R500	6	7560	3338	56	0	55	0	0	0	1	0	0	0
	20	25204	11666	54	5	16	0	1	0	20	0	17	5
	50	63029	16454	74	20	7	0	3	0	49	0	15	20
	100	124750	20108	84	45	2	0	12	0	59	1	11	44
R1000	6	30254	14928	51	1	48	0	0	0	1	0	1	1
	20	100874	35938	64	101	14	0	1	0	30	1	19	101
	50	252217	47938	81	300	6	0	3	0	58	3	14	296
	100	499500	57878	88	626	2	0	12	0	65	6	9	619

Table 3.1: Preprocessing applied on instance sets R100, R500 and R1000 (B: delay-bound, E_p: edge set after preprocessing, E_r: set of discarded edges, $t[s]$: median runtimes of complete preprocessing in seconds, E_r^i: set of edges discarded in preprocessing phase i, $t^i[s]$: median runtimes of preprocessing phase i).

Section 3.3.2: Considering the complete graphs for $B = 100$, for R100 instances we observe 56% removals in the first three phases compared to the expectation value 54% for stage three only, for R500 instances we obtain 73% compared to expected 73%, and for R1000 instances we obtain 79% compared to expected 79%.

It can be seen that the runtimes of the first three phases are below one second for all instance sets and can therefore be neglected. However, the runtimes of the fourth phase based on computing delay-constrained shortest paths heavily depends on the delay-bounds because of the pseudo-polynomial runtime of the underlying dynamic program. Thus, this last phase has to be used with caution, especially when using heuristic algorithms which are mainly used to tackle R500 and R1000 instances. Preprocessing is in general only senseful if the needed time is moderate compared to the runtime for construction and optimization stages. Furthermore, the total runtime including preprocessing clearly should not be higher than the total runtime without preprocessing.

Regarding exact algorithms which usually can only be applied to rather small instances with about 100 nodes in reasonable time when considering complete graphs, see Chapter 4 and 5, it is important to discard as many infeasible and redundant edges as possible, so the fourth phase should be enabled for all instances, here. Additionally, the consumed runtime of all preprocessing phases for those small instances is not worth mentioning.

Obviously, since the preprocessing tests based on alternative triangles and paths rely on the fact that neither cost nor delay values are Euclidian, they are not able to reduce instances with Euclidian edge costs or delays. Thus, in general the number of edges removed by the whole preprocessing phase is much lower if considering these kind of input graphs.

Without showing detailed results here, we observed in preliminary tests that the quality of heuristic solutions including preprocessing in general is not better than without preprocessing, but the runtime is usually shorter. In a few cases the final cost values of tests on the preprocessed

		R500						R1000					
		PBH			KBH			PBH			KBH		
B	Test	$\overline{c}$	σ	$t[s]$	$\overline{c}$	σ	$t[s]$	$\overline{c}$	σ	$t[s]$	$\overline{c}$	σ	$t[s]$
6	C	19651	1583	0.1	**10785**	643	0.0	24053	3065	0.5	**14717**	710	0.0
	CV	9624	624	0.8	**9177**	633	0.5	11691	845	4.0	**10123**	544	3.0
	CGV	9340	578	12.2	**9067**	643	9.2	10858	558	64.6	**9942**	505	57.5
8	C	13020	1709	0.0	**8285**	428	0.0	15291	1826	0.0	**11779**	575	0.0
	CV	6795	546	0.8	**6035**	292	0.5	9433	1163	4.2	**6796**	322	3.2
	CGV	6352	368	13.8	**5871**	293	12.8	7719	471	68.8	**6610**	284	60.3
10	C	9555	1666	0.0	**7071**	328	0.0	11275	2051	0.0	**10277**	500	0.0
	CV	5914	686	0.8	**4554**	210	0.8	7299	747	4.3	**5172**	219	3.3
	CGV	4975	274	14.7	**4421**	200	13.5	5715	408	72.7	**5040**	202	70.3
15	C	5793	1037	0.0	**5565**	401	0.0	**6945**	1113	0.1	7996	533	0.0
	CV	3941	432	1.1	**2939**	142	0.8	4726	562	4.7	**3402**	158	3.6
	CGV	3102	238	15.9	**2811**	117	16.0	3459	205	79.8	**3291**	121	86.4
20	C	**4235**	861	0.0	4733	379	0.0	**4972**	892	0.1	6788	437	0.1
	CV	2947	378	1.1	**2215**	117	0.9	3410	415	5.0	**2603**	108	5.1
	CGV	2247	192	15.0	**2124**	87	18.9	2579	112	84.9	**2517**	83	98.7
30	C	**2783**	400	0.0	3757	359	0.0	**3382**	502	0.2	5062	475	0.2
	CV	2011	245	1.2	**1553**	87	1.0	2314	204	7.5	**1888**	67	6.4
	CGV	1501	88	19.2	**1468**	69	21.7	1825	61	111.3	**1812**	56	134.3
40	C	**2070**	318	0.0	3353	353	0.0	**2540**	358	0.5	3979	416	0.5
	CV	1496	194	1.4	**1221**	52	1.1	1894	212	7.4	**1562**	55	7.4
	CGV	1167	56	20.8	**1155**	52	25.4	1491	45	134.1	**1486**	42	189.1

Table 3.2: Comparison of Prim-based and Kruskal-based heuristics, applied on instance sets R500 and R1000 (B: delay-bound, C: only construction, CV: construction and VND, CGV: construction with GRASP and VND, $\overline{c}$: average final objective values, σ: standard deviations, $t[s]$: median runtimes in seconds).

graphs are even a bit worse, most likely caused by reducing the solution space and thus leading to fewer possibilities to modify solutions in improvement heuristics.

3.13.3 Prim-Based vs. Kruskal-Based Heuristic

Three kinds of tests are performed to compare the Kruskal-based to the Prim-based heuristic [166]:

1. only the deterministic construction heuristic (in the result tables this test is abbreviated with "C")

2. the deterministic construction followed by the VND, using $F_d = 1.5$ ("CV") (see Section 3.4.4

3. the construction with the GRASP extension followed by the VND, using $\alpha = 0.25$, stopping after ten starts without gain and taking the average values of 30 runs ("CGV")

		EU500						EU1000					
		PBH			KBH			PBH			KBH		
B	Test	$\overline{c}$	σ	$t[s]$	$\overline{c}$	σ	$t[s]$	$\overline{c}$	σ	$t[s]$	$\overline{c}$	σ	$t[s]$
0.8	C	19.12	0.44	0.1	**18.03**	0.40	0.1	27.56	0.43	0.7	**25.40**	0.32	0.3
	CV	19.00	0.47	1.4	**17.53**	0.40	2.1	27.15	0.65	22.0	**24.81**	0.32	15.6
0.9	C	19.11	0.41	0.1	**18.04**	0.38	0.1	27.48	0.44	0.7	**25.36**	0.32	0.4
	CV	19.02	0.37	1.6	**17.41**	0.36	2.2	26.97	0.76	20.9	**24.65**	0.29	16.3
1.0	C	19.17	0.49	0.1	**17.83**	0.43	0.1	27.38	0.49	0.8	**25.32**	0.29	0.4
	CV	18.97	0.49	1.9	**17.26**	0.34	2.1	26.80	0.93	16.7	**24.51**	0.31	15.4
1.5	C	18.92	0.48	0.2	**17.46**	0.52	0.1	27.30	0.50	1.0	**24.78**	0.32	0.4
	CV	18.75	0.56	2.9	**16.79**	0.36	2.4	26.71	1.07	23.9	**23.85**	0.26	19.4
2.0	C	18.87	0.60	0.2	**17.37**	0.49	0.1	27.29	0.46	1.1	**24.54**	0.37	0.5
	CV	18.69	0.67	3.3	**16.51**	0.33	2.6	26.33	1.29	34.6	**23.49**	0.23	16.6
3.0	C	18.53	0.59	0.2	**17.02**	0.49	0.1	27.04	0.43	1.2	**24.17**	0.29	0.6
	CV	18.09	0.80	4.0	**16.22**	0.30	2.3	25.69	1.43	48.9	**23.14**	0.24	14.0

Table 3.3: Comparison of Prim-based and Kruskal-based heuristics, applied on Euclidian instance sets EU500 and EU1000 (B: delay-bound, C: only construction, CV: construction and VND, $\overline{c}$: average final objective values, σ: standard deviations, $t[s]$: median runtimes in seconds).

The comparison of only one constructed solution, see test "C" in Table 3.2, indicates that KBH produces usually significantly better solutions than the Prim-inspired algorithm PBH, especially if the delay-constraint is strict. Only in tests with high delay-bounds the Prim-based solution exceeds the Kruskal-based one, but this advantage disappears when also applying the VND. In this test and also when using the GRASP extension ("CV" and "CGV") KBH outperforms PBH with clear statistical significance. In addition we can observe a higher dependence of PBH on the specific edge costs and delays of the instances noticeable in the higher standard deviation values.

Concerning the runtime KBH can compete with PBH and often even beats it, although the administration effort is higher when updating the node information in each step of stage one. We can observe that the runtime is nearly independent of the specified delay-bound B in contrast to PBH, where tight bounds lead to longer runtimes due to the repeated delay-relaxation process, see Table 3.2. The general slight increase of the runtime when raising the bound is caused by the fact that in a preprocessing step all edges with $d_e > B$ are discarded. So tests with lower delay-bounds have to handle less edges.

The results on EU500 and EU1000 instances shown in Table 3.3 clearly demonstrate the superiority of KBH even if using high delay-bounds. At no time even the VND-improved PBH solution reaches the quality of our just constructed spanning tree.

3.13.4 Ranking-Based Multilevel vs. Kruskal-Based Heuristic

Due to RBMH being non-deterministic, 30 runs are performed for every instance and average results are used for comparison with KBH. Results without additional improvement in Table 3.4 show that in general KBH creates much better solutions within shorter runtime. However,

	R500						R1000					
	RBMH			KBH			RBMH			KBH		
B	$\overline{c}$	σ	$\overline{t[s]}$	$\overline{c}$	σ	$\overline{t[s]}$	$\overline{c}$	σ	$\overline{t[s]}$	$\overline{c}$	σ	$\overline{t[s]}$
10	9282	415	0.35	**7087**	335	0.02	13288	593	1.59	**10296**	484	0.06
30	4817	245	1.11	**3768**	382	0.04	7059	253	5.12	**5064**	460	0.15
50	3711	161	1.94	**2824**	232	0.06	5513	174	8.85	**3243**	360	0.24
75	3142	140	3.00	**2048**	255	0.09	4669	133	13.86	**2185**	232	0.35
100	2812	153	3.99	**1695**	250	0.11	4180	128	19.20	**1605**	196	0.46
150	2802	149	4.62	**1007**	145	0.11	4168	126	19.30	**1165**	131	0.38
200	2802	149	4.44	**784**	124	0.10	4168	126	18.92	**1080**	81	0.35

Table 3.4: Comparison of ranking-based multilevel and Kruskal-based heuristics without additional improvement, applied on random instance sets R500 and R1000 (B: delay-bound, $\overline{c}$: average final objective value, σ: standard deviation, $t[s]$: average runtimes in seconds).

	R500						R1000					
	RBMH+VND			KBH+VND			RBMH+VND			KBH+VND		
B	$\overline{c}$	σ	$\overline{t[s]}$	$\overline{c}$	σ	$\overline{t[s]}$	$\overline{c}$	σ	$\overline{t[s]}$	$\overline{c}$	σ	$\overline{t[s]}$
10	4634	225	1.99	**4557**	205	1.45	5290	212	9.33	**5171**	215	7.52
30	**1530**	85	4.42	1554	88	4.37	**1871**	71	23.55	1884	55	20.04
50	**1010**	64	7.99	1042	56	6.22	**1334**	50	33.81	1373	44	32.93
75	**765**	33	10.90	800	37	9.44	**1113**	32	57.75	1146	32	51.42
100	**642**	28	13.64	687	44	12.75	**1038**	12	75.79	1070	32	62.76
150	**547**	11	16.71	587	36	12.25	**1005**	4	74.13	1022	24	57.96
200	**522**	6	13.55	545	27	10.90	**1001**	2	74.58	1008	16	37.65

Table 3.5: Comparison of Ranking-based Multilevel (RBMH) and Kruskal-based (KBH) heuristics with additional improvement (VND), applied on random instance sets R500 and R1000 (B: delay-bound, $\overline{c}$: average final objective value, σ: standard deviation, $t[s]$: average runtimes in seconds).

RBMH is not directly intended to produce low cost spanning trees but rather use edges which have low costs as well as low delay. Therefore, there may be a lot of improvement potential in a solution provided by RBMH. To use this potential to obtain a solution of higher quality we applied the VND described in Section 3.8. The results with this additional improvement in Table 3.5 show that except for very low delay-bounds RBMH typically provides a better starting point for further improvement. Especially for very high delay-bounds the solutions provided by RBMH can be improved significantly. However, RBMH results also show higher runtimes due to the algorithm's higher complexity and longer improvement phases.

3.13.5 GRASP vs. GVNS vs. MMAS

Each result presented in Table 3.6 is derived from 30 runs with a CPU time limit of 300 seconds for each of the 30 instances. Three metaheuristics are included in the comparison: GRASP+VND, GVNS and MMAS (with 5 ants). All preprocessing steps except the search for alternate constrained paths, see Section 3.3.2, have been applied before starting the search re-

		B	6		20		50		100	
		$\alpha/sr/p$	$\bar{c}$	σ	$\bar{c}$	σ	$\bar{c}$	σ	$\bar{c}$	σ
R500	G+V	0.25	8997.3	672	2048.3	87	942.1	37	616.3	14
	GVNS	dynamic	8703.0	620	1961.5	88	901.1	35	601.1	14
		0.05	8701.1	617	1947.2	88	897.7	35	601.1	15
		0.1	8691.7	618	1938.9	89	893.7	34	599.3	14
		0.15	**8691.6**	618	1942.7	88	894.0	34	599.2	14
		0.2	8696.1	618	1947.8	90	896.4	35	599.8	14
	MMAS	0.6	8727.5	616	1937.4	85	891.4	34	598.0	13
		0.7	8726.4	614	1935.1	85	889.5	34	597.1	12
		0.8	8723.4	612	1932.1	84	**887.5**	34	**596.8**	13
		0.9	8722.4	613	**1930.8**	82	891.2	36	602.2	14
		0.95	8720.4	610	1941.3	83	914.9	40	612.2	14
R1000	G+V	0.25	9775.3	487	2473.0	76	1290.3	31	1026.8	9
	GVNS	dynamic	9497.9	486	2377.7	81	1257.4	33	1020.0	7
		0.05	9397.2	476	2346.9	80	1253.4	31	1020.1	7
		0.1	9412.1	480	2353.6	78	1252.5	31	**1019.4**	7
		0.15	9455.2	487	2365.8	80	1254.7	31	1019.6	7
		0.2	9488.3	485	2374.3	80	1257.0	32	1019.9	7
	MMAS	0.5	9385.3	485	2323.4	75	1243.7	28	1021.3	8
		0.6	9378.4	483	2312.4	73	1239.3	27	1020.9	8
		0.7	9376.4	481	**2308.8**	73	**1238.2**	27	1022.1	10
		0.8	9369.1	478	2309.9	74	1241.3	31	1028.8	14
		0.9	**9367.7**	477	2320.9	76	1281.3	46	1042.8	14

Table 3.6: Comparison of GRASP+VND, GVNS and MMAS on random instance sets R500 and R1000 (B: delay-bound, $\bar{c}$: average final objective values, σ: standard deviations; CPU time limit: 300 seconds; best results are printed bold)

ducing the complexity of the instances significantly. When having higher time limits the omitted preprocessing step might be advantageous.

The GRASP+VND approach was outperformed by almost all GVNS and MMAS runs almost independent of the parameter settings, which might be explained by the fact that the GRASP+VND has no memory. It "forgets" the solutions of past iterations and therefore cannot build on already obtained information. The MMAS mostly produces the best solutions probably because it has the most effective memory of all three methods in terms of the pheromone values containing the information of many solutions in one data structure.

A matter of high importance when using a MMAS is the fact that it can typically only exhibit its full effectiveness on a rather high number of iterations because of the longer exploration phase in the beginning [176], see Figure 3.8. When considering the R1000 instances, a full VND improvement of each constructed solution takes much time which leads to a smaller number of iterations within the time limit. The pheromone values therefore have not enough time to converge and the solutions are constructed rather randomly. So a faster local search in one of the two neighborhoods instead of the VND produces in general worse solutions in a single iteration but yields higher quality in the end due to the higher number of iterations. For the smaller R500 instances it is better to use the VND improvement since it is fast enough to allow a sufficient

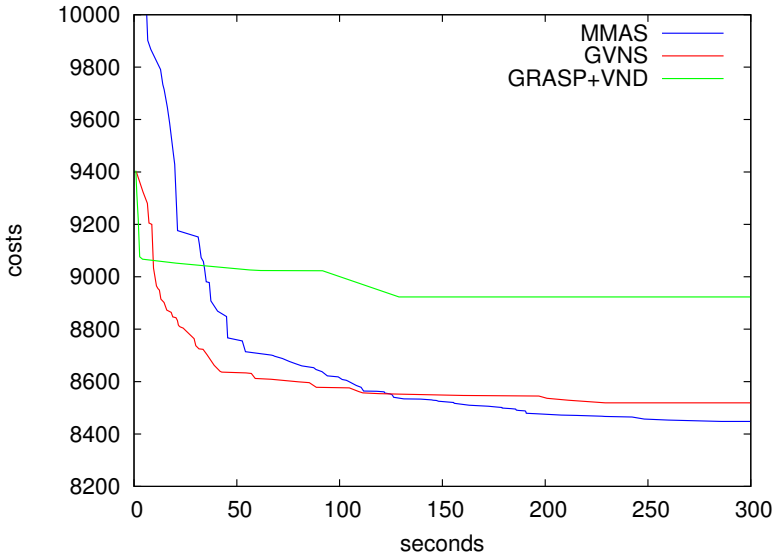


Figure 3.8: Typical run characteristics of all three heuristics applied on a R1000 instance with $B = 6$ with a time limit of 300 seconds.

number of iterations before time is running out.

When considering strict delay-bounds finding feasible solutions is more difficult and therefore the search gets caught in a local optimum more easily. Rather big changes have to be made to catapult the solution to another basin of attraction. Small changes like replacing a single edge to decrease the costs are often not possible because of a lack of residual delay in the nodes. So choosing the wrong way in the beginning of the search has more impact on the quality of the finally best solution than in cases with looser bounds. This fact can be observed in Table 3.6 independent of the heuristic method: The stricter the bound the higher the standard deviations.

A too small pheromone decay coefficient in the MMAS causes a fast convergence of the pheromone values and thus disregards diversification; a too high p-value has the opposite effect: the exploration phase lasts too long especially when having a tight time limit.

Relating to the MMAS results the following observation can be made: When tightening the delay-bound the p-value has to be increased to obtain better results. This behavior is another consequence of the facts mentioned above: When having strict bounds the quality and structure of the produced solutions varies much more and by using a higher p-value a single bad solution has not that much influence on the pheromone values because of the smoother evaporation of already deposited pheromones.

Generally speaking, the whole parameter setting of the MMAS heavily depends on the predefined target runtime. Here the small number of five ants speeds up the evolution of the pheromone values which is necessary for the time limit of 300 seconds. Regarding the final results it is not disadvantageous that possibly worse solutions are allowed to update pheromone values.

Applying Wilcoxon signed-rank tests with confidence level 0.95 to the results for $\alpha = 0.25$

	GVNS	MMAS	MA								
			no duplicate detection			hashing			solution archive		
R500			swap	0.005	0.01	swap	0.005	0.01	swap	0.005	0.01
$B = 6$	**8691.6**	8720.4	8716.3	8712.0	8703.8	8707.2	8705.2	8700.3	8710.3	8706.1	8702.0
20	1938.9	1930.8	1930.1	1929.4	1929.2	1929.9	1929.7	**1928.3**	1933.4	1933.8	1933.2
50	893.7	887.5	886.2	**885.9**	886.5	886.1	886.4	886.7	887.2	887.8	888.7
100	599.2	596.8	596.1	596.0	596.0	**595.9**	596.1	596.1	596.3	596.5	596.8
R1000			swap	0.001	0.005	swap	0.001	0.005	swap	0.001	0.005
$B = 6$	9397.2	9367.7	9367.6	9393.6	9366.8	9363.6	9388.1	9366.8	**9353.6**	9369.7	9354.7
20	2346.9	2308.8	**2307.2**	2313.5	2322.8	**2307.2**	2314.2	2322.2	2315.9	2320.9	2333.9
50	1252.5	1238.2	1238.1	1240.9	1248.0	**1237.6**	1241.2	1248.2	1240.7	1242.9	1250.0
100	**1019.4**	1020.9	1020.8	1021.7	1023.1	1021.0	1021.9	1023.5	1021.2	1021.9	1023.3

Table 3.7: Comparison of GVNS, MMAS, and MA with different methods of duplicate detection and mutation operators (swap, two values for p_m); values are average tree costs, B: delay-bound, time limit: 300 sec., best results are printed bold.

		no det.	hashing			solution archive		
Set	B	$\overline{I} = \overline{O}$	$\overline{I}$	$\overline{D\,[\%]}$	$\overline{O}$	$\overline{I} = \overline{O}$	$\overline{D\,[\%]}$	$\overline{GB}$
R500	6	**5018**	4774	33	3187	4993	61	0.30
	20	**2365**	2260	20	1792	1494	19	0.30
	50	**1211**	1185	10	1056	924	7	0.45
	100	**804**	797	5	747	678	4	0.65
R1000	6	4251	4309	17	3553	**4820**	27	0.70
	20	**1783**	1789	3	1732	1485	2	0.70
	50	**1020**	1017	0	1014	958	0	1.00
	100	**660**	662	0	659	650	0	1.35

Table 3.8: Statistics; I: number of iterations, D: detected duplicates in percent, O: kept offsprings, GB: approx. memory consumption of archive, best results are printed bold.

(GRASP), $sr = 0.1/0.05$ (GVNS on R500/R1000) and $p = 0.8/0.7$ (MMAS on R500/R1000) yields the following error probabilities p_{err}: GVNS and MMAS produce better results than GRASP with $p_{\text{err}} \ll 0.01$ for all bounds. MMAS performs better than GVNS on R500 instances with $p_{\text{err}} = 1$ ($B = 6$), $p_{\text{err}} \ll 0.01$ ($B = 20$), $p_{\text{err}} \ll 0.01$ ($B = 50, 100$) and on R1000 instances with $p_{\text{err}} \ll 0.01$ ($B = 6$), $p_{\text{err}} \ll 0.01$ ($B = 20, 50$), $p_{\text{err}} = 1$ ($B = 100$).

3.13.6 Memetic Algorithm

Due to the in-determinism of the MA, 30 runs are performed for each instance and setting. We use a time limit of 300 seconds for each run. All preprocessing methods presented in Section 3.3 except the most time-consuming arbitrary-path test are applied to the instances reducing the number of edges significantly. We compare the MA to MMAS and GVNS. In preliminary tests promising parameter values for the MA have been identified: the population size is set to 50, for R500 instances we set the mutation rate to $p_m \in \{0.005, 0.01\}$ and perform full VND for local improvement, for R1000 instances $p_m \in \{0.001, 0.005\}$ and only a single neighborhood is

randomly chosen (Edge-Replace with higher probability $p = 0.7$). The replacement parameter $r \in \{10, ..., 40\}$ is dynamically adapted at runtime: initially $r = 10$; if a new best solution is found, r is decreased by 2, and if the search stagnates, r is increased by 2. The higher the parameter r the higher the diversity in the population and the other way round. So if the algorithm should concentrate on intensification of the best solutions, r is automatically lowered while if it gets stuck in a local optimum diversity is increased again.

Experimental results are shown in Table 3.7. In most cases the MA outperforms existing methods except for two settings where GVNS is still leading. A surprising result is that the use of the trie-based solution archive in general is less beneficial than expected. For the considered problem and MA, the overhead of maintaining the archive is too high even though the operations on it are rather efficient. This can be clearly seen in Table 3.8 in the average numbers of iterations within the time limit. Here the variant with duplicate detection by hash values yields more kept offsprings after discarding detected duplicates (but one has to consider that only the current population is checked for duplicates). Only for rather low delay-bounds the archive is able to yield better results, i.e. both a higher number of new offsprings and a finally higher solution quality. The number of revisits is in general much higher for low delay-bounds since the solution space is smaller and the probability of getting stuck in a local optimum after local improvement is higher. Immediately after mutation the duplicate rate is in general rather low provided that the mutation operator is not too limited. Additionally, it can be observed that higher mutation rates are more beneficial in cases of tight bounds, see also the results in Table 3.6. This can be explained again by the fact that it is easier to get stuck in a local optimum requiring a substantial modification of the solution to reach new basins of attraction. In case of loose bounds small changes are enough to escape local optima making a simple swap move in most cases the best choice. Furthermore, the higher the mutation rate, the higher the diversity in the population and the smaller the probability of a revisit. So the solution archive is more effective when having low mutation rates. Generally, most of the time is spent with local improvement and if using full VND the number of achieved iterations further decreases. If only single neighborhoods are examined more iterations are possible and higher mutation rates to cover more areas of the search space are beneficial. Tests without local improvement substantially increased the number of iterations but lead to far worse solution quality.

3.14 Future Work

The idea of the ranking score to provide some kind of edge quality measurement could be further extended by investigating different variants of the ranking score formula. Weighting factors could be used to control the influence of costs versus delays which may be dynamically adapted throughout the solution process: Priority on low costs intensifies the search whereas attaching importance to low delays leads to more feasible possibilities in the remaining solution construction or improvement.

The VND and GVNS methods could be improved by additional neighborhood structures maybe based on new solution representations to better diversify the search and therefore find new feasible solutions. A further idea would be to generalize the Edge-Replace neighborhood to replace k edges in one move: first, a set of k probably most expensive edges is removed from the

solution, and then the $k + 1$ components are reconnected as cheap as possible. To achieve this we could apply e.g. the idea of the Kruskal-based heuristic or delay-constrained shortest paths or even reconnect the separated components optimally for small values of k.

In the memetic algorithm alternative more sophisticated problem-specific operators would be interesting, to pursue for example a recombination operator based on path relinking: we conjecture that any feasible RDCMST solution can be transformed to any other feasible solution by only applying a series of simple edge exchanges guaranteeing that all intermediate solutions are feasible, too. However, a proof of this assumption is still missing. Assuming the correctness of this conjecture, we apply it by creating a modification path linking together two feasible parent solution and choose the best intermediate solution as offspring. Alternatively, we could determine the optimal "offspring" tree in the subgraph only consisting of the edges from the two parent solutions.

Since in the ant colony and memetic algorithms local improvement of decoded solutions consumes most of the runtime, we should think about an adaptation of the decoding method to improve the quality of just constructed solutions. Then maybe less improvement is necessary leading to a higher number of iterations within a given time limit. One way to possibly achieve this is similar to the heuristic in [111] and iteratively adds a delay-constrained cheapest path, see Section 3.3.2, to an unconnected node using the corresponding delay in the encoded array as delay-bound. However, we have to assure that the solution construction does not consume too much time leading again to fewer iterations. Thus, we could also think about heuristic methods to solve the delay-constrained shortest path problem.

Regarding multilevel approaches, the ranking-based construction heuristic could be extended towards an iterated multilevel approach in which obtained solutions are iteratively re-coarsened and refined. Including some kind of diversification mechanism in the coarsening phase such an improvement method could be interesting in comparison to the leading approaches. Additionally, in each step of the refinement phase usually some sort of local search or VND is used for further improvement.

Finally, we want to further analyze the integration of solution archives in heuristics for the RDCMST problem, improve the transformation of revisited solutions to more promising ones by considering the solution quality, see e.g. [71] for a branch-and-bound extension, and decrease the time and space overhead caused by the archive. Since different delay arrays may decode to the same tree, a new unvisited delay array derived from the archive may again lead to an already visited solution. By using more sophisticated transformation mechanisms we can hopefully handle this situation.

CHAPTER 4

Rooted Delay-Constrained Steiner Tree Problem

This chapter discusses several exact mixed integer programming approaches for solving the *rooted delay-constrained Steiner tree (RDCST) problem*. Section 4.1 formally defines the considered problem and Section 4.2 mentions previous related work. In Section 4.3 some extensions to the reduction techniques for the RDCMST problem from Section 3.3 are described. Sections 4.4, 4.5, and 4.6 discuss existing and revised formulations for the RDCST problem: A model based on Miller-Tucker-Zemlin subtour elimination inequalities, a path model containing an exponential number of variables, and a multi-commodity flow model. Furthermore, in Section 4.7 we present a formulation based on infeasible path cuts extended by a set of strengthening valid inequalities. Section 4.8 shows how to transform the input graph to a so-called layered graph which is then utilized in a strong formulation in Section 4.9. We compare all the formulations from a polyhedral point of view in Section 4.10 and in experimental tests in Section 4.11. Finally, Section 4.12 mentions open problems and possible future research directions. Some parts of this chapter are based on the published articles [113–115, 163, 165].

4.1 Problem Definition

The RDCST problem is a generalization of the RDCMST problem discussed in Chapter 3 since we now do not have to connect all nodes to the root node but only a given subset. The rest of the nodes can optionally be included in a solution as intermediate relay nodes.

More formally, we are given an undirected graph $G = (V, E)$ with node set V, a fixed root node $s \in V$, set $R \subseteq V \setminus \{s\}$ of terminal or required nodes, set $S = V \setminus (R \cup \{s\})$ of potential Steiner nodes, edge set E, a cost function $c : E \to \mathbb{Z}_0^+$, a delay function $d : E \to \mathbb{Z}^+$, and a delay bound $B \in \mathbb{Z}^+$. A feasible solution to the RDCST problem is a Steiner tree $T =$

$(V', E'),\ s \in V',\ R \subset V' \subseteq V,\ E' \subseteq E$, satisfying the delay-constraints

$$d_v^T = \sum_{e \in P_T(s,v)} d_e \leq B,\ \forall v \in R. \tag{4.1}$$

$P_T(s, v)$ denotes the unique path from the specified root node s to terminal node $v \in R$ in Steiner tree T and d_v^T the corresponding total delay on this path. Further, we define the cost function

$$c_T = \sum_{e \in E'} c_e, \tag{4.2}$$

summing up the cost values of all edges in a solution T. An optimal solution T^* to the RDCST problem is a feasible solution with minimal total edge costs, i.e. $c_{T^*} \leq c_T,\ \forall T$.

Similarly to the RDCMST problem, we define a directed variant of this problem on graph $G' = (V, A)$ with arc set $A = \{(s, v) : \{s, v\} \in E\} \cup \{(u, v), (v, u) : \{u, v\} \in E,\ u, v \neq s\}$ consisting of two opposite arcs for each edge in graph G except for edges incident to root node s, for which we include only the corresponding arc going out from s. A feasible solution to the directed variant is a Steiner arborescence $T' = (V', A'),\ s \in V',\ R \subset V' \subseteq V,\ A' \subset A$, directed out of root node s. It can be easily seen that each feasible Steiner tree T bijectively corresponds to a feasible Steiner arborescence T'.

The $\mathcal{NP}$-hardness of the RDCST problem can be shown in several ways by reduction from $\mathcal{NP}$-hard special cases, e.g. the RDCMST problem, the Steiner tree problem on graphs [44, 89, 90, 123], where $B = \infty$, and the *Hop-Constrained Steiner Tree (HCST) Problem* [181], where $d_e = 1,\ \forall e \in E$, respectively.

A lower bound to the optimal cost value is provided by a minimal-cost Steiner tree T^{l} without considering the delay values. If such a tree T^{l} is feasible for the RDCST problem, i.e. satisfies the delay-constraints, then T^{l} also is an optimal solution for it. However, finding an optimal Steiner tree T^{l} is $\mathcal{NP}$-hard. Similarly to the RDCMST problem, if a feasible solution exists we are always able to construct a trivial feasible Steiner tree T^{u} via the shortest-delay-paths from root s to all terminal nodes $v \in R$ without considering edge costs computed e.g. by Dijkstra's polynomial time algorithm for the single-source shortest path problem [41]. Again, if T^{u} exceeds the given delay-bound for any terminal node there is no feasible solution for the RDCST problem. Additionally, all feasible solutions for the RDCMST problem on G are feasible solutions for the RDCST problem on G. However, these trees possibly contain Steiner nodes as redundant leaves in the tree which can safely be pruned in linear time usually reducing the tree costs.

Instead of using $c_{\{u,v\}}$ and $d_{\{u,v\}}$ to denote cost and delay values assigned to edge $\{u, v\} \in E$, we use the better readable notation c_{uv} and d_{uv}, respectively. The same holds for arcs $(u, v) \in A$ in directed graph G'. Variable $d_v,\ v \in V$, refers to the node delay with respect to one specific tree T. In case of multiple solutions the considered tree is explicitly included in the notation, i.e. $d_v^T,\ v \in V$.

4.2 Related Work

Many different names can be found in the literature for the RDCST problem: Delays are often interpreted as distances [68], leading to the name *Distance-Constrained Steiner Tree Problem*. The widely used term *Multicast Routing Problem* is application-oriented and rather unspecific concerning the included constraints. The prefix "source-based" usually describes the situation when the source node is aware of the whole network structure, whereas in distributed problem variants a node only knows information about its neighbors. Extensive surveys about optimization problems with different quality of service (QoS) constraints can be found in [135, 136, 156].

The original Steiner tree problem was introduced by Gilbert and Pollak in 1968 [56] and is defined in the Euclidian plane using any points in this plane as potential Steiner nodes. The Steiner tree problem on graphs was proposed by Dreyfus and Wagner in 1971 [44], together with an exact enumeration algorithm with runtime exponential in the number of terminal nodes. Kou, Markowsky, and Bernan [105] present a widely-known construction heuristic: First, a closure graph $G^c = (R, E')$ is built where an edge $\{u, v\},\ u, v \in R$, represents the shortest path from terminal u to terminal v with minimum cost c_{uv}. After finding a minimum spanning tree in G^c, all edges of this tree are expanded to their original paths. The final Steiner tree is obtained by again deriving a minimum spanning tree in this path graph. This fast 2-approximation algorithm has been the basic concept for numerous algorithms in literature, e.g. [102,187]. Another famous and frequently applied construction concept for Steiner trees is presented by Takahashi and Matsuyama [177]. They start with a single terminal node and in each step attach a not yet connected required node to the partial tree in the cheapest possible way. This simple heuristic is also 2-approximative and runs in $\mathcal{O}(|R| \cdot |V|^2)$ time using Dijkstra's shortest path algorithm [41]. Further improved approximation algorithms have been presented, e.g. with a ratio of about 1.55 in [158].

Solving the Steiner tree problem to proven optimality is subject to numerous articles in literature. Thus, we only want to mention some basic and influential work here. Wong [191] presents a dual ascent approach, Goemans et al. [62] compare several MIP formulations from a polyhedral point of view, Chopra et al. [27] provide detailed polyhedral analyses and some facet-defining inequalities, Koch et al. [100] apply many reduction rules before they solve the problem by a classical branch-and-cut approach, and Aragao et al. [142] propose dual heuristics to accelerate a branch-and-cut and branch-and-ascent approach. Polzin et al. [143] also provide a polyhedral comparison of Steiner tree relaxations and additionally introduce a new formulation based on so-called *common flows* yielding the so-far best LP bounds.

Since the introduction of the Steiner tree problem on graphs many variants of this problem with additional constraints emerged in literature. Practical applications, e.g. multimedia content distribution and VoIP, ask for QoS constraints such as limiting the communication delay between server and clients. Therefore, two problem variants particularly increased in popularity, the already mentioned RDCST problem and the *hop-constrained Steiner tree (HCST) problem* where $d_e = 1,\ \forall e \in E$, modeling the fact that in many cases only the number of distribution and routing nodes in an end-to-end connection is relevant.

The RDCST problem was introduced and proven to be $\mathcal{NP}$-hard by Kompella et al. [102, 103] who also presented a construction heuristic based on the Steiner tree heuristic by Kou et

al. [105]. To guarantee the satisfaction of the delay-constraints the closure graph in the first step is built using delay-constrained cheapest paths, see Section 3.3.2. Second, a Prim-based heuristic is applied to the closure graph to derive a feasible RDCST solution. A distributed variant of this algorithm can be found in [104]. A construction heuristic based on the idea by Takahashi et al. [177] can be found in [111] where again instead of shortest paths delay-constrained cheapest paths are utilized to satisfy the delay-constraints. Further construction heuristics can be found in [6, 76, 139, 196].

There are lots of recent publications for the RDCST problem presenting metaheuristic approaches, such as a genetic algorithm [195], tabu-search [173], GRASP [174, 192], path-relinking [55], and variable neighborhood descent (VND) [150]. A hybrid algorithm in [193] combines scatter search with tabu-search, VND, and path-relinking. An approach applying the multilevel refinement paradigm described in Section 2.2.4 and 3.5 in an iterative way can be found in the master's thesis of Seidl [169]. Construction and local search heuristics for the HCST problem have been described by Voß [181], Fernandes et al. [49], and Gouveia et al. [69].

Manyem and Stallmann [126] showed that the RDCST and HCST problem are not in APX even when considering the spanning tree variants, see Section 3.2.

Exact methods for the RDCST and related problems are dominated by MIP methods. Many basic results for comparing different MIP formulations for tree problems are described by Magnanti and Wolsey in [124]. Gouveia [64] proposed a MIP formulation based on Miller-Tucker-Zemlin (MTZ) inequalities [129] for the HCMST problem – the spanning tree variant of the HCST problem. In 1996, Gouveia [65] presents a multi-commodity flow (MCF) formulation and different Lagrangian relaxations for the HCMST problem and mentions a possible problem generalization with arbitrary edge distances which is now known as the RDCMST problem, see Chapter 3. Leggieri et al. [111] adapt the MTZ formulation in [64] to the RDCST problem and present a compact extended node-based formulation using lifted MTZ inequalities yielding, however, rather weak LP relaxation bounds, see Section 4.4. Hence, they further tightened the formulation in [110] by adding directed connection inequalities in a typical branch-and-cut way. Further MIP approaches for the RDCMST problem have been proposed in [68] and described in detail in Section 3.2. Gouveia et al. [70] extended the layered graph approach in [68] for the HCMST problem, such that not the hop-constrained shortest path problems for each terminal but the whole HCMST problem is modeled on a single layered graph which reduces to solving the classical Steiner tree problem on this graph without additional constraints. Using a classical directed cut formulation on this layered graph yields an LP bound which is at least as tight as the one of all other known formulations for the HCMST problem: When modeling the hop-constrained shortest path problems in separated layered graphs, in a fractional solution the same arc can be in different hop positions in different paths, which is not possible for the formulation in [70] modeling the whole problem in one layered graph.

Recently, Leitner, Raidl, and the author of this thesis [113–115] proposed a successful branch-and-price approach based on a natural path formulation for the RDCST problem. The column generation method by Gouveia et al. [68] for the RDCMST problem strongly suffers from degeneracy. Leitner et al. improved this situation by proposing a generally-applicable efficient approach based on alternative dual-optimal solutions to stabilize column generation. This acceleration technique is able to choose "more meaningful" delay-constrained paths with nega-

tive reduced costs in the pricing subproblem, in the sense that finally a much lower number of columns is needed in the path model to prove optimality.

4.3 Preprocessing

Obviously, we can apply all preprocessing techniques for the RDCMST problem to the RDCST problem, too, see Section 3.3. By additionally considering set S of potential Steiner nodes, we can possibly further reduce input graph G, cf. [111]. Many of these methods are adopted from the Steiner tree problem, cf. [100] for an extensive summary on reduction techniques. However, in contrast to the Steiner tree problem we have to give additional consideration on the edge delays. Therefore, not all preprocessing rules for the Steiner tree problem are feasible for the RDCST problem, and most methods have to be adapted to respect the delay-constraints. But we may also benefit from delay-constraints by being able to eliminate optional nodes that cannot act as relay nodes in any feasible solution. More formally, let $d_v^{\min}$ be the delay of the shortest-delay-path from root s to a node $v \in V$, cf. Section 3.3.1. Furthermore, let

$$d_v^R := \min_{P(v,u),\ u \in R} \sum_{e \in P(v,u)} d_e,\ \forall v \in S, \tag{4.3}$$

be the delay of the shortest-delay-path from a potential Steiner node $v \in S$ to the "nearest" terminal node $u \in R$. Next, we define the maximal path-delay to a node $v \in V$ by

$$d_v^{\max} = \begin{cases} 0 & \text{if } v = s \\ B & \text{if } v \in R \\ B - d_v^R & \text{if } v \in S \end{cases}. \tag{4.4}$$

Thus, if $v \in S$ and

$$d_v^{\min} > d_v^{\max}, \tag{4.5}$$

then node v can be eliminated together with all incident edges. It is important for the computation of d_v^R to only consider paths $P(v,u),\ u \in R$, which do not include root s since it makes no sense to use an optional node as relay node to a terminal node if the root node is nearer to this required node. Therefore, we compute the d_v^R values in the directed graph G' where the root does not have incoming arcs.

4.4 Miller-Tucker-Zemlin Formulation

The following formulation has originally been presented by Gouveia [64] for the HCMST problem and by Leggieri et al. [111] for the RDCST problem. The model is based on MTZ inequalities to eliminate subtours and at the same time guarantee a feasible solution regarding the delay-constraints. Here, we only discuss a basic variant of the finally strengthened model in [111] since we use this simpler model in the polyhedral comparison in Section 4.10. However, for experimental comparisons in Section 4.11 we use the tightest formulation from [111].

Leggieri et al. [111] introduce a dummy node 0 and extend arc set A to $\bar{A} = A \cup \{(0,v) : v \in S \cup \{s\}\}$ where all new arcs get assigned zero costs and delays. A feasible solution for

the RDCST problem is defined as spanning arborescence rooted in 0 where only root s and all unused optional nodes are allowed to connect directly to dummy node 0. Arcs are modeled by binary decision variables $x_{uv},\ \forall (u,v) \in \bar{A}$, and real-valued variables $\delta_v,\ \forall v \in V$, represent the total delay of path $P(s,v)$ in a solution. Model MTZ is defined as follows:

$$\min \quad \sum_{(u,v)\in A} c_{uv} x_{uv} \tag{4.6}$$

$$\text{s.t.} \quad \sum_{(u,v)\in \bar{A}} x_{uv} = 1 \qquad \forall v \in V \tag{4.7}$$

$$x_{0v} + x_{uv} + x_{vu} \leq 1 \qquad \forall v \in S,\ \forall (u,v) \in A \tag{4.8}$$

$$\delta_u + d_{uv} x_{uv} \leq \delta_v + (B-1)(1 - x_{uv}) \qquad \forall (u,v) \in A \tag{4.9}$$

$$\delta_s = 0 \tag{4.10}$$

$$\delta_v \in [1, B] \tag{4.11}$$

$$x_{uv} \in \{0,1\} \qquad \forall (u,v) \in \bar{A} \tag{4.12}$$

Equalities (4.7) force the indegree to each node to exactly 1 which is feasible also for root s and unused optional nodes due to dummy node 0. Inequalities (4.8) state that if a potential Steiner node is linked to dummy node 0 then it has to be a leaf. MTZ inequalities (4.9) eliminate subtours by assigning increasing delays to nodes along a path starting from root s, i.e. $\delta_u + d_{uv} \leq \delta_v$ if $x_{uv} = 1$. Furthermore, by limiting the node-delay values by inequalities (4.10) and (4.11), the MTZ inequalities ensure compliance with delay-constraints. MTZ_{LP} denotes the LP relaxation of MTZ.

4.5 Path Formulation

Gouveia et al. [68] propose a path formulation for the RDCST problem with a set of exponentially many path variables. Recently, we [113–115] reused this formulation to introduce a stabilized branch-and-price approach.

Consider binary arc variables $x_{uv},\ \forall (u,v) \in A$, and path variables $\lambda_p \in \{0,1\},\ \forall p \in \mathcal{P}$, where $\mathcal{P} = \bigcup_{v\in R} \mathcal{P}_v$, and $\mathcal{P}_v \subseteq 2^A$ is the set of all feasible paths from root s to terminal $v \in R$ represented by their set of arcs; $\sum_{(u,v)\in p} d_{uv} \leq B$ must hold for each path $p \in \mathcal{P}$. Model $PATH$ is defined as follows:

$$\min \quad \sum_{(u,v)\in A} c_{uv} x_{uv} \tag{4.13}$$

$$\text{s.t.} \quad \sum_{p\in \mathcal{P}_v} \lambda_p \geq 1 \qquad \forall v \in R \tag{4.14}$$

$$\sum_{p\in \mathcal{P}_v | (u,v)\in p} \lambda_p \leq x_{uv} \qquad \forall v \in R,\ \forall (u,v) \in A \tag{4.15}$$

$$\sum_{(u,v)\in A} x_{uv} \leq 1 \qquad \forall v \in V \setminus \{s\} \tag{4.16}$$

$$\lambda_p \geq 0 \qquad \forall p \in \mathcal{P} \tag{4.17}$$

$$x_{uv} \in \{0,1\} \qquad \forall (u,v) \in A \tag{4.18}$$

Inequalities (4.14) ensure that at least one path is realized for each terminal node, while inequalities (4.15) link path variables to arcs used by them. Inequalities (4.16) restrict the indegree of each node and thus together with inequalities (4.14) and (4.15) ensure that the directed solution is an arborescence rooted in s. Given strictly positive edge costs, removing inequalities (4.16) would also yield a valid model. We did nevertheless include them to stay consistent with the model by Gouveia et al. [68]. Further note that only lower bounds are given for variables λ_p, $\forall p \in \mathcal{P}$, in inequalities (4.17). These variables will become automatically integral due to the remaining inequalities: In case of integral arc variables, inequalities (4.15) bound the λ variables from above and ensure together with inequalities (4.14) that there exists a solution with integral path variables. $PATH_{\text{LP}}$ denotes the LP relaxation of $PATH$.

The number of feasible paths and hence the total number of variables of the model $PATH$ may be exponentially large for each terminal. Thus, we cannot solve it directly, but apply branch-and-price, i.e. embed delayed column generation in a branch-and-bound approach, cf. [11, 39]. For each node of the branch-and-bound tree we then need to solve the *restricted master problem* (RMP). This RMP is defined by replacing the integrality constraints (4.18) by $x_{uv} \geq 0$, $\forall (u,v) \in A$, and additionally considering only a small subset $\emptyset \neq \tilde{\mathcal{P}}_v \subseteq \mathcal{P}_v$, $\forall v \in R$, of path variables.

When solving a node of the branch-and-price tree by column generation, we need to repeatedly identify path variables with negative reduced costs and add at least one of them to the RMP, which in turn needs to be resolved. This process is repeated until no more variables with negative reduced costs exist. In order to prove that no more negative reduced cost variables do exist, we need to compute the path variable yielding minimal reduced costs. This pricing subproblem can be solved by finding a delay-constrained cheapest path, see Section 3.3.2, from root s to each terminal $v \in R$ in a support graph with non-negative arc costs corresponding to dual variable values.

4.6 Multi-Commodity Flow Formulation

We adopted the MCF formulation from Gouveia [65] for the HCMST problem to the RDCST problem. Additionally and independently, the following MCF model is presented in a recent article [111] in 2011. We use binary decision variables x_{uv}, $\forall (u,v) \in A$. Furthermore, real-valued flow variables f_{uv}^w, $\forall (u,v) \in A$, $\forall w \in R$, denote the flow on arc (u,v) from root s to terminal w. Model MCF is defined as follows:

$$\min \quad \sum_{(u,v)\in A} c_{uv} x_{uv} \tag{4.19}$$

$$\text{s.t.} \quad \sum_{(s,v)\in A} f_{sv}^w = 1 \qquad \forall w \in R \tag{4.20}$$

$$\sum_{(u,v)\in A} f_{uv}^w - \sum_{(v,u)\in A} f_{vu}^w = 0 \qquad \forall v \in V \setminus \{s,w\},\ \forall w \in R \tag{4.21}$$

$$\sum_{(v,w)\in A} f_{vw}^{w} - \sum_{(w,v)\in A} f_{wv}^{w} = 1 \qquad \forall w \in R \qquad (4.22)$$

$$\sum_{(u,v)\in A} d_{uv} f_{uv}^{w} \leq B \qquad \forall w \in R \qquad (4.23)$$

$$0 \leq f_{uv}^{w} \leq x_{uv} \qquad \forall (u,v) \in A,\ \forall w \in R \qquad (4.24)$$

$$x_{uv} \in \{0,1\} \qquad \forall (u,v) \in A \qquad (4.25)$$

Classical flow conservation constraints (4.20), (4.21) and (4.22) describe the flow of one commodity for each terminal $w \in R$ originating in root s, possibly passing any nodes in $V \setminus \{s, w\}$, and ending in target node w, respectively. Inequalities (4.23) add up the delays on the path to a terminal and bound these path-delays by B. Finally, linking inequalities (4.24) connect flow and arc variables. MCF_{LP} denotes the LP relaxation of MCF.

Note that, providing edge costs are strictly positive, objective (4.19) together with flow constraints (4.20)–(4.22), (4.24) and (4.25) describe optimal Steiner trees in directed graphs, cf. [124]. In principle, MCF models can also be formulated using undirected edge variables. However, Magnanti et al. [124] and Gouveia [65] show for the Steiner tree and the HCMST problem, respectively, that in this case $\mathcal{O}(|R|^2 \cdot |E|)$ constraints are needed to obtain the same strength as the directed variant with only $\mathcal{O}(|R| \cdot |A|)$ constraints.

4.7 Path-Cut Formulation

Although flow formulations usually provide rather tight LP bounds, the huge number of flow variables often leads to a slow and memory-intensive solving process, which can be clearly seen in the experimental results in Section 4.11. Thus, it is common practice for tree problems to consider a formulation only defined on arc variables but in general with an exponential number of inequalities guaranteeing connectivity. For tree problems without further constraints, e.g. the Steiner tree problem, the equivalence of a multi-commodity flow and a directed connection cut formulation can be shown by applying the max-flow-min-cut theorem [5]. In case of the RDCST problem it is not obvious how to model the delay-constraints only with arc variables. Here, we use a rather general approach by forbidding all simple paths with a delay higher than bound B. Let $P(v_1, v_k) = \{(v_i, v_{i+1}) : v_i \in V,\ i = 1, ..., k-1,\ v_i \neq v_j,\ i \neq j\}$ denote a simple path consisting of k different nodes and $k-1$ arcs. A path P is called infeasible for the RDCST problem if it cannot occur in any feasible solution, i.e. a solution including P violates the delay-bound. Let $\mathcal{P}_{\mathrm{inf}}$ denote the set of all infeasible paths P. Much work on infeasible paths has been published for the traveling salesman problem (TSP) with time windows, cf. [7, 8, 37], and the vehicle routing problem (VRP) with time windows, cf. [96]. However, to the best of the author's knowledge this topic has not yet been discussed for MIP approaches on constrained tree problems.

Similarly to model MCF, we use binary decision variables $x_{uv},\ \forall (u,v) \in A$. Model PC is

then defined as follows:

$$\min \quad \sum_{(u,v)\in A} c_{uv}x_{uv} \tag{4.26}$$

$$\text{s.t.} \quad \sum_{(u,v)\in A,\ u\in W,\ v\notin W} x_{uv} \geq 1 \qquad \forall W \subset V,\ s \in W,\ \overline{W} \cap R \neq \emptyset \tag{4.27}$$

$$\sum_{i=1}^{k-1} x_{v_i v_{i+1}} \leq k-2 \qquad \forall P(v_1, v_k) \in \mathcal{P}_{\text{inf}} \tag{4.28}$$

$$x_{uv} \in \{0,1\} \qquad \forall (u,v) \in A \tag{4.29}$$

Inequalities (4.27) guarantee connectivity by stating that at least one arc has to cross any cut partitioning node set V into a set W containing the source and a set $\overline{W} = V \backslash W$ including at least one terminal node. Inequalities (4.28) assure in a rather abstract way that the delay-constraints are satisfied. Both sets have in general an exponential number of inequalities. Thus, we add them in the typical branch-and-cut way, see Section 4.7.2 for according separation methods. PC_{LP} denotes the LP relaxation of PC.

To provide a "hot-start" of the branch-and-cut algorithm, we a priori add some inequalities which are indeed included in the sets above. Inequality (4.27) with $W = \{s\}$ ensures at least one arc going out from the root, and a subset of the subtour elimination inequalities [124] (equivalent to inequalities (4.27)) with two-node-sets prevents cycles of length two:

$$\sum_{(s,v)\in A} x_{sv} \geq 1 \tag{4.30}$$

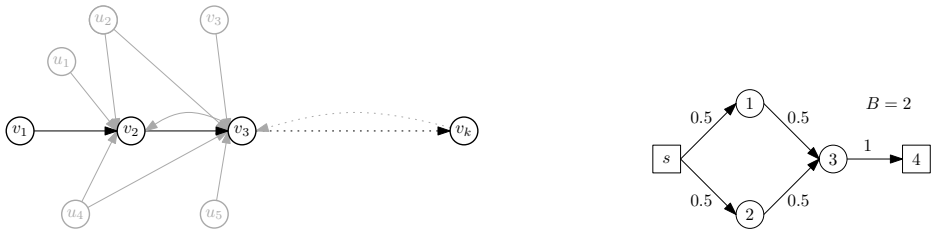
$$x_{uv} + x_{vu} \leq 1 \qquad \forall \{u,v\} \in E \tag{4.31}$$

Furthermore, inequalities (4.32) guarantee exactly one incoming arc for each terminal node. Provided that the objective function only has non-negative cost coefficients – as it is in our case – these in-degree constraints are also subsets of inequalities (4.27).

$$\sum_{(u,v)\in A} x_{uv} = 1 \qquad \forall v \in R \tag{4.32}$$

4.7.1 Valid Inequalities

Ascheuer et al. [7, 8] and Kallehauge et al. [96] argue that the general infeasible path inequalities (4.28) can be rather weak for the TSP with time windows and the VRP with time windows, respectively, and therefore propose various lifted variants of them, e.g. the so-called *tournament inequalities*. However, their stronger inequalities heavily rely on the fact that in these routing problems each node has exactly one incoming and one outgoing arc in a feasible solution. For the RDCST problem we can only assume that the indegree of a node is at most one. Nevertheless, we provide one set of lifted infeasible path inequalities: Let $V_P^i = \{u \in V : (u,v_i) \in A,\ u \neq v_{i-1}, v_{i+1},\ (u,v_i) \cup P(v_i,v_k) \in \mathcal{P}_{\text{inf}}\},\ \forall i \in \{2,...,k-1\}$, be the sets of nodes which



(a) (b)

Figure 4.1: (a) Illustration of lifted infeasible path inequalities. We assume that all paths starting in nodes $v_1, u_i,\ i = 1, ..., 5$, and ending in v_k are infeasible (without considering the additional reverse arcs). (b) The solution with $d_a = 1,\ \forall a \in A$, and $B = 2$ is feasible for PC_{LP} (arc labels denote variable values of the LP solution), but violates inequalities (4.33), e.g. for infeasible path $P = \{(s, 1), (1, 3), (3, 4)\}$ since $x_{s1} + x_{13} + x_{34} + x_{23} = 2.5 > |P| - 1 = 2$. Squared nodes denote terminal nodes.

form together with a sub-path of P again infeasible paths. The lifted inequalities are defined as follows:

$$\sum_{i=1}^{k-1} x_{v_i v_{i+1}} + \sum_{i=2}^{k-1} \sum_{u \in V_P^i} x_{uv_i} + \sum_{i=2}^{k-1} x_{v_{i+1} v_i} \leq k - 2 \qquad \forall P(v_1, v_k) \in \mathcal{P}_{\text{inf}} \tag{4.33}$$

Figure 4.1a shows an illustration of inequalities (4.33). Note, that the first sum and the right side of inequalities (4.33) are identical to inequalities (4.28). Thus, by including additional arc variables on the left side the constraints get tighter, see Fig. 4.1b for an example. It can be easily seen that due to the paths' infeasibility and the limited node's indegree we can choose at most $k - 2$ arcs at the same time for a feasible solution. Note that these liftings are also applicable to many other tree problems since they only require the representation of the solution as arborescence and a maximal node indegree of one.

A subset of inequalities (4.33) considering infeasible paths of length two can optionally be added a priori to model PC:

$$\sum_{u \in V_P^1} x_{uv_1} + x_{v_1 v_2} + x_{v_2 v_1} \leq 1 \qquad \forall (v_1, v_2) \in A \tag{4.34}$$

Obviously, inequalities (4.34) include inequalities (4.31).

4.7.2 Separation Methods

Here, we discuss details about used separation methods for inequalities (4.27), (4.28) and (4.33). In our implementation directed connection cuts (4.27) are separated first until no violated inequality can be found anymore. Then, we search for violated infeasible path inequalities (4.28) and (4.33), respectively. All found violated inequalities are stored in a simple archive based on hashes [31] making it possible to detect and discard duplicates.

Directed Connection Cuts

The separation problem for inequalities (4.27) basically reduces to finding the minimum cut in a flow network. Due to the equivalence of maximum flow and minimum cut [5] we determine the minimum cut by computing a maximum flow. First, we construct a flow network $G_x = (V, A)$ with arc capacities ς_{uv} depending on the current LP solution $\boldsymbol{x}_{\mathrm{LP}}$, i.e. $\varsigma_{uv} = x_{uv},\ \forall (u, v) \in A$. Then, we obtain the maximum flow from root s to a terminal node $r \in R$ by applying a push-relabel method by Cherkassky and Goldberg [25] based on FIFO queues which runs in $\mathcal{O}(|V|^3)$ time. If the capacity of the according minimum cut with cut-set $W,\ s \in W,\ r \notin W$, is less than one, i.e. $\sum_{(u,v)\in A,\ u\in W,\ v\notin W} \varsigma_{uv} < 1$, then we have found a violated inequality (4.27) with set W. This directly follows from the definition of the arc capacities above. If we repeat this for all terminal nodes $r \in R$, we obtain an exact separation algorithm for directed connection cuts (4.27) running in polynomial time.

Usually and also here this standard method is improved by several extensions, cf. [5, 26, 100, 119]:

- *Source- and back-cuts*: In general there can be more than one minimum cut separating root s and a terminal $r \in R$. However, there is exactly one minimum cut "nearest" to the source, i.e. minimizing the cardinality of cut-set W, and there is exactly one minimum cut "nearest" to terminal r, i.e. minimizing the cardinality of cut-set $\overline{W}$. In case of multiple violating minimum cuts both the so-called *source-cut* and *back-cut* are added to the model.

- *Nested cuts*: In many cases it is beneficial to add as many violated inequalities as possible within one separation iteration. Thus, by setting the arc capacities of a found violating minimum cut to one and repeating the maximum flow computation to the same terminal node, we possibly obtain another minimum cut with capacity less than one.

- *Minimum cardinality cuts*: Due to the structure of inequalities (4.27) it can be beneficial to provide cuts consisting of a small number of arc variables. Therefore, by adding the same value $\epsilon > 0$ to all arc capacities we finally obtain the minimum cut with the least number of cut arcs.

- *Random terminal sequence*: If using nested cuts it may be disadvantageous if the terminal nodes are always examined in the same order. Due to the capacity modification the detection of violated inequalities with respect to later considered terminals may be prevented leading to a possibly unbalanced cut generation. Thus, at the beginning of a separation iteration we derive a random permutation of the set of terminal nodes R defining the examination order. (To preserve determinism of our branch-and-cut algorithm we initialize the seed value of the random generator always with the same value 0.)

Infeasible Path Cuts

Ascheuer et al. [7] show that the problem of deciding whether a given path is infeasible is $\mathcal{NP}$-complete for the TSP with time windows. However, the situation is different for the RDCST problem:

Theorem 4.7.1. *Let $P(v_1, v_k) \subseteq A$ be a simple path from node v_1 to v_k in G'. Whether $P \in \mathcal{P}_{\inf}$ or not can be decided in polynomial time.*

Proof. If $P(v_1, v_k)$ is a feasible path, i.e. is part of some feasible solution T, then all incoming arcs for nodes $v_i,\ i = 2, ..., k$, are fixed to the corresponding arcs in P. Since each node of a feasible tree can only have at most one incoming arc all other arcs to nodes $v_i,\ i = 2, ..., k$, cannot be in T. Thus, we eliminate all these arcs and obtain a graph $G'' = (V, A'')$ with $A'' = A \setminus \{(u, v_i) : u \neq v_{i-1},\ i = 2, ..., k\}$. Now we apply the simple instance feasibility test by computing the shortest-delay-paths to all terminal nodes which can be done in polynomial time by Dijkstra's algorithm [41]. If $d_v^{\min} \leq B,\ \forall v \in R$, in G'' then we obtain a feasible solution and thus P is feasible. Otherwise, if not even the shortest-delay-paths provide a feasible solution, $P \in \mathcal{P}_{\inf}$. □

Although the proof of Theorem 4.7.1 provides an exact method, for efficiency reasons we use a heuristic decision method similar to the one proposed in [8, 37]. For a path $P(v_1, v_k)$ we check if condition

$$d_{v_1}^{\min} + \sum_{i=1}^{k-1} d_{v_i v_{i+1}} \leq d_{v_k}^{\max} \tag{4.35}$$

holds. For definitions of $d_v^{\min}$ and $d_v^{\max}$ see Section 4.3. In case of violation of (4.35), we clearly found a sufficient condition for the infeasibility of path P. Otherwise, we assume path P to be feasible. Thus, our separation method is only able to find a subset $\mathcal{P}'_{\inf} \subseteq \mathcal{P}_{\inf}$ of all infeasible paths. However, if the current LP solution is integer, then our separation method is exact, i.e. $\mathcal{P}'_{\inf} = \mathcal{P}_{\inf}$, cf. [8, 37].

We build a support graph based on the current LP solution and enumerate all paths by backtracking from each node. Ascheuer et al. [8] argue – without a written proof – that there can only be a polynomial number of paths violating the tournament constraints in LP solutions for the TSP with time windows due to the node-degree limitations. Thus, by using some simple stopping criteria for backtracking they obtain a polynomial separation algorithm (with heuristic path infeasibility test). In case of the RDCST problem the question is still open if there can only be a polynomial number of paths violating inequalities (4.28). However, in our experimental results we always observed rather small numbers of backtracking steps within one separation. Further details about this enumeration procedure can also be found in [37]. When searching for violated lifted inequalities (4.33) the according strengthening arcs are considered in the enumeration algorithm to guarantee that all violations are found (w.r.t. set $\mathcal{P}'_{\inf}$).

4.8 Transformation to Layered Graph

Similarly to [70] we transform digraph $G' = (V, A)$ to a layered digraph $G_{\mathrm{L}} = (V_{\mathrm{L}}, A_{\mathrm{L}})$ with node set $V_{\mathrm{L}} = \{s\} \cup \{v_l \mid v \in V \setminus \{s\},\ 1 \leq l \leq B\}$. Thus, we introduce copies of all nodes except the root for each possible delay value. Arc set $A_{\mathrm{L}} = A_{\mathrm{L}}^{\mathrm{s}} \cup A_{\mathrm{L}}^{\mathrm{g}}$ consists of

- root arcs $A_{\mathrm{L}}^{\mathrm{s}} = \{(s, v_{d_{sv}}) \mid (s, v) \in A\}$ and
- general arcs $A_{\mathrm{L}}^{\mathrm{g}} = \{(u_l, v_{l+d_{uv}}) \mid (u, v) \in A,\ u, v \neq s,\ 1 \leq l \leq B - d_{uv}\}$.

Arc delays d_{uv} are not needed in G_L since they are implicitly contained in the layered structure: node v_l in G_L represents node v in G' with delay $d_v^T = l$ on path $P_T(s, v)$ in some solution T. Arc costs in A_L are the same as the costs of corresponding arcs in A.

With respect to the RDCST problem, we want to find an arborescence $T_\mathrm{L} = (V_\mathrm{L}^T, A_\mathrm{L}^T)$ in G_L with $V_\mathrm{L}^T \subseteq V_\mathrm{L},\ A_\mathrm{L}^T \subseteq A_\mathrm{L}$, rooted in $s \in V_\mathrm{L}^T$, including exactly one node $v_l \in V_\mathrm{L}^T$ for each terminal node $v \in R$ and at most one node $u_l \in V_\mathrm{L}^T$ for each potential Steiner node $u \in S$, having minimal costs $c_{T_\mathrm{L}} = \sum_{(u_k,v_l)\in A_\mathrm{L}^T} c_{uv}$. An optimal arborescence T_L^* in G_L as defined above corresponds to an optimal Steiner arborescence T^* for the RDCST problem on G', moreover $c_{T_\mathrm{L}^*} = c_{T^*}$. A solution T in G is obtained from an arborescence T_L by simply mapping all nodes $v_l \in V_\mathrm{L}^T \setminus \{s\}$ to v and arcs $(u_k, v_l) \in A_\mathrm{L}^T$ to (u, v), respectively.

Due to its possibly huge size preprocessing in G_L is even more important than in G'. Let $\deg^-(u_k)$ and $\deg^+(u_k)$ denote the indegree and outdegree of node u_k, respectively. The following reduction steps are repeated as long as G_L is modified by one of them:

1. To partly prevent cycles of length two in G' an arc $(u_k, v_l) \in A_\mathrm{L}$ can be removed if $\deg^-(u_k) = 1 \wedge (v_m, u_k) \in A_\mathrm{L}$ or $v \in S \wedge \deg^+(v_l) = 1 \wedge (v_l, u_m) \in A_\mathrm{L}$.

2. If node $v_l \in V_\mathrm{L} \setminus \{s\}$ has no incoming arcs it cannot be reached from s and therefore is removed.

3. If node $v_l \in V_\mathrm{L} \setminus \{s\},\ v \in S$, has no outgoing arcs it is removed since a Steiner node cannot be a leaf in an optimal solution.

These preprocessing rules are able to reduce the number of nodes and arcs usually dramatically, especially for instances with a broad range of delay values. Further reduction methods for Steiner trees can be found in [100, 119]. See Fig. 4.2 for an example of layered graph transformation, preprocessing, and solution correspondence.

Furthermore, let $G'_\mathrm{L} = (V'_\mathrm{L}, A'_\mathrm{L})$ be an extended layered graph with node set

$$V'_\mathrm{L} = V_\mathrm{L} \cup R_\mathrm{L},\ R_\mathrm{L} = \{\hat{v} \mid v \in R\} \tag{4.36}$$

and arc set

$$A'_\mathrm{L} = A_\mathrm{L} \cup \hat{A},\ \hat{A} = \{(v_l, \hat{v}) \mid v_l \in V_\mathrm{L},\ \hat{v} \in R_\mathrm{L},\ v \in R\}. \tag{4.37}$$

In this extension we add terminal node set R_L and according arcs to graph G_L. Solving the RDCST problem on graph G corresponds to solving the classical Steiner arborescence problem on layered graph G'_L. In principle, we can now apply any existing approach to solve the Steiner arborescence problem. Note that due to the definition of the layered graph it has the property of acyclicity which may be utilized in a solution method, e.g. when modeling connectivity: All general MIP tree models either need additional variables or an exponential number of constraints. Acyclicity makes it possible to model the problem effectively with a polynomial number of constraints without additional variables, see [67, 133] and Section 4.9. Approximation algorithms for the Steiner arborescence problem in acyclic digraphs are presented e.g. by Zelikovsky et al. [194] and Hsu et al. [86].

Picard and Queyranne [140] considered as one of the first authors a layered graph approach for the time-dependent traveling salesman problem in 1978. Here, the cost of using an edge

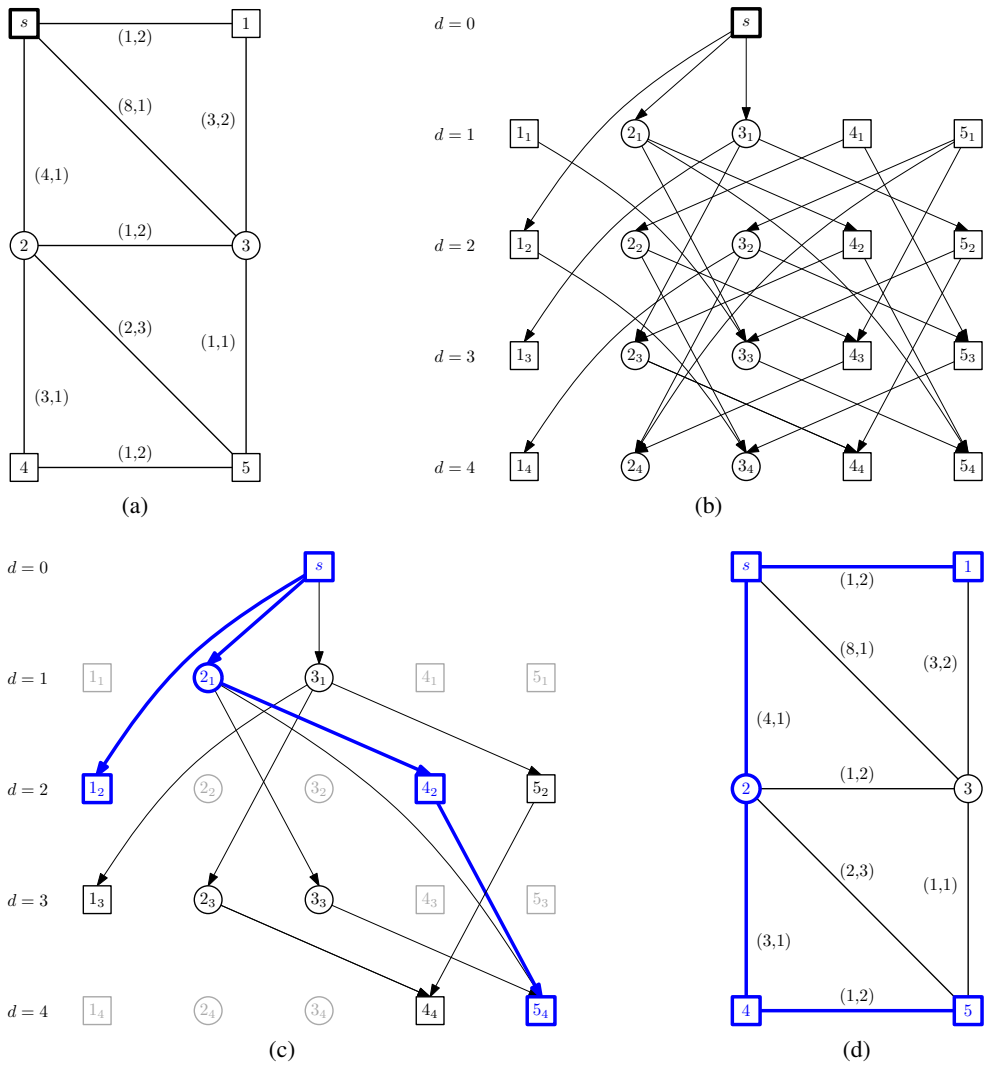


Figure 4.2: (a) Example graph with edge labels (c_e, d_e). Squared nodes denote terminal nodes. (b) Corresponding layered digraph for $B = 4$ (arc costs are omitted). (c) Preprocessed graph G_{L} with optimal solution denoted by blue arcs. (d) Optimal tree T^* in G with $c_{T^*} = 9$.

depends on the time when it is crossed. Especially this kind of problem aspects can be modeled quite easily on layered graphs since we simply assign different costs to the same original arc on different layers. Recently, layered graph approaches again increased in popularity, see [60] for strong formulations for the capacitated vehicle routing problem with unit demands, and [117, 118] for models for the hop-constrained facility location problem.

4.9 Layered Graph Formulation

Combining parts of the hop-indexed model in [67] and the layered graph model in [70] for the HCMST problem, we propose a MIP formulation on layered graph G_{L} for the RDCST problem. Again, we use binary variables $x_{uv},\ \forall (u,v) \in A$, to model original arcs in G'. Additionally, continuous variables $y_v^l,\ \forall v_l \in V_{\mathrm{L}} \setminus \{s\}$, and $x_{uv}^l,\ \forall (u_l, v_k) \in A_{\mathrm{L}}$, represent nodes and arcs in layered graph G_{L}, respectively. Model LAY is then defined as follows:

$$\min \quad \sum_{(u,v)\in A} c_{uv} x_{uv} \tag{4.38}$$

$$\text{s.t.} \quad \sum_{v_l \in V_L} y_v^l = 1 \qquad \forall v \in R \tag{4.39}$$

$$\sum_{v_l \in V_L} y_v^l \leq 1 \qquad \forall v \in S \tag{4.40}$$

$$\sum_{(u_k,v_l)\in A_{\mathrm{L}}} x_{uv}^k = y_v^l \qquad \forall v_l \in V_{\mathrm{L}} \setminus \{s\} \tag{4.41}$$

$$\sum_{(u_k,v_l)\in A_{\mathrm{L}}, u\neq w} x_{uv}^k \geq x_{vw}^l \qquad \forall (v_l, w_j) \in A_{\mathrm{L}}^{\mathrm{g}} \tag{4.42}$$

$$x_{sv}^0 = x_{sv} \qquad \forall (s,v) \in A \tag{4.43}$$

$$\sum_{(u_k,v_l)\in A_{\mathrm{L}}} x_{uv}^k = x_{uv} \qquad \forall (u,v) \in A,\ u \neq s \tag{4.44}$$

$$x_{uv}^k \geq 0 \qquad \forall (u_k, v_l) \in A_{\mathrm{L}} \tag{4.45}$$

$$y_v^l \geq 0 \qquad \forall v_l \in V_{\mathrm{L}} \setminus \{s\} \tag{4.46}$$

$$x_{uv} \in \{0,1\} \qquad \forall (u,v) \in A \tag{4.47}$$

Inequalities (4.39) and (4.40) state that from the set of layered graph nodes corresponding to one particular original node exactly one has to be chosen for required nodes and at most one for potential Steiner nodes, respectively. Indegree constraints (4.41) in G_{L} restrict the number of incoming arcs to a layered graph node v_l in dependency of y_v^l to at most one. Since G_{L} is acyclic inequalities (4.42) are enough to ensure connectivity. Equalities (4.43) and (4.44) link layered graph arcs to original arcs. LAY_{LP} denotes the LP relaxation of LAY.

In principle, variables x_{uv} and y_v^l are redundant since they can be substituted by Boolean layered graph arc variables x_{uv}^l using equalities (4.41), (4.43) and (4.44). However, model LAY is better readable by including them and branching on x_{uv} and Boolean y_v^l variables turned out to be more efficient in practice than branching on variables x_{uv}^l. In fact, branching on original arcs

usually is more balanced since setting $x^l_{uv} = 1$ for one particular layered graph arc in general is a stronger constraint on the set of feasible solutions than setting $x_{uv} = 1$.

Theorem 4.9.1. *Model LAY can be used to solve the RDCST problem.*

Proof. Constraints (4.39) and (4.41) force exactly one incoming arc to exactly one layered graph node v_l in G_L for each terminal node $v \in R$. We have to show that all included nodes $v_l \in V_\mathrm{L}$ are connected to the root node s. If the incoming arc (u_k, v_l) originates in s, i.e. $u_k = s$, we are done. Otherwise inequalities (4.42) ensure an incoming arc to u_k. Due to the acyclicity and the layered structure of G_L the source of an arc can only be in a lower layer than the target. Repeating this argumentation for node u_k extends the path in a backtracking way to the root node in layer 0. The union of all such paths forms a connected acyclic subgraph including one layered graph node for each terminal node. Since inequalities (4.41) restrict the indegree to a layered graph node to at most one, the resulting subgraph is an arborescence T_L rooted in s. Finally, linking inequalities (4.44) and (4.43) transform T_L back to a feasible arborescence T' in graph G'. □

The number of variables and constraints of model LAY can be estimated by equations (4.48) and (4.49) showing the high dependency on delay-bound B. Therefore B is crucial for the performance and memory consumption of this model which can be clearly observed in the experimental results, see Section 4.11.

$$\#\text{variables} = |A| + |V_\mathrm{L}| + |A_\mathrm{L}| = \mathcal{O}((|V| + |A|) \cdot B) \tag{4.48}$$

$$\#\text{constraints} = |R| + |S| + 2(|V_\mathrm{L}| - 1) + |A^\mathrm{g}_\mathrm{L}| + 2|A| + |A_\mathrm{L}| = \mathcal{O}((|V| + |A|) \cdot B) \tag{4.49}$$

4.9.1 Valid Inequalities

The following sets of valid inequalities are not necessary for the feasibility of model LAY but are useful to strengthen it w.r.t. its LP relaxation. Inequalities (4.30), (4.31), and (4.27) used for model PC in Section 4.7 are able to strengthen model LAY, too, see Fig. 4.3a for an example.

A stronger variant of (4.27) can be defined on the extended layered graph G'_L, see [70]:

$$\sum_{(u_k,v_l)\in A'_\mathrm{L},\ u_k\in W_\mathrm{L},\ v_l\in\overline{W_\mathrm{L}}} x^k_{uv} \geq 1 \qquad \forall W_\mathrm{L} \subset V'_\mathrm{L},\ s \in W_\mathrm{L},\ \overline{W_\mathrm{L}} \cap R_\mathrm{L} \neq \emptyset \tag{4.50}$$

It can be easily seen that inequalities (4.50) include (4.27). Figure 4.3b shows an example for strengthening model LAY.

4.9.2 Separation Methods

Inequalities (4.30) and (4.31) are included in the model a priori while (4.27) and (4.50) need to be separated dynamically during branch-and-cut. Violated inequalities are found in the same way as for model PC, see Section 4.7.2 for details. Clearly, the separation of inequalities (4.50) takes place on layered graph G'_L, and capacities for arcs $\hat{A}$ are set to 1 in the maximum flow calculations.

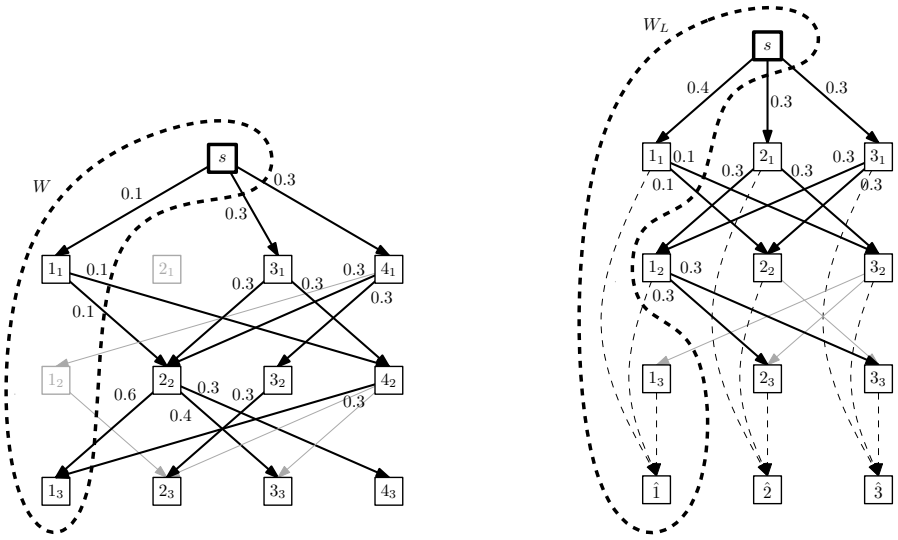


Figure 4.3: Both examples are feasible for LAY_{LP} (arc labels denote variable values of the LP solution, gray arcs mean $x^l_{uv} = 0$). (a) The solution violates inequality (4.27) with $W = \{s, 1\}$ and the root-constraint (4.30). (b) The solution violates inequality (4.50) with $W_L = \{s, 1_1, 1_3, \hat{1}\}$.

We observed in our experiments that usually a large number of violated inequalities (4.50) on the layered graph is found resulting in a significant increase of the according dual bound. In many cases the LP relaxation including all violated inequalities even is integral, without need for branching anymore. However, often many separation iterations are necessary to reach this best dual bound. Additionally, the LP resolvings after adding cuts usually take much time because of the huge size of the corresponding layered graph formulation. One of the reasons for the high number of resolvings may be the capacity modifications within the separation problem when searching for nested cuts: Setting the capacity of all detected cut arcs to 1 possibly prevents finding violating cuts to other terminal nodes, which then have to be found in later iterations. Therefore, by disabling nested cuts and capacity modifications we are hopefully able to reduce the number of separation rounds and thus the number of LP resolvings. Clearly, we have to accept the consequence that probably redundant violated inequalities are added to the model. However, preliminary results indicate that in most cases this version is advantageous compared to the variant including nested cuts.

4.10 Polyhedral Comparison

The aim of this section is to theoretically compare the polyhedra associated to the LP relaxation of the following formulations for the RDCST problem and projected into the space of $\boldsymbol{x}$ variables:

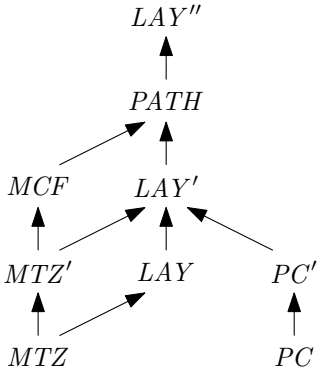


Figure 4.4: Polyhedral comparison of all described formulations. An arrow describes a direct relation between two formulations: the formulation at the tail of an arrow is weaker than the formulation at the head. If there is no directed path between two formulations, the corresponding polyhedra are incomparable.

- MTZ: compact extended formulation based on Miller-Tucker-Zemlin inequalities in Section 4.4
- MTZ': formulation MTZ with directed connection cut inequalities (4.27) on graph G'
- $PATH$: path formulation in Section 4.5
- MCF: multi-commodity flow formulation in Section 4.6
- PC: path-cut formulation in Section 4.7
- PC': formulation PC with inequalities (4.28) replaced by lifted variants (4.33)
- LAY: layered graph formulation in Section 4.9 with inequalities (4.30)–(4.31)
- LAY': formulation LAY with directed connection cut inequalities (4.27) on graph G'
- LAY'': formulation LAY with directed connection cut inequalities (4.50) on layered graph G'_{L}

We denote the polyhedra of the according LP relaxations projected into the space of $\boldsymbol{x}$ variables by P_{MTZ}, $P_{MTZ'}$, P_{BP}, P_{MCF}, P_{PC}, $P_{PC'}$, P_{LAY}, $P_{LAY'}$, and $P_{LAY''}$, respectively. We write $P_1 \subset P_2$ if $P_1 \subseteq P_2$ and there exist instances such that $\exists \boldsymbol{x} \in P_2 : \boldsymbol{x} \notin P_1$. Figure 4.4 summarizes the relations between the formulations which are now discussed one by one.

Proposition 4.10.1. $P_{LAY} \subset P_{MTZ}$.

Idea of proof. In principle, we can relate a node $v_l \in V_{\mathrm{L}}$ in layered graph G_{L} to node $v \in V$ with node-delay $\delta_v = l$ in formulation MTZ. However, the notion of node-delay is described more accurately in LAY since it equals the sum of the actual arc delays on the path to a node, whereas in MTZ it also depends on the arc variables on that path. Additionally, MTZ inequalities 4.9

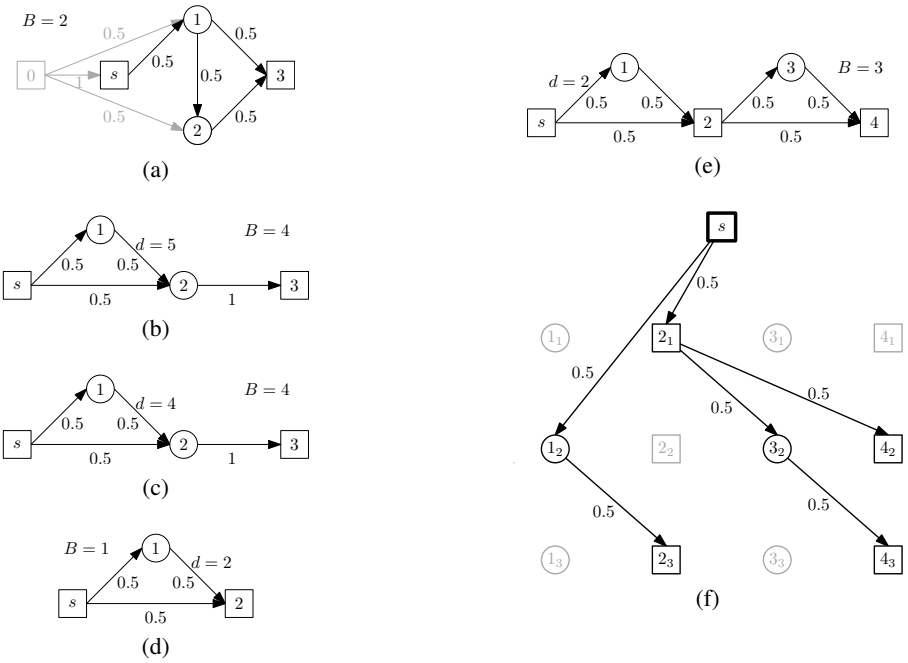


Figure 4.5: Examples for polyhedral comparison. Arc labels in all figures denote variable values of the LP solution, arc delays are 1 if not explicitly defined, and squared nodes denote terminal nodes.

relate δ_v values in a way that only upper bounds on the actual node-delays in a solution are obtained, even in the integer case. Therefore, an arc can be included in a solution of MTZ_{LP} which is infeasible according to the shortest-delay-path from root s in this solution, as shown in Fig. 4.5a for arc $(2, 3)$, which is not feasible for LAY_{LP}. □

Proposition 4.10.2. $P_{MTZ'} \subset P_{MTZ}$.

Proof. Clearly, $P_{MTZ'} \subseteq P_{MTZ}$ holds. Figure 4.5a shows a solution which is feasible for MTZ_{LP} but violates directed connection cut inequalities (4.27) with $W = \{s\}$. □

Proposition 4.10.3. $P_{LAY'} \subset P_{LAY}$.

Proof. Clearly, $P_{LAY'} \subseteq P_{LAY}$ holds. Figure 4.3a shows an example where directed connection cut inequalities (4.27) are able to strengthen LAY_{LP}: Basically, "oscillations" between layered graph nodes, e.g. arcs $(3_1, 2_2)$ and $(2_2, 3_3)$, which represent cycles of length two in graph G', prevent in some cases a feasible flow of one unit from root s to a terminal node in G'. Thus, there has to be minimum cut of capacity less than one resulting in a violated connection cut inequality (4.27). □

Proposition 4.10.4. $P_{LAY'} \subset P_{MTZ'}$.

Idea of proof. Both formulations include directed connection cut inequalities (4.27). Apart from that, we can apply the same argumentation as in Proposition 4.10.1, here. □

Proposition 4.10.5. $P_{PC'} \subset P_{PC}$.

Proof. Clearly, $P_{PC'} \subseteq P_{PC}$ holds. Figure 4.1b shows an example where lifted infeasible path inequalities (4.33) are able to strengthen PC_{LP}. In principle, the fact that inequalities (4.33) are able to combine different infeasible paths helps to tighten the formulation. □

Proposition 4.10.6. $P_{LAY'} \subset P_{PC'}$.

Idea of proof. Directed connection cut inequalities (4.27) are included in both formulations, and it is not hard to see that a similar argumentation as in Proposition 4.10.1 can be used here, too: Solution 4.5d is feasible for PC'_{LP} but not for LAY'_{LP} since due to $B = 1$ the corresponding layered graph only has one layer additional to root node s, and thus arc $(1, 2)$ has no counterpart in G_{L}. □

Proposition 4.10.7. $P_{MCF} \subset P_{MTZ'}$.

Idea of proof. By applying the max-flow-min-cut theorem [5] we know that subtour elimination by multi-commodity flows is equivalent to using directed connection cut inequalities. Furthermore, in [137] MTZ inequalities have been shown to be weaker than multi-commodity flow inequalities to prevent cycles. Regarding the delay-constraints, MTZ inequalities (4.9) provide a weak description of a delay-feasible path $P(s, v),\ v \in R$, in a sense that they cannot combine different fractional paths to terminal v whereas inequalities (4.23) in model MCF sum up the whole flow assigned to this terminal node possibly using different fractional paths. Example 4.5b shows such a situation: the solution is feasible for MTZ'_{LP} with $\delta_1 = 0.5,\ \delta_2 = 3,\ \delta_3 = 4$, but not for MCF_{LP} with $f^3_{uv} = x_{uv},\ \forall (u, v) \in A$, and $\sum_{(u,v)} d_{uv} f^3_{uv} = 4.5 > B$. □

Proposition 4.10.8. $P_{PATH} \subset P_{MCF}$.

Proof. Dahl et al. [32] argue for the HCMST problem that formulation $PATH$ describes the convex hull of the incidence vectors of delay-feasible paths which is in general not the case for MCF. Thus, MCF possibly includes delay-infeasible paths in a fractional solution. Obviously, this result can also be applied to the more general RDCST problem. An example is shown in Fig. 4.5c: The solution is feasible for MCF but infeasible for $PATH_{\mathrm{LP}}$ since path $P = \{(s, 1), (1, 2), (2, 3)\}$ is delay-infeasible and thus not part of the formulation. Therefore, path $P = \{(s, 2), (2, 3)\}$ has to be set to 1 forcing arc $(s, 2)$ to 1, too. □

Proposition 4.10.9. $P_{PATH} \subset P_{LAY'}$.

Idea of proof. It is well-known that formulation $PATH$ includes directed connection cut inequalities (4.27) on graph G'. Further, we already know that $PATH$ describes the convex hull of the incidence vectors of delay-feasible paths [32]. Additionally, connection cut inequalities (4.27) in model LAY' work on graph G' and thus do not respect delays. Therefore, a minimum cut in G' for some set S may have capacity greater or equal than 1 but using these cut arcs in a solution may violate the delay-bound. Connection cut inequalities (4.50) on layered graph G'_{L} prevent these situations, significantly strengthening LAY'_{LP}, see Proposition 4.10.10. In other words, LAY' is not able to combine different fractional delay-feasible paths to one

terminal node in a way that the multiple use of common arcs is reflected in the according arc variables. In contrast, $PATH$ sums up multiple arc crossings to the same terminal node.

The solution 4.5e on graph G' and the equivalent arborescence on layered graph G_{L} are feasible for LAY'_{LP}, but not for $PATH_{\mathrm{LP}}$ since path $P = \{(s,1),(1,2),(2,3),(3,4)\}$ is infeasible. Thus, to fulfill inequalities (4.14) for terminal node 4, we e.g. combine paths $P_1 = \{(s,2),(2,3),(3,4)\}$ and $P_2 = \{(s,2),(2,4)\}$, but then arc $(s,2)$ is forced to 1. □

Proposition 4.10.10. $P_{LAY''} \subset P_{PATH}$.

Proof. This result is shown in [70] for the HCMST problem and can easily be adapted to the RDCST problem. Basically, Gouveia et al. argue that in a fractional solution of $PATH_{\mathrm{LP}}$ the same arc can appear in different delay positions in different paths, which is not possible for formulation LAY''. □

Proposition 4.10.11. $P_1 \neq P_2,\ \forall P_1 \in \{P_{MTZ},\ P_{MTZ'},\ P_{MCF}\},\ \forall P_2 \in \{P_{PC},\ P_{PC'}\}$.

Proof. Solution 4.5c is feasible for MCF_{LP} with $f^3_{uv} = x_{uv}$ for all arcs $(u,v) \in A$, and $\sum_{(u,v)} d_{uv} f^3_{uv} = 4 \leq B$, but not for PC_{LP} with infeasible path $P = \{(1,2),(2,3)\}$ since $x_{12} + x_{23} = 1.5 > |P| - 1 = 1$. Furthermore, solution 4.5d is feasible for PC'_{LP} but not for MTZ_{LP} with $\delta_1 = 0.5,\ \delta_2 = 1.5 > B$. These two examples together with previously shown results imply all incomparableness relations. □

Proposition 4.10.12. $P_{MCF} \neq P_2,\ P_2 \in \{P_{LAY},\ P_{LAY'}\}$.

Proof. Solution 4.5c is feasible for MCF_{LP} with $f^3_{uv} = x_{uv}$ for all arcs $(u,v) \in A$, and $\sum_{(u,v)} d_{uv} f^3_{uv} = 4 \leq B$, but not for LAY_{LP} since there is no arc in layered graph G_{L} corresponding to arc $(1,2)$. Furthermore, the solution in Fig. 4.5e and 4.5f is feasible for LAY'_{LP} but not for MCF_{LP} with $f^4_{uv} = x_{uv},\ \forall (u,v) \in A$, and $\sum_{(u,v)} d_{uv} f^4_{uv} = 3.5 > B$. □

Proposition 4.10.13. $P_{LAY} \neq P_2,\ P_2 \in \{P_{MTZ'},\ P_{PC},\ P_{PC'}\}$.

Proof. Formulation LAY does not include all directed connection cut inequalities (4.27), see Fig. 4.3a for an example, whereas formulations MTZ', PC, and PC' include them. Furthermore, solution 4.5d is feasible for PC'_{LP} but not for LAY_{LP} because of arc $(1,2)$ which has no counterpart in layered graph G_{L}. Finally, solutions 4.5b and 4.5c are feasible for MTZ'_{LP} but not for LAY_{LP} again due to arc $(1,2)$. □

4.11 Computational Results

In this section we compare all mentioned formulations on different sets of benchmark instances. In detail we discuss the following solution approaches:

- M1: branch-and-cut (BC) based on compact extended formulation MTZ but with lifted MTZ inequalities by Leggieri et al. [111]
- M2: M1 extended by directed connection cut inequalities (4.27)

- BP: branch-and-price approach by Leitner et al. [113] based on path formulation $PATH$ from Section 4.5 stabilizing column generation by using alternative dual-optimal solutions (algorithm-specific parameter Q is set to 20)
- FL: BC based on multi-commodity flow formulation MCF in Section 4.6
- P1: BC based on path-cut formulation PC in Section 4.7 with inequalities (4.30)–(4.32) added a priori to the model
- P2: P1 with inequalities (4.28) replaced by lifted variants (4.33), and (4.34) added a priori to the model PC
- L1: BC based on layered graph formulation LAY in Section 4.9 with inequalities (4.30)–(4.31) added a priori to the model
- L2: L1 extended by directed connection cut inequalities (4.27) on graph G'
- L3: L1 extended by directed connection cut inequalities (4.50) on layered graph G'_{L}

4.11.1 Test Instances and Environment

We apply all MIP approaches to benchmark instances originally proposed by Gouveia et al. [68] for the spanning tree variant of the RDCST problem consisting of complete graphs with 41 nodes. The three main instance sets R, C, and E each have different graph structures defined by their edge cost functions: R has random edge costs, C and E both have Euclidian costs fixing root node s near the center and near the border, respectively. Each main instance set consists of different subsets of five input graphs varying in the number of possible discrete edge delay values, e.g. C100 denotes the set of instances with 100 different integer delay values $d_e \in \{1, ..., 100\},\ \forall e \in E$.

Additional instance sets Tα are based on the self-generated R100 instances for the RDCMST problem, cf. Section 3.13.1, where the set of terminal nodes is $R = \{1, ..., \alpha\}$.

The third set of benchmark instances is introduced by Leggieri et al. [111] and based on instances from the *SteinLib Library*[1] [101]. All instances of SteinLib set B, the first ten instances of set C, and the first five instances of set D are used which contain sparse graphs with $|V| \in [50, 1000]$ and $|E| \in [63, 1250]$. Each edge is assigned a random integer cost value uniformly distributed in $\{1, ..., 10\}$. Additionally, ten Euclidian complete graphs based on SteinLib instances Berlin52 ($|V| = 52,\ |R| = 16$) and Brazil58 ($|V| = 58,\ |R| = 25$) are used. Since SteinLib instances do not contain edge delays, they have been generated by Leggieri et al. in two different ways: randomly within $\{1, ..., 100\}$ (subset Ran), and correlated to the cost value by choosing a random number $r \in [0.8, 1.2]$ for each edge and setting $d_e = r \cdot c_e$ (subset Cor). Tested delay-bounds depend on a particular input graph by computing the maximum among the shortest-delay-paths to all terminal nodes, i.e. $\delta^{\max} = \max_{v \in R} d_v^{\min}$, and using $B \in \{1.1 \cdot \delta^{\max}, 1.5 \cdot \delta^{\max}\}$.

[1] `http://steinlib.zib.de`

To reduce the input graphs we applied all preprocessing methods described in Section 3.3 and 4.3 prior to solving. To provide an initial primal solution for the MIP tests in Section 4.11.3 we applied the Kruskal-based heuristic (KBH) from Section 3.4 followed by the VND from Section 3.8 to all spanning tree instances, whereas a simple heuristic denoted by CSP is used if $R \subset V \setminus \{s\}$, which iteratively adds delay-constrained shortest paths from the root node to terminal nodes while dissolving possible cycles [111]. However, since the KBH and VND work on undirected graphs and all MIP approaches are based on the corresponding directed counterpart, situations can arise where preprocessing on the directed graph is able to remove additional arcs which are included in the feasible solution obtained by the heuristics. In such cases the primal bound is still feasible but the tree cannot be handed over to the MIP solver as guiding solution. Especially for branch-and-price approach BP this is problematic since here we need an initial set of feasible paths to all terminal nodes. Thus, for BP we use the CSP heuristic which can cope with directed graphs for both Steiner and spanning tree instances.

We used IBM ILOG CPLEX 12.3 to solve the MIP models. The layered node and arc variables in model LAY and the flow variables in MCF are declared Boolean since the CPLEX presolver benefits from integrality of these variables and therefore can significantly reduce the model. Furthermore, because of too high time consumption we disabled the probing extension of CPLEX which checks the logical implications of setting each Boolean variable to 0 or 1. Branch-and-price approach BP is solved by ZIB SCIP 2.1.0 [2] with CPLEX 12.3 as embedded LP solver since CPLEX is not able to add columns within branch-and-bound nodes other than the root node. However, to provide a fair comparison between the approaches we have to take into account that SCIP is in average about 2-3 times slower than CPLEX when considering general benchmark instances[2]. For BP, we further deactivated presolving and separation of general purpose cutting planes (as recommended) and set parameter "fastmip" to 1. The dual simplex algorithm has been used for solving LPs in CPLEX, since it turned out to significantly outperform other options (primal simplex, barrier) in preliminary tests. Finally, a memory limit of 4 GB and a time limit of 10 000 CPU-seconds are set for each experiment. Apart from that, all other CPLEX and SCIP settings remain at their default.

Similarly to RDCMST problem benchmarks, cf. Section 3.13.1, all tests are performed on a single core of Intel Xeon E5540 processors with 2.53 GHz where eight cores share 24 GB of memory.

4.11.2 LP Bounds

Tables 4.1–4.3 show results for several instance sets when only solving the LP relaxation of the considered formulations. We report obtained average gaps between the optimal LP relaxation value and the optimal integer value denoted as LP-gap, and the median runtime to reach these best possible LP bounds. Dashes in LP-gap columns are used if for at least one of the instances the optimal LP value could not be obtained within the given time limit, whereas dashes in time columns represent the time limit of 10 000 seconds.

In general, Euclidian instances seem to be harder to solve, e.g. see C and E sets compared to R instances in Table 4.1. This is mainly because of the lower success rate of our preprocessing

[2] `http://scip.zib.de`

		average LP-gap in %									median time in seconds								
Set	B	M1	M2	BP	FL	P1	P2	L1	L2	L3	M1	M2	BP	FL	P1	P2	L1	L2	L3
R5	6	31.9	31.7	**0.0**	16.2	40.0	1.9	0.6	0.6	**0.0**	**0**	**0**	2	22	**0**	**0**	**0**	**0**	**0**
	8	30.8	30.5	0.1	16.6	32.3	6.4	0.6	0.6	**0.0**	**0**	**0**	3	20	**0**	**0**	**0**	**0**	**0**
	10	24.4	24.0	**0.0**	12.3	23.2	8.5	1.2	1.2	**0.0**	**0**	**0**	4	18	**0**	**0**	**0**	**0**	**0**
	12	16.3	15.9	0.1	7.6	14.7	5.3	0.4	0.4	**0.0**	**0**	**0**	3	10	**0**	**0**	**0**	**0**	**0**
C5	6	15.9	13.8	0.2	7.2	19.0	1.7	1.0	0.7	**0.0**	**0**	**0**	2	8	**0**	**0**	**0**	**0**	**0**
	8	16.7	13.9	0.1	4.6	16.2	3.4	1.9	1.5	**0.0**	**0**	**0**	4	10	**0**	**0**	**0**	**0**	**0**
	10	16.0	13.1	0.5	5.2	13.6	5.8	4.4	3.1	**0.1**	**0**	**0**	4	8	**0**	**0**	**0**	**0**	1
	12	13.5	10.5	0.2	4.0	10.3	5.5	4.6	3.1	**0.0**	**0**	**0**	4	5	**0**	**0**	**0**	**0**	3
E5	6	23.0	21.1	0.4	9.3	26.2	5.0	3.4	3.1	**0.0**	**0**	**0**	7	79	**0**	**0**	**0**	**0**	**0**
	8	22.9	20.2	0.2	6.8	21.7	9.0	5.4	4.6	**0.0**	**0**	**0**	14	117	**0**	1	**0**	**0**	1
	10	20.2	16.9	0.3	4.5	16.8	9.9	6.5	5.6	**0.0**	**0**	**0**	25	122	**0**	**0**	**0**	1	5
	12	17.6	14.3	0.5	3.9	14.1	9.5	7.3	5.6	**0.0**	**0**	**0**	44	88	**0**	**0**	**0**	1	18
R10	10	32.2	31.9	0.1	16.8	36.1	4.4	1.1	1.1	**0.0**	**0**	**0**	3	25	**0**	**0**	**0**	**0**	**0**
	15	30.6	30.2	**0.0**	15.9	31.2	8.4	1.1	1.1	**0.0**	**0**	**0**	4	27	**0**	**0**	**0**	**0**	**0**
	20	21.0	20.6	**0.0**	9.8	19.8	7.0	0.5	0.5	**0.0**	**0**	**0**	4	16	**0**	**0**	**0**	**0**	**0**
	25	14.9	14.5	0.1	7.8	14.1	6.3	1.0	0.9	**0.0**	**0**	**0**	4	15	**0**	**0**	**0**	1	1
C10	10	16.1	14.3	**0.0**	7.9	16.9	2.6	2.1	1.6	**0.0**	**0**	**0**	3	14	**0**	**0**	**0**	**0**	**0**
	15	16.7	14.2	0.2	5.8	15.1	4.6	2.9	2.1	**0.0**	**0**	**0**	4	15	**0**	**0**	**0**	**0**	2
	20	13.7	10.8	0.2	4.3	10.7	5.3	4.1	2.7	**0.0**	**0**	**0**	5	14	**0**	**0**	**0**	1	12
	25	10.9	8.1	0.2	3.7	7.8	4.6	4.2	2.4	**0.0**	**0**	**0**	6	8	**0**	**0**	1	3	41
E10	10	22.7	20.9	0.1	8.9	25.0	4.2	2.7	2.3	**0.0**	**0**	**0**	6	75	**0**	**0**	**0**	**0**	**0**
	15	22.2	19.2	0.5	6.6	19.9	10.1	6.8	5.9	**0.0**	**0**	**0**	19	134	**0**	**0**	**0**	1	9
	20	18.7	15.4	0.7	4.7	15.1	10.4	7.8	6.2	**0.0**	**0**	**0**	64	131	**0**	**0**	**0**	3	70
	25	15.5	12.3	0.8	3.7	11.9	9.2	8.3	6.2	**0.1**	**0**	**0**	85	105	**0**	**0**	1	9	265
R100	100	32.4	32.0	0.2	18.2	35.6	7.1	3.3	3.3	**0.1**	**0**	**0**	4	28	**0**	**0**	2	2	4
	150	25.7	25.4	0.1	13.1	24.4	8.6	2.8	2.7	**0.0**	**0**	**0**	4	20	**0**	**0**	5	7	56
	200	16.5	16.2	**0.0**	8.2	15.1	6.0	0.6	0.5	**0.0**	**0**	**0**	5	13	**0**	**0**	14	20	24
	250	10.3	10.0	**0.0**	5.4	9.4	4.4	0.2	0.2	**0.0**	**0**	**0**	6	7	**0**	**0**	34	46	181
C100	100	19.3	16.9	0.1	8.6	18.6	5.7	4.1	3.3	**0.0**	**0**	**0**	8	51	**0**	**0**	3	9	192
	150	14.9	11.9	**0.0**	4.9	12.1	5.1	3.4	2.4	**0.0**	**0**	**0**	9	31	**0**	**0**	16	67	1293
	200	11.5	8.5	**0.1**	3.7	8.2	4.5	3.9	2.2	-	**0**	**0**	13	16	**0**	**0**	43	417	6202
	250	8.9	6.0	**0.1**	2.9	5.8	3.5	3.4	1.8	-	**0**	**0**	11	14	**0**	**0**	76	566	-
E100	100	22.2	19.6	0.3	8.0	21.9	7.9	6.5	5.2	**0.0**	**0**	**0**	17	163	**0**	**0**	4	13	713
	150	19.1	16.0	**0.3**	5.2	16.1	9.2	7.6	5.5	-	**0**	**0**	37	130	**0**	**0**	17	115	-
	200	15.8	12.6	**0.5**	4.1	12.2	8.3	8.0	5.5	-	**0**	**0**	79	122	**0**	**0**	57	423	-
	250	12.7	9.6	**0.3**	3.0	9.3	7.0	7.4	4.7	-	**0**	**0**	59	79	**0**	**0**	160	4117	-
R1000	1000	39.4	38.9	0.4	21.3	43.3	11.2	3.0	2.9	**0.0**	**0**	**0**	10	54	**0**	**0**	143	174	1596
	1500	29.4	29.0	**0.5**	15.5	29.3	13.5	4.2	4.1	-	**0**	**0**	16	32	**0**	**0**	825	1336	-
	2000	19.1	18.8	**0.0**	9.3	17.6	7.8	1.6	1.6	-	**0**	**0**	20	23	**0**	**0**	1634	2241	2328
	2500	11.2	10.8	**0.0**	5.6	10.2	4.2	1.1	1.1	-	**0**	**0**	20	14	**0**	**0**	3679	2994	3139
C1000	1000	20.0	17.4	**0.0**	9.1	21.1	4.5	2.2	1.9	-	**0**	**0**	15	57	**0**	**0**	152	252	6560
	1500	17.2	14.2	**0.2**	5.8	14.6	6.1	4.0	2.8	-	**0**	**0**	19	36	**0**	**0**	930	2818	-
	2000	14.6	11.7	**0.5**	5.4	11.4	7.0	5.4	-	-	**0**	**0**	33	27	**0**	**0**	2352	-	-
	2500	10.2	7.4	**0.1**	3.3	7.1	4.5	-	-	-	**0**	**0**	36	11	**0**	**0**	5299	-	-
E1000	1000	22.3	20.0	**0.0**	9.3	21.8	7.7	5.6	4.7	-	**0**	**0**	24	122	**0**	**0**	339	1091	-
	1500	19.7	16.5	**0.3**	5.6	16.7	9.4	6.6	-	-	**0**	**0**	50	122	**0**	**0**	1184	8089	-
	2000	16.2	13.0	**0.1**	3.6	12.6	8.5	7.4	-	-	**0**	**0**	77	95	**0**	**0**	3044	-	-
	2500	13.3	10.2	**0.3**	2.6	9.8	7.0	-	-	-	**0**	**0**	150	71	**0**	**0**	7256	-	-

Table 4.1: LP results for instances by Gouveia et al. [68] (B: delay-bound, LP-gap: gap between optimal LP relaxation and optimal integer value, M1: Miller-Tucker-Zemlin approach, M2: M1 with connection cuts, BP: stabilized branch-and-price, FL: flow approach, P1: path-cut approach, P2: P1 with lifted cuts, L1: layered graph approach, L2: L1 with connection cuts on G', L3: L1 with connection cuts on G'_{L}, best results are printed bold).

		average LP-gap in %									median time in seconds								
Set	B	M1	M2	BP	FL	P1	P2	L1	L2	L3	M1	M2	BP	FL	P1	P2	L1	L2	L3
T10	16	24.2	21.8	0.1	10.5	20.6	16.3	2.0	0.9	**0.0**	**0**	**0**	**0**	1	**0**	**0**	**0**	**0**	**0**
	30	37.9	35.5	**0.1**	16.4	35.4	31.2	2.9	1.4	**0.1**	**0**	**0**	2	10	**0**	**0**	**0**	1	1
	50	43.2	40.7	0.2	17.5	40.8	36.6	4.0	2.0	**0.1**	**0**	**0**	3	31	1	1	5	8	15
	100	42.8	40.7	**0.0**	18.1	41.3	36.9	4.5	2.1	**0.0**	**0**	**0**	5	59	1	1	69	124	187
T30	16	23.3	22.7	**0.1**	12.4	23.3	13.6	0.4	0.3	**0.1**	**0**	**0**	2	19	**0**	**0**	**0**	**0**	**0**
	30	37.0	36.1	0.1	17.7	37.2	27.1	1.7	1.4	**0.0**	**0**	**0**	8	255	1	1	1	1	2
	50	40.5	39.6	0.3	19.7	40.5	31.6	2.8	2.2	**0.1**	**0**	**0**	17	641	1	2	8	11	52
	100	40.6	39.9	**0.4**	19.2	41.2	33.0	4.0	3.5	-	**0**	**0**	41	1863	2	2	118	176	1033
T50	16	22.3	22.0	0.1	13.2	23.7	10.3	0.5	0.5	**0.0**	**0**	**0**	5	146	**0**	**0**	**0**	**0**	**0**
	30	36.4	35.9	0.2	19.2	38.0	22.3	1.6	1.5	**0.1**	**0**	**0**	18	971	1	1	1	1	3
	50	38.9	38.3	0.2	19.5	40.0	25.9	2.1	1.9	**0.1**	**0**	**0**	37	2480	2	2	8	10	53
	100	39.3	38.9	**0.4**	-	40.7	27.8	3.6	3.4	-	**0**	1	109	7429	2	3	77	124	1461
T70	16	21.0	20.8	**0.0**	13.4	23.3	6.0	0.3	0.3	**0.0**	**0**	**0**	8	454	1	**0**	**0**	**0**	**0**
	30	34.8	34.4	0.2	19.7	37.4	15.4	1.5	1.5	**0.1**	**0**	**0**	30	1824	1	2	1	1	4
	50	37.5	37.1	0.2	-	39.6	19.6	1.8	1.7	**0.1**	**0**	**0**	65	4758	2	3	7	8	53
	100	38.1	37.8	**0.4**	-	40.0	22.5	2.8	2.7	-	**0**	1	231	-	3	5	76	91	1277
T99	16	20.2	20.1	0.2	13.9	23.7	0.8	0.5	0.4	**0.1**	**0**	**0**	14	1463	**0**	**0**	**0**	**0**	**0**
	30	32.4	32.2	0.3	19.5	36.1	4.4	1.4	1.4	**0.1**	**0**	**0**	53	2917	1	1	1	1	4
	50	35.8	35.6	0.3	-	38.9	8.7	1.7	1.7	**0.1**	**0**	1	118	7598	1	3	6	6	50
	100	36.0	35.9	**0.4**	-	38.6	12.4	2.2	2.2	-	**0**	1	367	-	2	5	43	51	886

Table 4.2: LP results for random instances from Section 3.13.1 (B: delay-bound, LP-gap: gap between optimal LP relaxation and optimal integer value, M1: Miller-Tucker-Zemlin approach, M2: M1 with connection cuts, BP: stabilized branch-and-price, FL: flow approach, P1: path-cut approach, P2: P1 with lifted cuts, L1: layered graph approach, L2: L1 with connection cuts on G', L3: L1 with connection cuts on G'_{L}, best results are printed bold).

phase which includes special rules for random graphs like alternative paths and triangles, see Section 3.3.2 for details. Additionally, Gouveia et al. [68] already mentioned that the E instances with the root node placed near the border are much harder to solve than the similar C instances with root s near the center.

The theoretical discussion in Section 4.10 indicates the superiority of formulations $PATH$ and LAY'', which is also reflected in the experimental results. Especially the LP relaxation of LAY'' is in most cases integral – if it can be computed within the time limit. We can clearly see the crucial disadvantage of layered graph approaches here: When the number of achievable delay values and the delay-bound increases, the size of the layered graph and thus the corresponding model increases, too, resulting in high computation times, e.g. see instance sets R1000, C1000, and E1000 in Table 4.1 and Brazil-Cor in Table 4.3. How this drawback can be partly avoided is discussed in Chapter 6.

Compared to LAY'', the second strongest formulation $PATH$ seems to be more robust and rather independent of the delay-bound, although the runtime of the pricing subproblem indeed depends on B. However, the main reason for the low runtimes of BP is the method introduced by Markus Leitner [113] stabilizing and accelerating the column generation process. The unstabilized variant is presented in [68] showing rather poor performance mainly due to degeneracy issues.

		average LP-gap in %									median time in seconds								
Set	$\overline{B}$	M1	M2	BP	FL	P1	P2	L1	L2	L3	M1	M2	BP	FL	P1	P2	L1	L2	L3
B-Ran	314	5.9	5.0	**0.0**	3.9	3.1	1.7	0.3	0.1	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	427	4.4	3.3	**0.0**	2.2	2.8	1.4	0.7	0.2	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
B-Cor	40	3.0	2.1	**0.0**	1.3	1.7	0.9	0.4	0.2	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	54	2.0	0.9	**0.0**	0.6	0.8	0.6	0.7	0.1	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
C-Ran	397	6.1	5.3	**0.0**	-	4.4	2.7	0.8	0.2	**0.0**	**0**	**0**	19	892	1	1	**0**	1	1
	541	4.1	2.8	**0.0**	-	2.6	1.9	0.6	0.1	-	**0**	**0**	49	125	1	2	39	47	226
C-Cor	50	3.9	2.4	**0.0**	-	1.9	1.3	1.7	0.2	**0.0**	**0**	**0**	24	513	1	2	**0**	1	2
	68	1.1	0.2	**0.0**	-	0.2	0.1	1.0	**0.0**	**0.0**	**0**	**0**	49	177	1	1	6	6	22
D-Ran	554	4.5	3.0	**0.0**	-	2.3	1.3	**0.0**	**0.0**	**0.0**	**0**	1	73	548	7	6	2	2	5
	755	2.8	1.0	**0.0**	-	0.9	0.5	1.0	**0.0**	-	**0**	**0**	140	317	5	5	180	384	1315
D-Cor	66	1.6	1.5	**0.0**	-	0.3	0.1	**0.0**	**0.0**	**0.0**	**0**	**0**	97	243	5	5	1	2	3
	90	1.1	0.1	**0.0**	**0.0**	0.1	**0.0**	1.2	**0.0**	**0.0**	**0**	1	217	107	4	5	16	20	28
Berlin-Ran	19	4.7	4.6	**0.2**	3.7	3.3	2.2	0.4	**0.2**	**0.2**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	26	7.3	6.6	**0.0**	5.6	5.8	3.4	0.1	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Berlin-Cor	165	13.7	3.3	**0.0**	1.5	1.9	1.5	7.7	0.3	**0.0**	**0**	**0**	1	**0**	**0**	**0**	**0**	2	19
	225	17.6	3.0	**0.0**	0.8	2.6	2.3	12.6	0.7	-	**0**	**0**	3	1	**0**	**0**	10	80	8021
Brazil-Ran	20	9.9	9.3	**0.0**	5.6	8.2	5.3	0.1	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	27	16.8	14.3	**0.0**	9.5	15.1	7.3	1.3	1.1	**0.0**	**0**	**0**	1	4	**0**	**0**	**0**	**0**	**0**
Brazil-Cor	3979	18.5	8.4	**0.0**	4.7	5.7	1.2	-	-	-	**0**	**0**	39	5	1	**0**	4731	-	-
	5425	15.7	3.9	**0.0**	1.3	3.1	1.5	-	-	-	**0**	**0**	76	4	**0**	**0**	-	-	-

Table 4.3: LP results for instances by Leggieri et al. [111] ($\overline{B}$: average delay-bound, LP-gap: gap between optimal LP relaxation and optimal integer value, M1: Miller-Tucker-Zemlin approach, M2: M1 with connection cuts, BP: stabilized branch-and-price, FL: flow approach, P1: path-cut approach, P2: P1 with lifted cuts, L1: layered graph approach, L2: L1 with connection cuts on G', L3: L1 with connection cuts on G'_{L}, best results are printed bold).

The results obtained by path-cut formulation PC' are quite surprising: From a theoretical point of view this formulation cannot compete with most of the other ones because of the rather weak infeasible path inequalities, see Section 4.7.1. However, in practice the strengthened inequalities (4.33) could significantly reduce the LP-gap outperforming the MTZ and flow formulations in most cases. At the same time the consumed runtime remains extremely low, i.e. below some seconds in all cases. Additionally, compared to the layered graph approaches the runtime of P2 is obviously independent of the delay-bound. To conclude, formulations with a low number of variables and some sets of strong valid inequalities added in the typical branch-and-cut way to the model seem to be highly promising from a practical point of view, cf. [96].

4.11.3 Branch-and-Cut Results

Tables 4.4–4.7 show results for all considered instance sets when solving the RDCST problem to optimality using the formulations within a branch-and-cut system. We report obtained average gaps between the best primal and dual bounds, the median runtime to reach these bounds, and the number of instances solved to optimality within the time limit. Dashes in gap and time columns represent 100% and 10 000 seconds, respectively. It may be surprising that for some instances the time to find the optimal solution is even less than the corresponding time for just solving the

		average gap in %									median time in seconds									# optimal solutions (out of 5)								
Set	B	M1	M2	BP	FL	P1	P2	L1	L2	L3	M1	M2	BP	FL	P1	P2	L1	L2	L3	M1	M2	BP	FL	P1	P2	L1	L2	L3
R5	6	**0.0**	**0.0**	**0.0**	4.7	3.0	**0.0**	**0.0**	**0.0**	**0.0**	7	18	2	-	1298	**0**	**0**	**0**	**0**	**5**	**5**	**5**	2	4	**5**	**5**	**5**	**5**
	8	**0.0**	**0.0**	**0.0**	3.0	6.8	**0.0**	**0.0**	**0.0**	**0.0**	32	330	3	6558	3624	1	**0**	**0**	**0**	**5**	**5**	**5**	3	3	**5**	**5**	**5**	**5**
	10	**0.0**	2.5	**0.0**	1.4	6.7	**0.0**	**0.0**	**0.0**	**0.0**	37	306	3	776	2998	2	**0**	**0**	**0**	**5**	4	**5**	4	3	**5**	**5**	**5**	**5**
	12	**0.0**	**0.0**	**0.0**	1.3	2.5	**0.0**	**0.0**	**0.0**	**0.0**	10	74	4	63	267	1	**0**	**0**	**0**	**5**	**5**	**5**	4	4	**5**	**5**	**5**	**5**
C5	6	**0.0**	**0.0**	**0.0**	1.5	1.5	**0.0**	**0.0**	**0.0**	**0.0**	3	12	2	3187	1620	**0**	**0**	**0**	**0**	**5**	**5**	**5**	3	3	**5**	**5**	**5**	**5**
	8	1.5	**0.0**	**0.0**	0.9	5.6	**0.0**	**0.0**	**0.0**	**0.0**	83	235	3	6125	-	1	**0**	**0**	**0**	4	**5**	**5**	3	2	**5**	**5**	**5**	**5**
	10	4.6	3.3	**0.0**	1.4	4.9	**0.0**	**0.0**	**0.0**	**0.0**	-	-	7	-	-	42	2	**1**	**1**	1	1	**5**	2	1	**5**	**5**	**5**	**5**
	12	5.1	2.7	**0.0**	0.5	4.3	**0.0**	**0.0**	**0.0**	**0.0**	-	-	7	2715	-	72	7	4	**3**	0	1	**5**	3	0	**5**	**5**	**5**	**5**
E5	6	**0.0**	1.6	**0.0**	6.5	15.2	**0.0**	**0.0**	**0.0**	**0.0**	924	4954	8	-	-	9	**1**	**1**	**1**	**5**	4	**5**	0	0	**5**	**5**	**5**	**5**
	8	11.3	10.7	**0.0**	6.7	20.4	1.5	**0.0**	**0.0**	**0.0**	-	-	18	-	-	163	6	6	**3**	0	0	**5**	0	0	4	**5**	**5**	**5**
	10	14.2	11.5	**0.0**	1.9	16.9	5.6	**0.0**	**0.0**	**0.0**	-	-	50	-	-	-	27	43	**6**	0	0	**5**	1	0	1	**5**	**5**	**5**
	12	14.2	11.5	**0.0**	2.3	14.1	7.1	0.7	0.3	**0.0**	-	-	157	-	-	-	193	274	**22**	0	0	**5**	2	0	0	4	4	**5**
R10	10	**0.0**	**0.0**	**0.0**	2.6	4.6	**0.0**	**0.0**	**0.0**	**0.0**	27	101	3	-	1042	1	**0**	**0**	**0**	**5**	**5**	**5**	2	4	**5**	**5**	**5**	**5**
	15	**0.0**	1.4	**0.0**	4.2	7.1	**0.0**	**0.0**	**0.0**	**0.0**	82	686	4	-	3532	2	**0**	**0**	**0**	**5**	4	**5**	1	3	**5**	**5**	**5**	**5**
	20	**0.0**	1.6	**0.0**	1.2	3.0	**0.0**	**0.0**	**0.0**	**0.0**	29	1252	4	2066	2081	4	**0**	**0**	**0**	**5**	4	**5**	4	4	**5**	**5**	**5**	**5**
	25	**0.0**	**0.0**	**0.0**	**0.0**	0.7	**0.0**	**0.0**	**0.0**	**0.0**	11	168	4	208	139	5	**1**	**1**	**1**	**5**	**5**	**5**	**5**	4	**5**	**5**	**5**	**5**
C10	10	**0.0**	**0.0**	**0.0**	2.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	15	89	4	-	951	**0**	**0**	**0**	**0**	**5**	**5**	**5**	1	**5**	**5**	**5**	**5**	**5**
	15	2.1	1.6	**0.0**	1.3	6.1	**0.0**	**0.0**	**0.0**	**0.0**	255	3572	4	5337	-	11	**1**	**1**	2	3	3	**5**	3	1	**5**	**5**	**5**	**5**
	20	5.2	3.2	**0.0**	**0.0**	3.4	**0.0**	**0.0**	**0.0**	**0.0**	-	-	**6**	2816	-	84	15	21	10	1	0	**5**	**5**	2	**5**	**5**	**5**	**5**
	25	3.8	0.9	**0.0**	0.3	1.2	**0.0**	**0.0**	**0.0**	**0.0**	-	5954	27	1575	2329	53	125	34	**22**	1	3	**5**	4	3	**5**	**5**	**5**	**5**
E10	10	2.6	2.0	**0.0**	4.9	13.2	**0.0**	**0.0**	**0.0**	**0.0**	-	6253	7	-	-	5	**1**	**1**	**1**	2	3	**5**	1	0	**5**	**5**	**5**	**5**
	15	14.2	12.5	**0.0**	6.9	20.5	7.0	0.6	**0.0**	**0.0**	-	-	43	-	-	-	25	30	**10**	0	0	**5**	0	0	0	4	**5**	**5**
	20	13.5	12.0	**0.0**	3.5	15.6	8.8	1.0	2.0	**0.0**	-	-	405	-	-	-	818	692	**83**	0	0	**5**	0	0	0	4	3	**5**
	25	12.0	10.8	**0.0**	3.0	13.3	11.9	6.9	4.7	**0.0**	-	-	1425	-	-	-	-	-	**317**	0	0	**5**	2	0	0	1	0	**5**

Table 4.4: Branch-and-cut results for instances by Gouveia et al. [68] (B: delay-bound, gap: gap between best primal and dual bound, M1: Miller-Tucker-Zemlin approach, M2: M1 with connection cuts, BP: stabilized branch-and-price, FL: flow approach, P1: path-cut approach, P2: P1 with lifted cuts, L1: layered graph approach, L2: L1 with connection cuts on G', L3: L1 with connection cuts on G'_{L}, best results are printed bold).

		average gap in %									median time in seconds								# optimal solutions (out of 5)								
Set	B	M1	M2	BP	FL	P1	P2	L1	L2	L3	M1	M2	BP	FL	P1	P2	L1	L2	L3	M1	M2	BP	FL	P1	P2	L1	L2	L3
R100	100	6.0	6.1	**0.0**	7.9	12.0	1.3	**0.0**	**0.0**	**0.0**	85	263	4	-	2688	**2**	4	5	5	4	4	**5**	2	3	4	**5**	**5**	**5**
	150	7.1	7.7	**0.0**	4.0	9.7	7.0	2.5	**0.0**	**0.0**	32	177	4	1556	539	**1**	9	10	10	4	4	**5**	4	4	4	4	**5**	**5**
	200	4.2	4.2	**0.0**	1.9	5.5	**0.0**	**0.0**	**0.0**	**0.0**	47	286	6	387	632	**1**	18	19	19	4	4	**5**	4	4	**5**	**5**	**5**	**5**
	250	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	7	83	6	44	47	**2**	51	59	62	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
C100	100	8.3	3.6	**0.0**	5.9	12.2	0.9	**0.0**	**0.0**	**0.0**	-	-	**8**	-	-	13	35	65	228	0	2	**5**	0	0	4	**5**	**5**	**5**
	150	7.3	4.9	**0.0**	1.5	6.9	**0.0**	**0.0**	**0.0**	0.9	-	-	**10**	-	-	71	476	1367	627	0	1	**5**	2	0	**5**	**5**	**5**	4
	200	6.0	2.3	**0.0**	0.2	3.7	1.2	2.3	1.2	2.1	-	-	**13**	4334	-	138	4567	1046	7747	0	1	**5**	4	2	3	3	4	3
	250	3.4	0.7	**0.0**	**0.0**	1.1	1.0	2.4	1.3	3.9	-	2428	**14**	59	422	17	-	2987	-	2	3	**5**	**5**	3	3	1	3	0
E100	100	13.0	13.3	**0.0**	7.6	16.3	1.2	**0.0**	**0.0**	**0.0**	-	-	**16**	-	-	884	419	591	593	0	0	**5**	0	0	3	**5**	**5**	**5**
	150	15.1	12.8	**0.0**	6.4	17.3	7.4	7.8	5.4	2.1	-	-	**40**	-	-	-	-	-	9162	0	0	**5**	0	0	1	0	0	3
	200	14.0	11.4	**0.0**	3.9	12.8	10.4	11.6	8.8	10.4	-	-	**245**	-	-	-	-	-	-	0	0	**5**	0	0	0	0	0	0
	250	12.9	9.5	**0.0**	1.6	10.5	10.0	10.3	7.8	11.7	-	-	**425**	-	-	-	-	-	-	0	0	**5**	2	0	0	0	0	0
R1000	1000	9.1	18.5	**0.0**	9.9	23.3	**0.0**	**0.0**	**0.0**	**0.0**	974	-	12	-	-	**6**	272	306	245	3	1	**5**	1	0	**5**	**5**	**5**	**5**
	1500	15.1	18.5	**0.0**	8.8	21.0	**0.0**	3.0	3.1	2.4	-	-	**18**	-	-	132	4370	4117	9586	1	1	**5**	1	1	**5**	4	4	3
	2000	6.7	5.3	**0.0**	**0.0**	7.4	3.2	**0.0**	**0.0**	0.9	241	2316	**20**	7966	1155	**20**	2855	3632	3548	3	3	**5**	**5**	3	4	**5**	**5**	4
	2500	1.0	1.9	**0.0**	**0.0**	2.1	**0.0**	20.0	1.8	20.0	17	342	25	88	91	**1**	5815	5603	6099	4	4	**5**	**5**	4	**5**	4	4	4
C1000	1000	4.4	3.8	**0.0**	4.7	9.8	1.2	2.0	1.6	2.7	-	7714	17	-	-	**5**	458	478	6018	2	3	**5**	1	1	4	4	4	3
	1500	11.2	6.1	**0.0**	2.2	12.6	**0.0**	4.0	3.1	6.8	-	-	**29**	-	-	52	-	-	-	0	0	**5**	2	0	**5**	1	2	0
	2000	10.3	7.1	**0.0**	2.6	10.3	4.6	26.5	28.0	13.2	-	-	**39**	-	-	-	-	-	-	0	0	**5**	2	0	0	0	0	0
	2500	5.8	2.3	**0.0**	0.6	1.7	1.3	64.2	64.5	64.5	-	-	**33**	677	1952	96	-	-	-	0	2	**5**	4	3	3	0	0	0
E1000	1000	12.9	9.9	**0.0**	7.6	17.2	3.3	6.0	6.0	8.9	-	-	**26**	-	-	449	-	-	-	0	1	**5**	0	1	3	1	1	1
	1500	15.5	11.6	**0.0**	5.8	17.2	6.4	10.3	15.1	16.1	-	-	**50**	-	-	-	-	-	-	0	0	**5**	1	0	2	0	0	0
	2000	15.7	13.4	**0.0**	2.4	15.6	12.8	31.2	49.4	33.4	-	-	**88**	-	-	-	-	-	-	0	0	**5**	1	0	0	0	0	0
	2500	13.8	10.0	**0.0**	0.5	12.1	10.6	-	-	-	-	-	**218**	8067	-	-	-	-	-	0	0	**5**	4	0	0	0	0	0

Table 4.5: Branch-and-cut results for instances by Gouveia et al. [68] (B: delay-bound, gap: gap between best primal and dual bound, M1: Miller-Tucker-Zemlin approach, M2: M1 with connection cuts, BP: stabilized branch-and-price, FL: flow approach, P1: path-cut approach, P2: P1 with lifted cuts, L1: layered graph approach, L2: L1 with connection cuts on G', L3: L1 with connection cuts on G'_{L}, best results are printed bold).

		average gap in %									median time in seconds									# optimal solutions (out of 30)								
Set	B	M1	M2	BP	FL	P1	P2	L1	L2	L3	M1	M2	BP	FL	P1	P2	L1	L2	L3	M1	M2	BP	FL	P1	P2	L1	L2	L3
T10	16	**0.0**	**0.0**	**0.0**	**0.0**	0.3	**0.0**	**0.0**	**0.0**	**0.0**	1	1	**0**	7	4	1	**0**	**0**	**0**	**30**	**30**	**30**	**30**	29	**30**	**30**	**30**	**30**
	30	8.2	12.7	**0.0**	**0.0**	25.3	10.7	**0.0**	**0.0**	**0.0**	63	-	2	147	-	643	**1**	**1**	**1**	20	13	**30**	**30**	7	17	**30**	**30**	**30**
	50	14.7	23.4	**0.0**	0.6	33.6	21.4	**0.0**	**0.0**	**0.0**	5469	-	**3**	749	-	-	7	7	7	15	6	**30**	29	1	6	**30**	**30**	**30**
	100	22.2	31.7	**0.0**	2.1	38.8	29.1	0.4	0.9	**0.0**	-	-	**5**	1805	-	-	76	72	76	9	1	**30**	24	1	4	29	29	**30**
T30	16	**0.0**	0.8	**0.0**	2.9	1.6	**0.0**	**0.0**	**0.0**	**0.0**	4	7	2	5538	58	3	**0**	**0**	**0**	**30**	28	**30**	19	27	**30**	**30**	**30**	**30**
	30	14.4	18.8	**0.0**	15.1	34.0	14.5	**0.0**	**0.0**	**0.0**	-	-	9	-	-	-	**2**	**2**	**2**	8	3	**30**	0	0	12	**30**	**30**	**30**
	50	26.2	32.6	**0.0**	19.5	43.1	27.1	**0.0**	**0.0**	**0.0**	-	-	**20**	-	-	-	**20**	21	34	0	0	**30**	0	0	3	**30**	**30**	**30**
	100	36.5	37.6	**0.0**	20.7	47.3	36.7	0.4	0.6	**0.0**	-	-	**47**	-	-	-	142	129	362	0	0	**30**	0	0	0	29	28	**30**
T50	16	**0.0**	**0.0**	**0.0**	11.4	1.6	**0.0**	**0.0**	**0.0**	**0.0**	10	25	5	-	501	4	**0**	**0**	**0**	**30**	**30**	**30**	1	24	**30**	**30**	**30**	**30**
	30	16.6	19.1	**0.0**	19.5	34.0	6.6	**0.0**	**0.0**	**0.0**	-	-	19	-	-	590	**4**	**4**	**4**	2	1	**30**	0	0	19	**30**	**30**	**30**
	50	28.4	29.0	**0.0**	20.2	43.8	22.7	**0.0**	**0.0**	**0.0**	-	-	44	-	-	-	23	**20**	33	0	0	**30**	0	0	6	**30**	**30**	**30**
	100	36.3	36.9	**0.0**	42.1	47.0	36.1	0.7	0.8	**0.0**	-	-	**202**	-	-	-	320	282	768	0	0	**30**	0	0	0	27	27	**30**
T70	16	**0.0**	0.4	**0.0**	13.1	3.0	**0.0**	**0.0**	**0.0**	**0.0**	14	57	8	-	1090	1	**0**	**0**	**0**	**30**	28	**30**	0	21	**30**	**30**	**30**	**30**
	30	16.0	19.0	**0.0**	21.0	32.8	2.7	**0.0**	**0.0**	**0.0**	-	-	31	-	-	262	**4**	**4**	**4**	0	0	**30**	0	0	23	**30**	**30**	**30**
	50	26.1	30.0	**0.0**	33.9	42.4	17.3	**0.0**	**0.0**	**0.0**	-	-	79	-	-	-	**23**	26	50	0	0	**30**	0	0	9	**30**	**30**	**30**
	100	33.3	36.2	**0.1**	95.4	47.0	32.5	0.7	0.7	0.9	-	-	**241**	-	-	-	254	318	873	0	0	28	0	0	0	28	27	**29**
T99	16	0.2	0.5	**0.0**	17.0	4.1	**0.0**	**0.0**	**0.0**	**0.0**	46	195	15	-	7462	**0**	**0**	**0**	**0**	28	27	**30**	0	15	**30**	**30**	**30**	**30**
	30	14.6	15.6	**0.0**	30.3	29.2	0.3	**0.0**	**0.0**	**0.0**	-	-	66	-	-	17	**3**	**3**	4	0	0	**30**	0	0	29	**30**	**30**	**30**
	50	23.3	26.9	**0.0**	-	39.7	6.3	**0.0**	**0.0**	**0.0**	-	-	150	-	-	401	**22**	24	44	0	0	**30**	0	0	18	**30**	**30**	**30**
	100	31.6	31.8	**0.3**	-	42.9	16.5	0.4	**0.3**	0.6	-	-	526	-	-	-	254	**250**	719	0	0	**28**	0	0	2	27	**28**	**28**

Table 4.6: Branch-and-cut results for random instances from Section 3.13.1 (B: delay-bound, gap: gap between best primal and dual bound, M1: Miller-Tucker-Zemlin approach, M2: M1 with connection cuts, BP: stabilized branch-and-price, FL: flow approach, P1: path-cut approach, P2: P1 with lifted cuts, L1: layered graph approach, L2: L1 with connection cuts on G', L3: L1 with connection cuts on G'_{L}, best results are printed bold).

		average gap in %									median time in seconds									# optimal solutions (out of 18/10/5/5/5)								
Set	$\overline{B}$	M1	M2	BP	FL	P1	P2	L1	L2	L3	M1	M2	BP	FL	P1	P2	L1	L2	L3	M1	M2	BP	FL	P1	P2	L1	L2	L3
B-Ran	314	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**
	427	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**
B-Cor	40	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**
	54	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**
C-Ran	397	0.9	0.8	**0.0**	21.8	2.7	0.3	**0.0**	**0.0**	**0.0**	12	123	18	-	77	14	**1**	**1**	2	8	8	**10**	4	7	9	**10**	**10**	**10**
	541	1.9	2.4	**0.0**	21.8	3.9	2.5	**0.0**	**0.0**	**0.0**	19	30	48	5415	10	**6**	89	93	106	7	7	**10**	5	7	7	**10**	**10**	**10**
C-Cor	50	0.3	0.4	**0.0**	21.3	0.7	0.1	**0.0**	**0.0**	**0.0**	9	38	23	-	52	15	**1**	**1**	2	8	8	**10**	4	7	9	**10**	**10**	**10**
	68	**0.0**	**0.0**	**0.0**	10.1	0.3	**0.0**	**0.0**	**0.0**	**0.0**	7	**2**	48	333	13	3	11	8	21	**10**	9	**10**	8	9	**10**	**10**	**10**	**10**
D-Ran	554	0.3	2.0	**0.0**	40.2	0.3	**0.0**	**0.0**	**0.0**	**0.0**	11	110	76	-	105	44	**3**	**3**	**3**	3	3	**5**	2	3	**5**	**5**	**5**	**5**
	755	0.1	0.1	**0.0**	1.9	0.1	**0.0**	**0.0**	**0.0**	0.1	**7**	248	145	-	95	20	442	481	1725	4	3	**5**	2	4	**5**	**5**	**5**	4
D-Cor	66	**0.0**	**0.0**	**0.0**	20.9	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	6	17	113	1572	36	16	**2**	**2**	**2**	**5**	**5**	**5**	3	**5**	**5**	**5**	**5**	**5**
	90	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**1**	**1**	199	999	10	8	27	32	32	**5**	**5**	**5**	4	**5**	**5**	**5**	**5**	**5**
Berlin-Ran	19	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	26	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	3	**0**	**0**	**0**	**0**	**0**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
Berlin-Cor	165	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	2	1	1	2	1	**0**	9	4	7	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	225	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	2.4	**0.0**	3.3	58	**1**	4	3	**1**	**1**	-	115	5155	**5**	**5**	**5**	**5**	**5**	**5**	2	**5**	4
Brazil-Ran	20	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	15	1	**0**	**0**	**0**	**0**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	27	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	2	5	1	401	16	4	**0**	**0**	**0**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
Brazil-Cor	3979	7.6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	33.4	57.0	39.4	-	2	44	11	3	**1**	-	-	-	0	**5**	**5**	**5**	**5**	**5**	0	0	0
	5425	10.3	0.8	**0.0**	**0.0**	**0.0**	**0.0**	-	-	-	-	191	67	**14**	25	48	-	-	-	0	3	**5**	**5**	**5**	**5**	0	0	0

Table 4.7: Branch-and-cut results for instances by Leggieri et al. [111] ($\overline{B}$: average delay-bound, gap: gap between best primal and dual bound, M1: Miller-Tucker-Zemlin approach, M2: M1 with connection cuts, BP: stabilized branch-and-price, FL: flow approach, P1: path-cut approach, P2: P1 with lifted cuts, L1: layered graph approach, L2: L1 with connection cuts on G', L3: L1 with connection cuts on G'_{L}, best results are printed bold).

LP relaxation, see Section 4.11.2. This is because of the built-in features of modern MIP solvers like CPLEX and SCIP, e.g. sophisticated presolving methods, general purpose mixed integer cuts, and efficient primal heuristics, see [120] for a general overview of acceleration techniques for MIP solvers.

The overall picture looks similar to the LP results in the previous Section 4.11.2: When considering dense graphs with large sets of achievable edge delays and high delay-bounds in Table 4.5 the branch-and-price approach significantly outperforms all other methods. However, regarding some of these instances path-cut approach P2 is close on BP's heels, sometimes it is even slightly faster. All other formulations including layered graph models are out of question, here.

However, on instance sets with small delay-bounds, see Tables 4.4, the layered graph approaches seem to be more suitable than BP. Here, the small size of corresponding layered graphs allow both extremely tight bounds and efficient computation of them. It is interesting to see that e.g. for the instances in Table 4.4 it definitely makes sense to add directed connection cut inequalities (4.50) on layered graph G'_{L} to tighten the LP bounds, but in Tables 4.5–4.7 variants L1 and L2 are superior to L3. Here, the additional overhead of cut separation and the repeated LP resolvings does not pay off.

The picture is not clear for the experiments in Table 4.7: BP is the only one which can solve all instances to optimality within the given time limit. However, other approaches like P2, L1-3, or even M1, have lower average computation times. But if we take the slightly worse general performance of SCIP in comparison to CPLEX into account, runtimes get similar again. In other words, no clear winner can be determined for these instances.

To conclude, the formulations based on MTZ and simple path-cut inequalities provide too weak bounds to be competitive and the multi-commodity flow formulation typically includes too many variables. The rest of the approaches – branch-and-price, layered graph approaches, and the path-cut formulation with lifted inequalities – all have their strengths and weaknesses: BP provides robust performance throughout all tests. The formulations based on layered graphs provide the tightest bounds and are leading on sets with small delay-bounds but are unusable for large delays and bounds. Finally, the lifted path-cut approach P2 does not obtain the best LP bounds but due to the small number of variables and the fact that the model stays quite small throughout the branch-and-cut process this approach is still competitive in some cases.

4.12 Future Work

We have seen that the size of the corresponding layered graph is crucial for the performance of these approaches. Thus, we aim at extending our preprocessing methods to further reduce graph G_{L}. It is definitely worth to spend more runtime here applying more sophisticated and complex reduction rules which may additionally consider costs to eliminate suboptimal arcs and nodes. A different approach to deal with large graph sizes will be discussed in Chapter 6.

The simple formulation based on lifted infeasible path inequalities yields surprisingly good results. Thus, we want to find further strengthenings of these inequalities or maybe find other similar sets of valid inequalities. Currently, edge delays are not incorporated in the inequalities but considered in a more abstract way in the definition of infeasible paths. However, we also

may sum up the delays of used edges within the constraint and then ensure the satisfaction of the delay-bound. We already considered inequalities of this type but they are still in a preliminary and rather weak stage.

Similarly as Gouveia et al. [70] did for the HCMST problem, we should further think about the reasons why layered graph formulations are that strong. Maybe we are able to find new sets of valid inequalities projected from the layered graph space into the space of original arc variables. These additional inequalities can then be used to further strengthen formulation PC.

Within a branch-and-bound system it is highly important for the overall performance to provide strong primal bounds, i.e. feasible solutions. Whenever an incumbent solution is found it could be improved by heuristics, e.g. by a variable neighborhood descent. The hopefully better bound could then be used as new cut-off value for subtree pruning and the corresponding tree as guiding solution for MIP solver internal heuristics.

There are already many articles presenting heuristic approaches for the RDCST problem. However, it might be worth to adapt our methods presented in Chapter 3 for the RDCMST problem accordingly and apply them on the RDCST problem. This is not a trivial task since successful heuristics for the RDCST problem usually consider potential Steiner nodes in a special way, e.g. it is common for genetic algorithms for Steiner tree problems to use a binary vector for the optional nodes deciding which optional nodes are included in the tree and which not. Clearly, even if we know the exact set of nodes included in the solution the problem to decode one specific vector to an according optimal delay-constrained tree is still $\mathcal{NP}$-hard in our case since then we are actually faced with the RDCMST problem.

CHAPTER 5

Rooted Delay- and Delay-Variation-Constrained Steiner Tree Problem

This chapter discusses several exact mixed integer programming approaches for solving the *rooted delay- and delay-variation-constrained Steiner tree (RDDVCST) problem*. Section 5.1 formally defines the problem and Section 5.2 summarizes previous related work. Modifications and extensions to the reduction techniques for the RDCST problem from Section 4.3 are proposed in Section 5.3. In Section 5.4 we present a formulation based on multi-commodity flows for the RDDVCST problem. The transformation to a layered graph and a corresponding model is described in Section 5.5 and 5.6, respectively. The presented models are compared theoretically in Section 5.7 and practically in Section 5.8. Finally, Section 5.9 discusses open problems and possible future research directions. Most parts are based on the published article [165].

5.1 Problem Definition

The RDDVCST problem is a generalization of the RDCST problem discussed in Chapter 4 in which we have to satisfy an additional constraint relating the path-delays to different terminal nodes.

More formally, we are given an undirected graph $G = (V, E)$ with node set V, a fixed root node $s \in V$, set $R \subseteq V \setminus \{s\}$ of terminal or required nodes, set $S = V \setminus (R \cup \{s\})$ of potential Steiner nodes, edge set E, a cost function $c : E \to \mathbb{Z}_0^+$, a delay function $d : E \to \mathbb{Z}^+$, a delay bound $B \in \mathbb{Z}^+$, and a delay-variation-bound $D \in \mathbb{Z}_0^+$. A feasible solution to the RDDVCST problem is a Steiner tree $T = (V', E'),\ s \in V',\ R \subset V' \subseteq V,\ E' \subseteq E$, satisfying the delay-constraints

$$d_v^T = \sum_{e \in P_T(s,v)} d_e \leq B,\ \forall v \in R, \tag{5.1}$$

where $P_T(s,v)$ denotes the unique path from the specified root node s to terminal node $v \in R$ in Steiner tree T and d_v^T the corresponding total delay on this path. We further limit the difference between the path-delays to any two terminal nodes by the constraint

$$\max_{u,v \in R} |d_u^T - d_v^T| \leq D. \tag{5.2}$$

Finally, we define the cost function

$$c_T = \sum_{e \in E'} c_e, \tag{5.3}$$

summing up the cost values of all edges in a solution T. An optimal solution T^* to the RDDVCST problem is a feasible solution with minimal total edge costs, i.e. $c_{T^*} \leq c_T,\ \forall T$.

Similarly to the RDCST problem, we define a directed variant of this problem on graph $G' = (V, A)$ with arc set $A = \{(s,v) : \{s,v\} \in E\} \cup \{(u,v),(v,u) : \{u,v\} \in E,\ u,v \neq s\}$ consisting of two opposite arcs for each edge in graph G except for edges incident to root node s, for which we include only the corresponding arc going out from s. A feasible solution to the directed variant is a Steiner arborescence $T' = (V', A'),\ s \in V',\ R \subset V' \subseteq V,\ A' \subset A$, directed out of root node s. It can be easily seen that each feasible Steiner tree T bijectively corresponds to a feasible Steiner arborescence T'.

The RDDVCST problem is $\mathcal{NP}$-hard because the RDCST problem, where $D = B$, is an $\mathcal{NP}$-hard special case, see Chapter 4. Furthermore, Rouskas and Baldine [160] showed that even the problem of finding a feasible solution without considering the costs is $\mathcal{NP}$-hard.

A lower bound to the optimal cost value clearly is provided by relaxing the delay- and/or the delay-variation-constraints, e.g. resulting in a minimal-cost Steiner tree T^{l} without considering any constraints on delays. If such a tree T^{l} is feasible for the RDDVCST problem, i.e. satisfies the delay- and delay-variation-constraints, then T^{l} also is an optimal solution for it. However, finding an optimal Steiner tree T^{l} is still $\mathcal{NP}$-hard. In contrast to the RDCMST and RDCST problem, we are not able to construct a trivial feasible solution here, due to the $\mathcal{NP}$-hardness result mentioned above.

Instead of using $c_{\{u,v\}}$ and $d_{\{u,v\}}$ to denote cost and delay values assigned to edge $\{u,v\} \in E$, we use the better readable notation c_{uv} and d_{uv}, respectively. The same holds for arcs $(u,v) \in A$ in directed graph G'. Variable $d_v,\ v \in V$, refers to the node delay with respect to one specific tree T. In case of multiple solutions the considered tree is explicitly included in the notation, i.e. $d_v^T,\ v \in V$.

5.2 Related Work

Rouskas and Baldine [159, 160] introduce a variant of the RDDVCST problem called *delay- and delay-variation-bounded multicast tree (DVBMT)* problem. In it the aim is to just find a feasible tree satisfying both the delay- and delay-variation-constraints without considering edge costs at all. To solve the DVBMT problem the authors present a construction heuristic with relatively high runtime complexity again adapting the concept by Takahashi and Matsuyama [177], cf. Section 4.2: They start with a feasible path to one terminal node and iteratively connect the rest of the terminals in feasible ways as long as possible by computing k-shortest-delay-paths. Haberman and Rouskas [77] tackle the RDDVCST problem for the first time and present

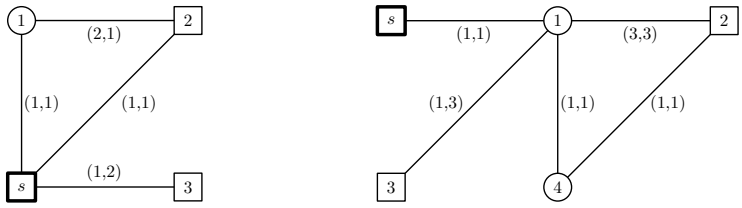


Figure 5.1: Example graphs with $B = 4,\ D = 0$, and squared nodes representing terminal nodes: (a) Removing edge $\{1, 2\}$ according to reduction rule (3.4) results in an infeasible instance. (b) Similarly, removing edge $\{1, 2\}$ according to Theorem 3.3.1 also leads to infeasibility.

a heuristic similar to the one in [160] but additionally considering edge costs. Lee et al. [109] provide another construction heuristic: first, the shortest-delay-paths to all terminals are combined to form a tree naturally satisfying the delay-constraint. Second, tree costs are reduced possibly violating delay- and delay-variation-constraints. Not feasibly connected terminals are then removed and re-added to the tree by low-delay paths. Low et al. [121] present a two phase construction approach: in the first phase a tree is obtained by only considering the costs and the delay-constraint. If the delay-variation-constraint is violated in this solution the second phase searches for alternative paths in a distributed way. Sheu et al. [171] improve the worst-case time complexity of the heuristic in [160] for the DVBMT problem still obtaining high quality solutions in the sense that the delay-variation is quite low. A further construction heuristic avoiding drawbacks of previously proposed approaches can be found in [10]. Zhang et al. [108] propose a simulated annealing approach for the RDDVCST problem using a path-based solution encoding scheme and a path-exchange neighborhood only allowing feasible moves. Several of the mentioned articles also discuss the complexity of the solution modification if nodes join or leave the multicast group.

To the best of our knowledge only one MIP formulation exists so far for another problem variant in which the delay-variation D is minimized while satisfying the delay-constraints without considering edge costs: Sheu et al. [172] present an MCF formulation, which we revise and adapt to the RDDVCST problem in Section 5.4.

5.3 Preprocessing

Clearly, we can apply all procedures to eliminate infeasible edges described in Section 3.3.1 and 4.3 for the RDCMST and RDCST problem, respectively. However, unfortunately it is not feasible here to use methods removing suboptimal edges as defined in Section 3.3.2. In some cases we may have to choose more expensive paths with higher delay to a terminal node to satisfy the delay-variation-constraint. In other words, smaller delay combined with smaller cost does not imply a better connection anymore. Figure 5.1 shows two of these situations where preprocessing based on root edges (3.4) and alternative triangles subject to Theorem 3.3.1 is

invalid, respectively. A similar counterexample can easily be found for the reduction procedure based on alternative paths.

However, we are still able to utilize the delay-variation-bound to further reduce graph G by removing all edges connecting two terminal nodes with $d_e > D$ because they clearly cannot appear in any feasible solution. Additionally, in graph G' we can safely remove all arcs $(u, v) \in A$ going out of a terminal node $u \in R$ with $d_{uv} > D$.

5.4 Multi-Commodity Flow Formulation

The following formulation extends the multi-commodity flow formulation for the RDCST problem in Section 4.6 by additionally considering the delay-variation-constraint. We use binary decision variables x_{uv}, $\forall (u, v) \in A$. Furthermore, real-valued flow variables f_{uv}^w, $\forall (u, v) \in A$, $\forall w \in R$, denote the flow on arc (u, v) from root s to terminal w. The minimal path-delay is described by variable $\delta_{\min}$. Model MCF is defined as follows:

$$\min \quad \sum_{(u,v)\in A} c_{uv} x_{uv} \tag{5.4}$$

$$\text{s.t.} \quad \sum_{(u,v)\in A} f_{uv}^w - \sum_{(v,u)\in A} f_{vu}^w = \begin{cases} -1 & \text{if } v = s \\ 1 & \text{if } v = w \\ 0 & \text{else} \end{cases} \qquad \forall w \in R \tag{5.5}$$

$$\delta_{\min} \leq \sum_{(u,v)\in A} d_{uv} f_{uv}^w \leq \delta_{\min} + D \qquad \forall w \in R \tag{5.6}$$

$$\delta_{\min} \in [1, B - D] \tag{5.7}$$

$$0 \leq f_{uv}^w \leq x_{uv} \qquad \forall (u, v) \in A,\ \forall w \in R \tag{5.8}$$

$$x_{uv} \in \{0, 1\} \qquad \forall (u, v) \in A \tag{5.9}$$

Classical flow constraints (5.5) describe the flow of one commodity for each terminal $w \in R$ originating in root s, possibly passing any nodes in $V \setminus \{s, w\}$, and ending in target node w, respectively. Constraints (5.6) add up the delays on the path to a terminal and define lower and upper delay-bounds over all required nodes respecting the delay-variation D. Since variable $\delta_{\min}$ is restricted to $[1, B - D]$, inequalities (4.23) are included in (5.6) and thus delay-bound B is satisfied implicitly. Finally, linking constraints (5.8) connect flow and arc variables.

Providing edge costs are strictly positive, objective (5.4) together with constraints (5.5), (5.8) and (5.9) describe optimal Steiner trees in directed graphs, cf. [124]. However, by adding constraints (5.6) and (5.7) detached cycles consisting of Steiner nodes may occur in an optimal solution to model MCF, see Fig. 5.2b: arcs $(0, 1)$ and $(1, 2)$ connect both terminal nodes to the root within the given delay-bound $B = 4$ but result in a delay-variation of $D = 3$. Instead of using optimal arcs $(0, 1)$ and $(0, 2)$, see Fig. 5.2a, it is cheaper and feasible in model MCF to add a circular flow for terminal 1 on the detached cycle $(3, 4, 5)$, so $f_{01}^1 = f_{34}^1 = f_{45}^1 = f_{53}^1 = 1$ and $f_{01}^2 = f_{12}^2 = 1$. Due to constraints (5.6) the "path-delay" to node 1 is now increased to 4 and thus $D = 0$. To prevent infeasible solutions we guarantee root connectivity for all used Steiner nodes. Therefore, we add sets of flow variables and constraints for each potential Steiner node.

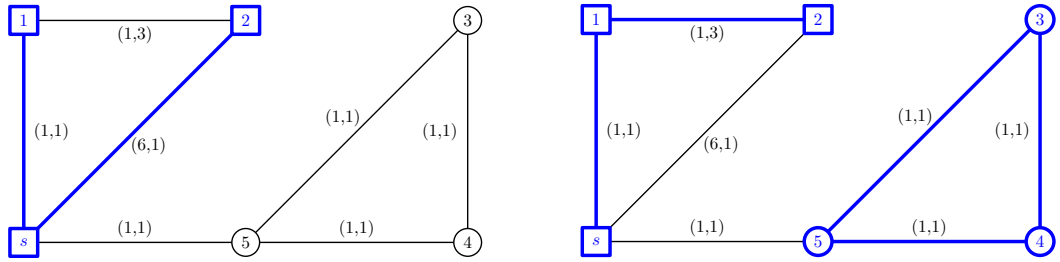


Figure 5.2: (a) Example graph G with edge labels (c_e, d_e). Squared nodes denote terminal nodes and blue edges show the optimal solution for $B = 4,\ D = 0$, with $c_T = 7$. (b) The optimal solution to model MCF has costs $c_T = 5$ but is infeasible for the RDDVCST problem.

But only if there is an incoming arc to a Steiner node the corresponding flow is activated. This finally feasible model MCF' extends MCF by:

$$\sum_{(u,v)\in A} f_{uv}^w - \sum_{(v,u)\in A} f_{vu}^w = \begin{cases} -\sum\limits_{(u,w)\in A} x_{uw} & \text{if } v = s \\ \sum\limits_{(u,w)\in A} x_{uw} & \text{if } v = w \\ 0 & \text{else} \end{cases} \qquad \forall w \in S \quad (5.10)$$

$$0 \le f_{uv}^w \le x_{uv} \qquad \forall (u,v) \in A,\ \forall w \in S \quad (5.11)$$

Flow constraints (5.10) for optional nodes are similar to the counterparts (5.5) for terminal nodes but extended by indegree terms to optionally enable or disable the corresponding flows.

5.5 Transformation to Layered Graph

The transformation of graph $G' = (V, A)$ to a layered graph $G_{\mathrm{L}} = (V_{\mathrm{L}}, A_{\mathrm{L}})$ works exactly in the same way as for the RDCST problem, see Section 4.8 for further details. Again, we want to find an arborescence $T_{\mathrm{L}} = (V_{\mathrm{L}}^T, A_{\mathrm{L}}^T)$ in G_{L} with $V_{\mathrm{L}}^T \subseteq V_{\mathrm{L}},\ A_{\mathrm{L}}^T \subseteq A_{\mathrm{L}}$, rooted in $s \in V_{\mathrm{L}}^T$, including exactly one node $v_l \in V_{\mathrm{L}}^T$ for each terminal node $v \in R$ and at most one node $u_l \in V_{\mathrm{L}}^T$ for each potential Steiner node $u \in S$, having minimal costs $c_{T_{\mathrm{L}}} = \sum_{(u_k,v_l)\in A_{\mathrm{L}}^T} c_{uv}$. Additionally, we have to satisfy the transformed delay-variation-constraint

$$\max_{u_k, v_l \in V_{\mathrm{L}}^T,\ u,v \in R} |k - l| \le D. \quad (5.12)$$

Due to its possibly huge size preprocessing in G_{L} is even more important than in G'. Let $\deg^-(u_k)$ and $\deg^+(u_k)$ denote the indegree and outdegree of node u_k, respectively. The following reduction steps are repeated as long as G_{L} is modified by one of them:

1. A node $v_l \in V_{\mathrm{L}},\ v \in R$, is removed if $\exists u \in R \setminus \{v\}$ with $u_k \notin V_{\mathrm{L}},\ \forall k \in \{l - D, l + D\}$, since v_l cannot be in any feasible solution.

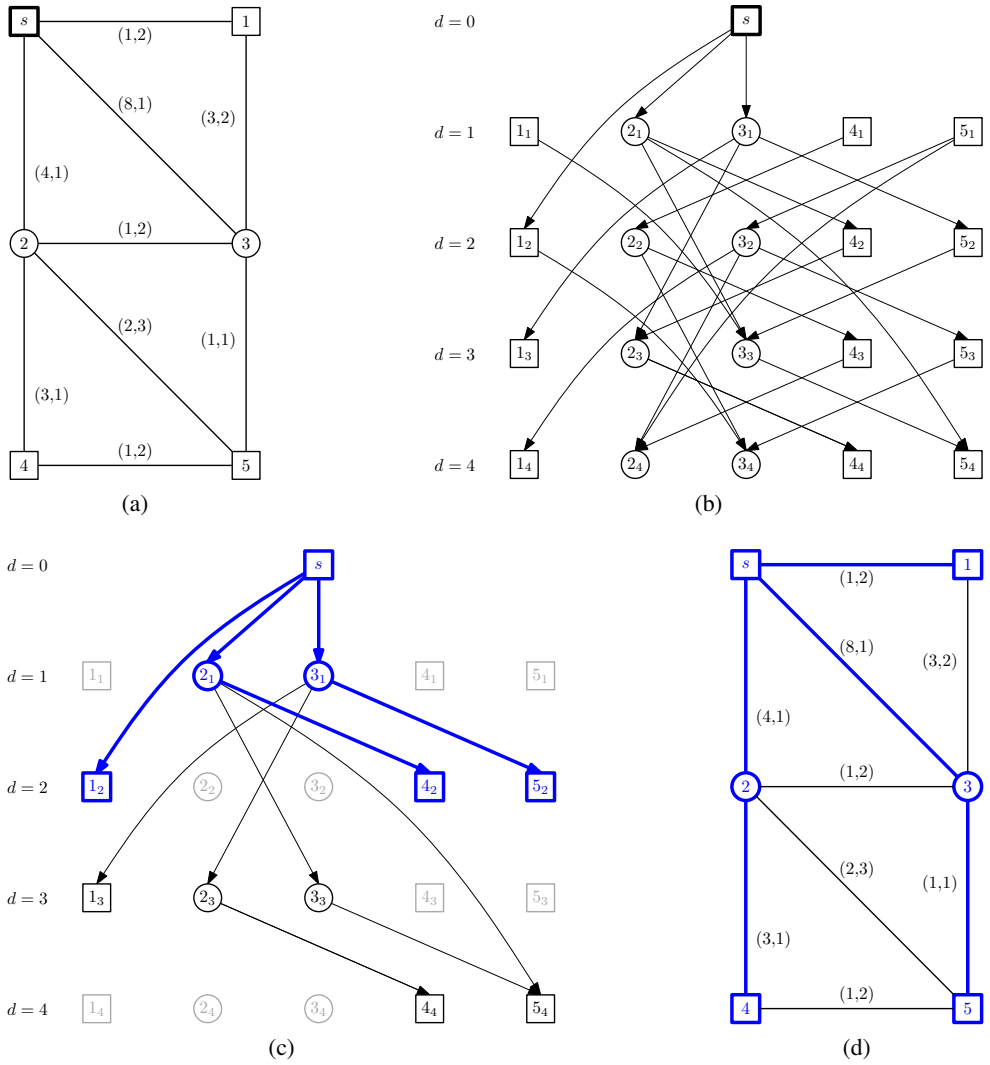


Figure 5.3: (a) Example graph with edge labels (c_e, d_e). Squared nodes denote terminal nodes. (b) Corresponding layered digraph for $B = 4$ and $D = 1$ (arc costs are omitted). (c) Preprocessed graph G_{L} with optimal solution denoted by blue arcs. (d) Optimal tree T^* in G with $c_{T^*} = 17$.

2. To partly prevent cycles of length two in G' an arc $(u_k, v_l) \in A_{\mathrm{L}}$ can be removed if $\deg^-(u_k) = 1 \wedge (v_m, u_k) \in A_{\mathrm{L}}$ or $v \in S \wedge \deg^+(v_l) = 1 \wedge (v_l, u_m) \in A_{\mathrm{L}}$.

3. If node $v_l \in V_{\mathrm{L}} \setminus \{s\}$ has no incoming arcs it cannot be reached from s and therefore is removed.

4. If node $v_l \in V_{\mathrm{L}} \setminus \{s\},\ v \in S$, has no outgoing arcs it is removed since a Steiner node cannot be a leaf in an optimal solution.

A naive implementation of the first rule would examine for each layered node $v_l \in V_{\mathrm{L}},\ v \in R$, all other nodes $u_k \in V_{\mathrm{L}},\ u \in R \setminus \{s\},\ k \in \{l - D, l + D\}$, running in $\mathcal{O}(|V_{\mathrm{L}}| \cdot |R| \cdot D)$ time. We significantly improved this performance by using a sweep algorithm basically dragging a "window" with width $2D + 1$ through the layers of G_{L}. In each window position we update the set of active terminal nodes within that interval. If at any time the number of active nodes falls below $|R|$ then all nodes $v_l \in V_{\mathrm{L}},\ v \in R$, on layer l in the middle of the current window can be removed. This improved variant runs in $\mathcal{O}(|V_{\mathrm{L}}|)$ time.

It was interesting to see in preliminary tests that the complete reduction procedure is sometimes able to eliminate the whole layered graph if the delay-variation-constraint is set too tight to allow a feasible solution. On the same instances a branch-and-cut approach based on model MCF' had to solve thousands of branch-and-bound nodes to prove infeasibility. Thus, the layered graph transformation and reduction can be seen as a heuristic method for detecting infeasibility in an early stage of the solving process.

See Fig. 5.3 for an example of layered graph transformation, preprocessing, and solution correspondence. Note that example graph G is the same as in Fig. 4.2 for the RDCST problem but the cost of an optimal solution increases from 9 to 17 by imposing a delay-variation-constraint with $D = 1$.

5.6 Layered Graph Formulation

We extend the model presented in Section 4.9 for the RDCST problem without delay-variation-constraint. Again, we use binary variables $x_{uv},\ \forall (u, v) \in A$, to model original arcs in G'. Additionally, continuous variables $y_v^l,\ \forall v_l \in V_{\mathrm{L}} \setminus \{s\}$, and $x_{uv}^l,\ \forall (u_l, v_k) \in A_{\mathrm{L}}$, represent nodes and arcs in layered graph G_{L}, respectively. Model LAY is then defined as follows:

$$\min \quad \sum_{(u,v) \in A} c_{uv} x_{uv} \tag{5.13}$$

$$\text{s.t.} \quad \sum_{v_l \in V_L} y_v^l = 1 \qquad \forall v \in R \tag{5.14}$$

$$\sum_{v_l \in V_L} y_v^l \leq 1 \qquad \forall v \in S \tag{5.15}$$

$$\sum_{(u_k, v_l) \in A_{\mathrm{L}}} x_{uv}^k = y_v^l \qquad \forall v_l \in V_{\mathrm{L}} \setminus \{s\} \tag{5.16}$$

$$\sum_{(u_k,v_l)\in A_{\mathrm{L}},u\neq w} x^k_{uv} \geq x^l_{vw} \qquad \forall (v_l, w_j) \in A^g_{\mathrm{L}} \tag{5.17}$$

$$x^0_{sv} = x_{sv} \qquad \forall (s,v) \in A \tag{5.18}$$

$$\sum_{(u_k,v_l)\in A_{\mathrm{L}}} x^k_{uv} = x_{uv} \qquad \forall (u,v) \in A,\ u \neq s \tag{5.19}$$

$$\delta_{\min} \leq \sum_{l=1}^{B} l \cdot y^l_v \leq \delta_{\min} + D \qquad \forall v \in R \tag{5.20}$$

$$\delta_{\min} \in [1, B-D] \tag{5.21}$$

$$x^k_{uv} \geq 0 \qquad \forall (u_k, v_l) \in A_{\mathrm{L}} \tag{5.22}$$

$$y^l_v \geq 0 \qquad \forall v_l \in V_{\mathrm{L}} \setminus \{s\} \tag{5.23}$$

$$x_{uv} \in \{0,1\} \qquad \forall (u,v) \in A \tag{5.24}$$

Inequalities (5.14) and (5.15) state that from the set of layered graph nodes corresponding to one particular original node exactly one has to be chosen for required nodes and at most one for potential Steiner nodes, respectively. Indegree constraints (5.16) in G_{L} restrict the number of incoming arcs to a layered graph node v_l in dependency of y^l_v to at most one. Since G_{L} is acyclic inequalities (5.17) are enough to ensure connectivity. Equalities (5.18) and (5.19) link layered graph arcs to original arcs. Delay-variation-bound D is guaranteed by (5.20) and (5.21). LAY_{LP} denotes the LP relaxation of LAY.

In principle, variables x_{uv} and y^l_v are redundant since they can be substituted by Boolean layered graph arc variables x^l_{uv} using equalities (5.16), (5.18) and (5.19). However, model LAY is better readable by including them and branching on x_{uv} and Boolean y^l_v variables turned out to be more efficient in practice than branching on variables x^l_{uv}. In fact, branching on original arcs usually is more balanced since setting $x^l_{uv} = 1$ for one particular layered graph arc in general is a stronger constraint on the set of feasible solutions than setting $x_{uv} = 1$.

5.6.1 Valid Inequalities

The following sets of valid inequalities are not necessary for the feasibility of model LAY but are useful to strengthen it w.r.t. its LP relaxation. The different sets of directed connection cut inequalities and their separation discussed in Section 4.9.1 and 4.9.2, respectively, clearly are valid for the RDDVCST problem, too. Additionally, we now introduce some sets of valid inequalities especially considering the delay-variation-constraint.

Let $L_v = \{l \mid v_l \in V_{\mathrm{L}}\} \subseteq \{1, ..., B\}$ denote the set of achievable layers in G_{L} for a node $v \in V$. We know that a terminal node $u_k \in V_{\mathrm{L}},\ u \in R$, on layer $k \in L_u$ can only be in a feasible solution if no other terminal node $v_l \in V_{\mathrm{L}},\ v \in R$, on layer $l \in L_v$ outside the interval $[k-D, k+D]$ is included. This leads to inequalities

$$y^k_u + y^l_v \leq 1 \qquad \forall u, v \in R,\ \forall k \in L_u,\ \forall l \in L_v,\ |k-l| > D. \tag{5.25}$$

The number of inequalities (5.25) is in $\mathcal{O}(|R|^2 \cdot B^2)$. We can aggregate them to form stronger

constraints

$$y_u^k + \sum_{l \in L_v \setminus \{k-D,\dots,k+D\}} y_v^l \leq 1 \qquad \forall u, v \in R,\ \forall k \in L_u. \tag{5.26}$$

The number of these is in $\mathcal{O}(|R|^2 \cdot B)$. Now we relate arbitrary subsets of layers of two terminal nodes leading to a violation of the delay-variation-constraint:

$$\sum_{l \in L'_u} y_u^l + \sum_{l \in L'_v} y_v^l \leq 1 \qquad \forall u, v \in R,\ \forall L'_u \subseteq L_u,\ \forall L'_v \subseteq L_v \text{ with}$$
$$|l_u - l_v| > D,\ \forall l_u \in L'_u,\ \forall l_v \in L'_v \tag{5.27}$$

In the most general variant we consider infeasible combinations of arbitrary subsets of layers of an arbitrary subset of terminal nodes:

$$\sum_{v \in R'} \sum_{l \in L'_v} y_v^l \leq 1 \qquad \forall R' \subseteq R,\ \forall v \in R',\ \forall L'_v \subseteq L_v, \text{ with}$$
$$|l_u - l_v| > D,\ \forall u, v \in R',\ \forall l_u \in L'_u,\ \forall l_v \in L'_v \tag{5.28}$$

Note that due to the inequalities' conditions w.r.t. R', v, and L'_v, the sum on the left side can include at most B y-variables, but the number of constraints can be exponential. In Fig. 5.4a an example is given where constraints (5.26)–(5.28) tighten LAY_{LP}.

5.6.2 Separation Methods

To find violated inequalities (5.28) we consider an optimal LP solution S and build a support graph $G_S = (V_S, A_S)$ with node set

$$V_S = \{s\} \cup \{v_l \in V_{\mathrm{L}} \mid v \in R,\ y_v^l > 0\} \tag{5.29}$$

and arc set

$$\begin{aligned} A_S = &\{(s, v_l) \mid v_l \in V_S \setminus \{s\}\} \\ &\cup \{(v_k, v_l) \mid v_k, v_l \in V_S \setminus \{s\},\ k < l,\ \nexists v_i \in V_S : k < i < l\} \\ &\cup \{(u_k, v_l) \mid u_k, v_l \in V_S \setminus \{s\},\ u \neq v,\ k < l,\ l - k > D\}. \end{aligned} \tag{5.30}$$

Furthermore, we assign arc costs $c_a = y_v^l,\ \forall a = (u_k, v_l) \in A_S$.

Lemma 5.6.1. *Given an LP solution S and the corresponding graph G_S, each path $P \subseteq A_S$ with source node s and costs $c_P > 1$ corresponds to an inequality* (5.28) *I_S^P by solution S and vice versa.*

Proof. Assume a path P in G_S starting in s with costs $c_P > 1$ is given. By relating arc $a = (u_k, v_l)$ to variable y_v^l the sum of arc costs of P corresponds to a sum of y_v^l-variable values since $c_a = y_v^l$. Due to the definition of G_S, P can only consist of arcs $(u_k, v_l) \in A_S$ with $k < l$ and either $u = v$ or $u \neq v \wedge l - k > D$. Therefore the sum of variables y_v^l corresponding to a path P

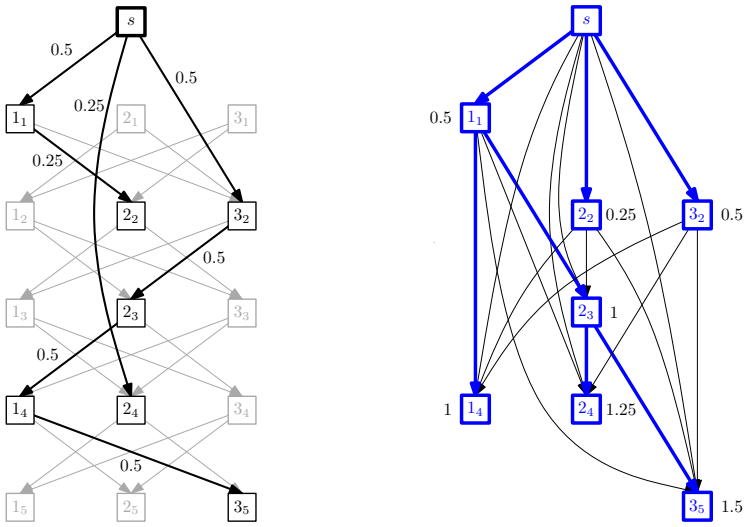


Figure 5.4: (a) The solution ($D = 1$, arc labels denote variable values of the LP solution, gray arcs mean $x^l_{uv} = 0$) is feasible for LAY_{LP} and for inequalities (5.25) but not for (5.26)–(5.28) since $y^1_1 + y^3_2 + y^4_2 = 1.25 > 1$ and $y^1_1 + y^3_2 + y^5_3 = 1.5 > 1$. (b) Graph G_S for the separation problem corresponding to solution (a): The blue arcs represent the longest path tree and the node values denote the maximal costs.

forms the left side of a valid inequality (5.28) and since $c_P > 1$ we obtain a violated inequality I^P_S for solution S.

Now, let I_S be a violated inequality (5.28) for solution S. First we remove all variables with $y^l_v = 0$ and sort the remaining sum of variables y^l_v by ascending layers l. Due to the constraint definition no two variables can have the same layer l and if we consider two consecutive variables y^k_u and y^l_v then either $u = v$ or $u \neq v \wedge l - k > D$. Furthermore, there has to be either an arc $(u_k, v_l) \in A_S$ or in case of $u = v$ possibly a path $P' = (u_k, ..., u_l)$ including other nodes u_i with $k < i < l$. So the series of variables in I_S can again be related to a path P in G_S starting in s and since the sum of variable values is larger than 1 the costs of path P are at least that high. □

Following Lemma 5.6.1 we now search for the longest paths from s to at most $|R|$ leaves in G_S. The single-source longest path problem can here be solved in linear time since G_S is a directed acyclic graph, cf. [5]. Obviously, all inequalities $I^{P'}_S$ corresponding to sub-paths $P' \subset P$ with $c_{P'} > 1$ are dominated by I^P_S. To further strengthen inequality I^P_S we try to feasibly add as many summands as possible, not only the node variables which are positive in solution S. Otherwise similar violated inequalities are possibly found in further iterations. So if we consider an arc $(v_k, v_l) \in A_S$ on a violating path P connecting two layered graph nodes corresponding to the same original node we additionally add all variables y^i_v, $\forall v_i \in V_{\text{L}}$, $k < i < l$, to I^P_S. Using this separation routine we are able to guarantee that the "most violated" inequalities are found

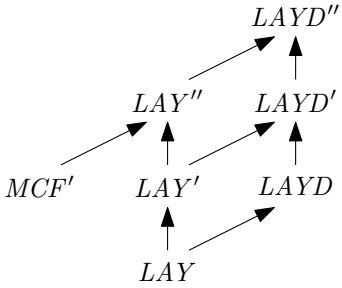


Figure 5.5: Polyhedral comparison of all described formulations. An arrow describes a direct relation between two formulations: the formulation at the tail of an arrow is weaker than the formulation at the head. If there is no directed path between two formulations, the corresponding polyhedra are incomparable.

hopefully resulting in a large increase of the optimal LP relaxation value.

Figure 5.4b shows graph G_S for the separation problem corresponding to solution 5.4a: Arc costs $c_a = y_v^l,\ \forall a = (u_k, v_l) \in A_S$, can be easily derived from Fig. 5.4a since equalities 5.16 directly link layered node and incoming arc variables, i.e. $y_1^1 = y_1^4 = y_2^3 = y_3^2 = y_3^5 = 0.5,\ y_2^2 = y_2^4 = 0.25$. The longest paths can be obtained by a straight-forward label-setting algorithm [5] examining each arc exactly once. Finally, the set of blue arcs represent the longest path tree and the node values denote the according maximal costs. Considering the cost values assigned to leaves 1_4, 2_4, and 3_5, there are two values greater than 1, and by backtracking the corresponding paths to root s we obtain the two violated inequalities $y_1^1 + y_2^3 + y_2^4 \leq 1$ and $y_1^1 + y_2^3 + y_3^5 \leq 1$.

5.7 Polyhedral Comparison

We did not consider a formulation based on MTZ inequalities for the RDDVCST problem similar to the one in Section 4.4 because first, these models usually have a weak performance for the RDCST problem and second, it is not obvious how to model the delay-variation-constraint since the δ_v variables only represent upper bounds to the exact node-delays in an integer solution. Similarly, the formulation based on infeasible path inequalities from Section 4.7 cannot be adapted in a straight-forward way to include the delay-variation-constraint since there is no notion of node-delay in this model. Finally, an adaptation of the path formulation utilized in a branch-and-price approach as described in Section 4.5 is currently work in progress. Unfortunately, the according pricing subproblem is in general much more difficult to solve than the one for the RDCST problem since we now may have negative costs on arcs in the delay-constrained cheapest path problem. Therefore, we cannot re-use the simple dynamic programming approach.

Now we theoretically compare the polyhedra associated to the LP relaxation of the following formulations for the RDDVCST problem and projected into the space of $\boldsymbol{x}$ variables:

- MCF': multi-commodity flow formulation in Section 5.4
- LAY: layered graph formulation in Section 5.6 with inequalities (4.30)–(4.31)

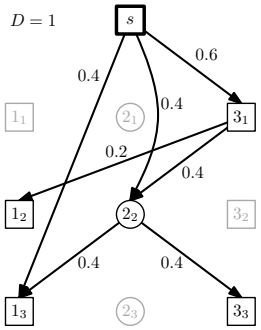


Figure 5.6: Example solution which is feasible LAY''_{LP} but not for $LAYD''_{\mathrm{LP}}$. Arc labels denote variable values of the LP solution and squared nodes denote terminal nodes.

- LAY': formulation LAY with directed connection cut inequalities (4.27) on graph G'
- LAY'': formulation LAY with directed connection cut inequalities (4.50) on layered graph G'_{L}
- $LAYD$: formulation LAY with delay-variation inequalities (5.28)
- $LAYD'$: formulation LAY' with delay-variation inequalities (5.28)
- $LAYD''$: formulation LAY'' with delay-variation inequalities (5.28)

We denote the polyhedra of the according LP relaxations projected into the space of $\boldsymbol{x}$ variables by $P_{MCF'}$, P_{LAY}, $P_{LAY'}$, $P_{LAY''}$, P_{LAYD}, $P_{LAYD'}$, and $P_{LAYD''}$, respectively. We write $P_1 \subset P_2$ if $P_1 \subseteq P_2$ and there exist instances such that $\exists \boldsymbol{x} \in P_2 : \boldsymbol{x} \notin P_1$. Figure 5.5 summarizes the relations between the formulations which are now discussed one by one. Most of these relations are implied by corresponding propositions from Section 4.10 for the RDCST problem.

Proposition 5.7.1. *$P_{LAY''} \subset P_{LAY'} \subset P_{LAY}$, and $P_{LAYD''} \subset P_{LAYD'} \subset P_{LAYD}$.*

Proof. These relations directly follow from Proposition 4.10.10, 4.10.9, and 4.10.3. □

Proposition 5.7.2. *$P_{LAYD''} \subset P_{LAY''}$, $P_{LAYD'} \subset P_{LAY'}$, and $P_{LAYD} \subset P_{LAY}$.*

Proof. Clearly, $P_{LAYD''} \subseteq P_{LAY''}$, $P_{LAYD'} \subseteq P_{LAY'}$, and $P_{LAYD} \subseteq P_{LAY}$ holds. Figure 5.6 shows an example with $D = 1$ where delay-variation inequalities (5.28) are able to strengthen LAY''_{LP}: $\sum_l l \cdot y_1^l = 2.8$, $\sum_l l \cdot y_3^l = 1.8 \Rightarrow \delta_{\min} = 1.8$, but $y_1^3 + y_3^1 = 1.4 > 1$. Obviously, this example can also be used to show the proper inclusion of the other relations. □

Proposition 5.7.3. *$P_{LAY''} \subset P_{MCF'}$.*

Proof. By applying the max-flow-min-cut theorem, we know that formulation LAY'' ensures a flow of one unit from root s to each terminal node $v \in R_{\mathrm{L}}$ in layered graph G'_{L}. Moreover, this also holds in graph G' since inequalities (4.50) include (4.27). Since the flow in G'_{L} can only

use delay-feasible paths the sum in inequalities (5.6) is always less or equal than delay-bound B. Furthermore, by setting $\sum_{(u,v)} d_{uv} f_{uv}^w = \sum_l l \cdot y_w^l,\ \forall w \in R$, we can easily see that for each feasible solution for LAY''_{LP} we can construct a feasible solution for MCF'_{LP}. However, we also know from Fig. 4.5c and Proposition 4.10.12 that MCF'_{LP} may include delay-infeasible paths in a fractional solution. □

Proposition 5.7.4. $P_{MCF'} \neq P_2,\ P_2 \in \{P_{LAY},\ P_{LAY'},\ P_{LAYD},\ P_{LAYD'}\}$.

Proof. This directly follows from Proposition 4.10.12 by using a non-restricting delay-variation-constraint in the example solutions, e.g. $D = B$. □

Proposition 5.7.5. $P_{LAY''} \neq P_{LAYD}$, $P_{LAY''} \neq P_{LAYD'}$, *and* $P_{LAY'} \neq P_{LAYD}$.

Proof. On the one hand, Fig. 5.6 shows a solution which is feasible for LAY'_{LP} and LAY''_{LP} but not for $LAYD_{\mathrm{LP}}$. On the other hand, by setting $D = B$, Fig. 4.3a is feasible for $LAYD_{\mathrm{LP}}$ but infeasible for LAY'_{LP}, and Fig. 4.3b is feasible for $LAYD'_{\mathrm{LP}}$ but not for LAY''_{LP}. □

5.8 Computational Results

In this section we compare all mentioned formulations on different sets of benchmark instances. In detail we discuss the following solution approaches:

- FL: branch-and-cut (BC) based on multi-commodity flow formulation MCF' in Section 5.4
- L1: BC based on layered graph formulation LAY in Section 5.6 with inequalities (4.30)–(4.31) added a priori to the model
- L2: L1 extended by directed connection cut inequalities (4.27) on graph G'
- L3: L1 extended by directed connection cut inequalities (4.50) on layered graph G'_{L}
- D1: L1 extended by delay-variation inequalities (5.28)
- D2: L2 extended by delay-variation inequalities (5.28)
- D3: L3 extended by delay-variation inequalities (5.28)

5.8.1 Test Instances and Environment

We apply all MIP approaches to the set of benchmark instances introduced by Gouveia et al. [68] and re-used for computational experiments for the RDCST problem, see Section 4.11.1 for further details. However, we focus here on the most difficult subset E with Euclidian costs and the root s placed near the border. Additionally, we also consider E sets containing smaller complete graphs with 21 nodes. Thus, e.g. E21-10 denotes the set of instances where $|V| = 21$ and $d_e \in \{1, ..., 10\},\ \forall e \in E$. These instances originally have been designed for the RDCMST

problem requiring all nodes in a solution. However, imposing an additional delay-variation-constraint possibly results in a star-like solution tree with low height since all nodes have to be connected with a similar path-delay. Therefore, we set $R = \{0, ..., \lfloor |V|/2 \rfloor\}$ to allow more "practicable" solutions (root $s = |V| - 1$ for all instances). Moreover, $D \in \{1, 3, 5\}$ for sets E21-10 and E41-10, and $D \in \{10, 30, 50\}$ for sets E21-100 and E41-100.

To reduce the input graphs we applied all preprocessing methods described in Section 5.3 prior to solving. No initial primal solution is provided since we currently have no working heuristics for the RDDVCST problem at hand and the methods proposed in Chapter 3 cannot be reliably used here since they are not able to ensure the delay-variation-constraint.

We used IBM ILOG CPLEX 12.3 to solve the MIP models. The layered node and arc variables in model LAY and the flow variables in MCF' are declared Boolean since the CPLEX presolver benefits from integrality of these variables and therefore can significantly reduce the model. Furthermore, because of too high time consumption we disabled the probing extension of CPLEX which checks the logical implications of setting each Boolean variable to 0 or 1. The dual simplex algorithm has been used for solving LPs in CPLEX, since it turned out to significantly outperform other options (primal simplex, barrier) in preliminary tests. Finally, a memory limit of 3 GB and a time limit of 10 000 CPU-seconds are set for each experiment. Apart from that, all other CPLEX settings remain at their default.

All tests are performed either on a single core of Intel Xeon E5540 processors with 2.53 GHz where eight cores share 24 GB of memory, or on a single core of Intel Xeon E5649 processors with 2.53 GHz where 12 cores share 48 GB of memory. In preliminary tests both systems yielded nearly the same performance within usual limits of tolerances.

5.8.2 LP Bounds

Table 5.1 shows results for several instance sets when only solving the LP relaxation of the considered formulations. Since optimal integer values are not available for all instances, we report average LP bounds relative to FL in percent, and the median runtime to reach these bounds. Dashes in time columns represent the time limit of 10 000 seconds.

According to the polyhedral comparison in Section 5.7, D3 should always obtain at least as good LP bounds as all other approaches. However, as indicated in the results this best dual bound cannot always be reached within the given time limit. In many cases huge numbers of added connection cuts on layered graph G'_{L} and delay-variation inequalities lead to large models and slow LP resolvings. However, the combination of these two sets of valid inequalities is often able to dramatically increase the dual bound compared to all other formulations.

In contrast to the layered graph approaches, the results clearly illustrate the independence of FL's runtime from the delay-bound. Thus, FL yields quite good LP bounds in moderate runtimes even for the most difficult subset E41-100 where the layered graph approaches obviously show weaknesses. Interestingly, this does not imply superiority of FL when embedded in a branch-and-cut, as shown in the next section.

In general, the obtained integrality gaps even if adding all valid inequalities, are much higher than those of the corresponding RDCST problem without the delay-variation-constraint, see Section 4.11.2. This documents that the delay-variation-constraint indeed imposes a big additional challenge.

			average LP bounds relative to FL in %							median time in seconds						
Set	B	D	FL	L1	L2	L3	D1	D2	D3	FL	L1	L2	L3	D1	D2	D3
E21-10	10	1	0.0	14.5	19.9	22.5	20.5	24.5	**28.3**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
		3	0.0	2.1	5.8	9.0	7.7	11.5	**16.0**	**0**	**0**	**0**	**0**	**0**	**0**	1
		5	0.0	1.7	4.5	7.0	5.1	7.9	**12.6**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	15	1	0.0	7.1	12.4	15.3	15.4	20.5	**27.0**	**0**	**0**	**0**	**0**	1	1	4
		3	0.0	-0.8	3.6	7.2	5.1	10.3	**14.7**	1	**0**	**0**	2	1	1	11
		5	0.0	-2.4	1.8	5.1	0.2	5.2	**10.5**	1	**0**	1	2	1	1	5
	20	1	0.0	1.7	8.0	12.3	9.0	15.6	**23.1**	**0**	1	1	3	4	11	82
		3	0.0	-2.7	3.6	6.6	3.3	9.6	**14.4**	**1**	2	2	6	3	6	88
		5	0.0	-3.6	2.4	5.6	-0.6	5.6	**10.3**	**1**	**1**	2	10	2	4	76
	25	1	0.0	-2.7	5.6	9.2	5.8	13.6	**20.1**	**1**	**1**	2	7	24	44	655
		3	0.0	-4.2	3.0	5.5	1.6	9.3	**13.8**	**1**	3	4	19	7	18	702
		5	0.0	-5.1	2.2	4.5	-1.9	5.6	**9.8**	**1**	**1**	2	49	3	8	347
E41-10	10	1	0.0	7.4	10.9	13.8	12.5	15.4	**20.0**	14	**0**	1	1	1	2	10
		3	0.0	-0.4	4.0	8.5	4.2	6.9	**13.4**	32	**1**	**1**	6	**1**	3	14
		5	0.0	-0.7	3.9	9.2	2.4	5.2	**12.1**	46	**1**	**1**	6	**1**	**1**	6
	15	1	0.0	1.8	7.0	11.8	6.4	11.2	**18.1**	54	**1**	3	25	16	52	560
		3	0.0	-4.8	0.7	6.0	-2.1	3.2	**10.5**	56	**2**	7	85	6	27	827
		5	0.0	-5.3	0.4	6.4	-3.3	2.3	**10.5**	59	**2**	7	97	5	19	463
	20	1	0.0	-2.5	4.0	8.8	1.2	7.3	**13.7**	37	**4**	11	306	118	430	5349
		3	0.0	-5.9	1.0	6.1	-4.1	3.1	**9.6**	62	**8**	24	3070	29	112	7957
		5	0.0	-6.2	0.2	5.9	-4.8	2.0	**9.5**	48	**9**	22	1578	19	72	5819
	25	1	0.0	-4.7	2.7	7.3	-1.5	5.6	**10.6**	58	**7**	29	3024	405	1693	-
		3	0.0	-6.6	1.3	5.8	-4.8	3.5	**8.4**	58	**13**	54	-	93	307	-
		5	0.0	-6.6	1.0	6.1	-5.3	2.9	**8.7**	54	**18**	62	5329	55	183	-
E21-100	100	10	0.0	9.5	13.5	15.5	22.2	26.1	**32.8**	**0**	**0**	1	1	2	3	11
		30	0.0	0.0	9.7	11.0	8.7	11.7	**13.8**	**0**	2	3	10	4	7	12
		50	0.0	0.9	7.4	11.2	8.0	10.3	**15.4**	**0**	2	3	10	2	3	12
	150	10	0.0	1.6	7.5	11.3	12.4	18.3	**26.8**	**1**	2	6	43	33	105	2844
		30	0.0	-3.8	5.1	7.2	2.9	8.5	**12.7**	**1**	16	30	297	30	101	1327
		50	0.0	-2.5	3.4	5.9	1.9	6.5	**11.5**	**0**	9	21	131	19	44	785
	200	10	0.0	-4.1	3.8	6.6	5.3	13.0	**19.3**	**1**	10	25	580	276	1116	-
		30	0.0	-6.4	2.4	4.8	-1.5	6.3	**9.7**	**1**	23	82	3390	279	1139	-
		50	0.0	-4.6	1.8	4.1	-0.9	5.1	**9.5**	**1**	23	58	1640	110	500	-
	250	10	0.0	-6.3	1.7	4.6	2.0	10.5	**12.5**	**0**	18	49	5251	1276	4553	-
		30	0.0	-7.7	1.3	3.3	-3.0	5.3	**6.5**	**1**	40	155	-	504	2494	-
		50	0.0	-6.0	1.2	2.5	-2.5	4.3	**7.1**	**1**	46	155	3253	245	588	-
E41-100	100	10	0.0	6.0	9.8	13.9	12.8	16.0	**21.5**	35	**14**	30	628	182	412	-
		30	0.0	-2.3	3.3	7.9	0.8	5.4	**10.9**	68	**47**	118	4496	107	524	-
		50	0.0	-2.9	2.4	6.8	-1.4	3.2	**9.5**	86	**37**	84	2439	57	170	-
	150	10	0.0	-1.4	4.5	7.5	3.6	**8.2**	**8.2**	**60**	77	305	-	2864	-	-
		30	0.0	-5.4	1.1	**3.9**	-3.0	1.5	1.9	**70**	233	721	-	4269	-	-
		50	0.0	-5.4	0.5	**3.2**	-4.2	-0.5	0.7	**56**	126	1472	-	764	-	-
	200	10	0.0	-4.6	**2.8**	2.6	-1.1	-1.2	-1.2	**52**	223	757	-	-	-	-
		30	0.0	-6.3	**0.1**	-0.7	-4.8	-4.9	-4.8	**75**	601	4582	-	-	-	-
		50	0.0	-6.2	**0.3**	-0.6	-5.2	-5.2	-4.8	**88**	352	-	-	-	-	-
	250	10	0.0	-6.2	**2.0**	-0.7	-4.0	-3.9	-3.8	**49**	405	2060	-	-	-	-
		30	**0.0**	-25.1	-21.0	-22.2	-24.3	-24.3	-24.3	**65**	933	-	-	-	-	-
		50	**0.0**	-6.5	-3.2	-4.4	-5.9	-5.9	-5.9	**77**	2297	-	-	-	-	-

Table 5.1: LP results for instances by Gouveia et al. [68] (B: delay-bound, D: delay-variation-bound, FL: flow approach, L1: layered graph approach, L2: L1 with connection cuts on G', L3: L1 with connection cuts on G'_{L}, D1-3: L1-3 with delay-variation cuts, best results are printed bold).

5.8.3 Branch-and-Cut Results

Test results embedding our model variants in a branch-and-cut approach are shown in Tables 5.2 and 5.3 where dashes denote either a 100% gap or reached time limit.

It can clearly be seen that while the layered graph approaches performed at least reasonably well, FL is not competitive in most cases. Only for the small E21-100 instances FL can sometimes outperform the other methods since here the number of nodes is quite low resulting in a manageable number of flow variables and the delay-bounds are rather high which is disadvantageous for the layered graph approaches. However, for most E41 instances FL could not even find a feasible solution indicated by the 100% gaps. Although the LP relaxation bounds of the flow model can be computed quite fast, see Table 5.1, they are much too weak often leading to a huge branch-and-bound tree.

It is interesting to see that L1 mostly obtains worse LP bounds than FL and often needs even more time to compute them, but when considering the corresponding branch-and-cut approaches, L1 outperforms FL in most cases. Here, the CPLEX presolver and integrated general-purpose cuts greatly benefit from the structure of the layered graph model and thus can both significantly reduce the formulation size in the beginning and additionally strengthen the LP relaxations throughout the solving process.

Approaches L1, L2, D1, and D2, performed best without being able to elect one clear winner and could at least solve about half of the instances to optimality. Considering the subsets with small delay-bounds in Table 5.2, we can clearly see that it makes sense to separate delay-variation inequalities. However, the picture is not that clear anymore for instances in Table 5.3: Here, we observed for D1 and D2 a high number of added cuts due to a larger graph G_{L} and thus more possibilities to combine nodes in V_{L} to form violated delay-variation inequalities. Analogously to infeasible path inequalities for the RDCST problem in Section 4.7, many similar violated inequalities are found differing only in a few terms. To partly prevent these situations we aim to find stronger inequalities relating more than one node combinations within one inequality.

Obviously, directed connection cuts are only rarely in graph G' and not at all in G'_{L} helpful to improve computation times. Reasons for this are both the higher complexity of the separation problem compared to the fast method for finding violated inequalities (5.28) and the fact that in most cases the number of added connection cuts is rather high leading to slow LP relaxation resolvings.

We can see for both the flow and layered graph approaches that in most cases it is easier to solve instances with higher delay-variation-bound which is quite intuitive since it is usually hard for CPLEX heuristics to find good primal bounds if D is rather strict. Note that this obvious correlation does not hold for delay-bounds when considering layered graph approaches: Here, stricter delay-bounds result in smaller layered graphs and thus smaller search spaces. We are able to implicitly encode the delay-bound in the layered structure which is not possible for the delay-variation-bound. To satisfy D we need additional inequalities (5.20) which are not directly reflected in the structure of layered graph G_{L} and therefore make it usually harder to solve the LP relaxation and to find useful feasible solutions.

			# optimal solutions (out of 5)							average gap in %							median time in seconds						
Set	B	D	FL	L1	L2	L3	D1	D2	D3	FL	L1	L2	L3	D1	D2	D3	FL	L1	L2	L3	D1	D2	D3
E21-10	10	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	2599	**1**	**1**	**1**	**1**	**1**	**1**
		3	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	1882	**2**	**2**	4	**2**	**2**	**2**
		5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	150	1	1	**0**	1	1	1
	15	1	0	**5**	**5**	**5**	**5**	**5**	**5**	23.4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**6**	13	126	13	11	73
		3	1	**5**	**5**	**5**	**5**	**5**	**5**	14.8	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	18	25	346	**16**	21	117
		5	2	**5**	**5**	**5**	**5**	**5**	**5**	5.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**12**	18	48	15	17	45
	20	1	0	**5**	**5**	2	**5**	**5**	3	31.2	**0.0**	**0.0**	7.3	**0.0**	**0.0**	4.1	-	**55**	85	-	210	181	1077
		3	0	**5**	**5**	1	**5**	**5**	4	18.9	**0.0**	**0.0**	14.8	**0.0**	**0.0**	2.3	-	284	870	-	**220**	243	7381
		5	1	**5**	**5**	3	**5**	**5**	4	14.0	**0.0**	**0.0**	3.4	**0.0**	**0.0**	0.1	-	215	188	8596	127	**117**	1404
	25	1	0	**5**	**5**	0	4	3	1	27.9	**0.0**	**0.0**	19.1	1.5	4.8	14.2	-	**347**	434	-	4592	2330	-
		3	0	**5**	**5**	1	**5**	**5**	1	17.2	**0.0**	**0.0**	46.2	**0.0**	**0.0**	12.3	-	**787**	1730	-	2934	3115	-
		5	0	**5**	**5**	1	**5**	**5**	2	17.2	**0.0**	**0.0**	21.2	**0.0**	**0.0**	6.7	-	772	1981	-	824	**669**	-
E41-10	10	1	0	**5**	**5**	**5**	**5**	**5**	**5**	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**26**	43	414	**26**	33	79
		3	0	**5**	**5**	4	**5**	**5**	**5**	62.1	**0.0**	**0.0**	0.5	**0.0**	**0.0**	**0.0**	-	116	191	779	**43**	63	223
		5	0	**5**	**5**	**5**	**5**	**5**	**5**	53.2	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**17**	34	159	18	24	**17**
	15	1	0	**4**	0	0	1	0	0	-	**2.7**	15.6	24.3	8.7	10.3	24.4	-	**3631**	-	-	-	-	-
		3	0	2	0	0	**4**	2	0	-	8.5	15.1	30.2	**0.8**	4.3	17.2	-	-	-	-	**6707**	-	-
		5	0	4	3	0	**5**	**5**	3	-	3.0	7.8	22.7	**0.0**	**0.0**	5.3	-	1855	8922	-	**1346**	1770	8219
	20	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	27.5	**26.1**	88.2	34.9	32.7	56.0	-	-	-	-	-	-	-
		3	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	28.3	27.6	71.2	22.8	**18.8**	51.4	-	-	-	-	-	-	-
		5	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	21.5	21.7	52.4	20.0	**17.5**	43.3	-	-	-	-	-	-	-
	25	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	35.3	**35.2**	-	52.5	68.4	-	-	-	-	-	-	-	-
		3	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	35.3	33.9	-	27.6	**26.8**	71.1	-	-	-	-	-	-	-
		5	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	33.0	28.4	75.1	28.1	**23.3**	67.7	-	-	-	-	-	-	-

Table 5.2: Branch-and-cut results for instances by Gouveia et al. [68] (B: delay-bound, D: delay-variation-bound, FL: flow approach, L1: layered graph approach, L2: L1 with connection cuts on G', L3: L1 with connection cuts on G'_{L}, D1-3: L1-3 with delay-variation cuts, best results are printed bold).

			# optimal solutions (out of 5)							average gap in %							median time in seconds						
Set	B	D	FL	L1	L2	L3	D1	D2	D3	FL	L1	L2	L3	D1	D2	D3	FL	L1	L2	L3	D1	D2	D3
E21-100	100	10	0	**5**	**5**	4	**5**	**5**	**5**	19.8	**0.0**	**0.0**	0.8	**0.0**	**0.0**	**0.0**	-	**42**	76	1489	46	49	168
		30	4	**5**	**5**	**5**	**5**	**5**	**5**	1.7	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	2733	89	193	349	79	**36**	65
		50	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	1015	27	**22**	63	23	24	35
	150	10	0	**4**	2	0	1	0	0	45.5	**2.9**	10.0	53.9	23.6	20.4	67.9	-	**2152**	-	-	-	-	-
		30	1	**5**	4	1	4	**5**	1	13.9	**0.0**	0.6	27.1	1.6	**0.0**	8.6	-	3405	**2034**	-	8009	7673	-
		50	4	**5**	**5**	4	**5**	**5**	**5**	5.3	**0.0**	**0.0**	5.2	**0.0**	**0.0**	**0.0**	1250	1475	633	5220	**491**	577	3309
	200	10	0	**1**	0	0	0	0	0	37.3	**22.5**	33.5	91.0	59.9	76.4	-	-	-	-	-	-	-	-
		30	**2**	1	1	0	1	1	1	**8.4**	17.2	15.8	46.0	17.5	17.8	63.2	-	-	-	-	-	-	-
		50	**4**	1	**4**	0	1	2	1	5.4	10.5	**1.9**	26.5	13.5	5.7	60.7	**2353**	-	7225	-	-	-	-
	250	10	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**48.0**	62.6	66.0	-	-	-	-	-	-	-	-	-	-	-
		30	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**22.0**	26.0	26.0	80.0	37.6	37.3	80.0	-	-	-	-	-	-	-
		50	**3**	0	1	0	1	1	0	**9.8**	22.5	11.7	52.1	13.8	14.5	81.3	**5830**	-	-	-	-	-	-
E41-100	100	10	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	48.2	**42.4**	89.6	63.6	66.1	-	-	-	-	-	-	-	-
		30	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	32.9	32.4	70.0	30.5	**25.7**	90.3	-	-	-	-	-	-	-
		50	0	0	0	0	0	**1**	0	86.7	14.5	**13.2**	44.5	15.6	**13.2**	54.7	-	-	-	-	-	-	-
	150	10	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	92.0	**80.0**	-	-	-	-	-	-	-	-	-	-	-
		30	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	**42.7**	48.4	-	88.0	86.6	-	-	-	-	-	-	-	-
		50	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	38.1	31.0	-	50.2	**27.6**	-	-	-	-	-	-	-	-
	200	10	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		30	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	**87.8**	88.2	-	-	-	-	-	-	-	-	-	-	-
		50	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	56.1	56.2	-	**49.5**	-	-	-	-	-	-	-	-	-
	250	10	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		30	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	**89.1**	-	-	-	-	-	-	-	-	-	-	-	-
		50	**0**	**0**	**0**	**0**	**0**	**0**	**0**	-	**57.9**	86.6	-	75.6	-	-	-	-	-	-	-	-	-

Table 5.3: Branch-and-cut results for instances by Gouveia et al. [68] (B: delay-bound, D: delay-variation-bound, FL: flow approach, L1: layered graph approach, L2: L1 with connection cuts on G', L3: L1 with connection cuts on G'_{L}, D1-3: L1-3 with delay-variation cuts, best results are printed bold).

5.9 Future Work

As clearly shown in the experimental results, the generally still relatively large integrality gaps of all models ask for investigating also other modeling approaches, e.g. path models. However, as already mentioned in Section 5.7, the according pricing subproblem definitely is a hard challenge. Moreover, an adaptation of the quite successful formulation based on infeasible path inequalities for the RDCST problem from Section 4.7 would be interesting, here. But then we need some additional sets of valid inequalities to ensure the delay-variation-constraint which do not seem to be obvious. For addressing the poor scalability of the layered graph models w.r.t. larger delay-bounds, an appropriate extension of the approach presented in Chapter 6 seems to be highly promising. However, we will also see that this is not as straight-forward as e.g. for the RDCST problem. The proof of Proposition 5.7.3 shows us a way to link flow variables of MCF' to layered node variables of LAY. Thus, in principle we can use delay-variation inequalities (5.28) to strengthen formulation MCF'. Whether such an improved multi-commodity flow model is interesting in a practical sense will be subject to future experiments.

To continue the discussion in the last paragraph of Section 5.8.3, our aim is to find a modeling scheme for constructing a layered graph where both the delay- and delay-variation-constraint are implicitly ensured without need to consider them explicitly in a corresponding MIP model. Having the tight LP bounds of the layered graph formulations for the RDCST problem from Section 4.11.2 in mind, we believe that we could close a significant part of the current integrality gaps by using such a modeling approach.

Further computational gains could be obtained by finding additional graph reduction methods on both the original and corresponding layered graph, and fast construction and LP heuristics to provide tighter primal bounds. Most heuristic approaches in literature rely on k-shortest-path algorithms which are computationally expensive. Additionally, not much local search methods exist which have been applied successfully in numerous other problems and can be easily integrated in other metaheuristic and exact methods to improve incumbent solutions. Two construction ideas came to our mind, the first one working on graph G and the second one on layered graph G_{L}.

Inspired by the re-balancing of AVL trees [31], a construction heuristic could work similar to the following procedure:

1. Start with an arbitrary tree solution containing all terminal nodes and satisfying delay-bound B, e.g. obtained by exact or heuristic methods for the RDCST problem, see Chapter 4.

2. If the tree satisfies the delay-variation-constraint then stop, else continue.

3. Find two terminals $u, v \in R$ with maximal delay-variation and follow the common path from root s to the unique branching node w where the two paths split up.

4. Examine path $P(u, v)$ and find the node w' on the path which divides the whole path delay in two parts with minimum delay difference.

5. Exchange path $P(s, w)$ by a new path $P(s, w')$ which could be obtained in different ways, e.g. by computing shortest-delay-paths or delay-constrained cheapest paths.

6. Optionally repeat this "re-balancing" for sub-branches on path $P(u, v)$.

7. Go to 2.

Clearly, an implementation of this algorithm may comprise sophisticated treatment of special cases. Additionally, this procedure could be used in a modified variant within a local search.

Our second construction concept works on layered graph G_{L} and is defined as follows:

1. Search for D consecutive layers $\{l, ..., l + D - 1\}$ including at least one layered node for each terminal node $v \in R$; there are $B - D + 1$ such intervals and checking whether one interval includes all required nodes can be done similarly to the sweep algorithm from Section 5.3.

2. Remove all nodes $v_k \in V_{\mathrm{L}}$ on layers $k > l+D-1$ and all terminal nodes $u_k \in V_{\mathrm{L}},\ u \in R$, on layers $k < l$.

3. Try to find a feasible solution in this heavily reduced layered graph by some kind of simple heuristic or solving a corresponding MIP. Note that the delay-variation-constraint does not have to be considered here anymore. If no solution can be found, go to 1.

The existence of D consecutive layers containing all terminal nodes is only a necessary but not sufficient condition for feasibility. First, we have to figure out if after reducing the layered graph all terminal nodes can still be reached from the root node. However, even if all terminals are reachable connectivity may be only possible by using two layered graph nodes corresponding to the same original node which again is not feasible. To conclude, deciding whether there is a feasible solution for one particular set of D consecutive layers is in general $\mathcal{NP}$-complete. If we consider all feasible sets of layers and solve the according subproblem by an exact method, e.g. a MIP, then we even obtain an exact algorithm for solving the RDDVCST problem. However, we believe that this is not competitive from a computational point of view.

Finally, we could consider the problem variant minimizing the delay-variation without considering edge costs at all. However, in preliminary tests this problem turned out to be even harder to solve than the RDDVCST problem. Additionally, the delay-variation inequalities (5.28) are not valid for this variant since the delay-variation-constraint D is implied in the definition of these inequalities and thus has to known a priori. Clearly, we may assume a large value for D at the beginning resulting in rather weak inequalities, and decrease D when new incumbent solutions are found.

CHAPTER 6

Adaptive Layers Framework

This chapter discusses a quite general approach to solve network design and routing problems with constraints limiting consumed resources on paths and routes, respectively. Section 6.1 motivates our so-called *Adaptive Layers Framework (ALF)* and Section 6.2 summarizes similar approaches. Section 6.3 describes the theoretical foundation on which our method is based. The details of two variants of our framework are discussed in Section 6.4 followed by computational results in Section 6.5 on the RDCMST and RDCST problems from Chapter 3 and 4, respectively. Section 6.6 examines the application of ALF to a variant of the RDCST problem with additional consideration of a quota constraint. A case study of using ALF to solve the vehicle routing problem with time windows is presented in Section 6.7. Finally, Section 6.8 discusses open problems and possible future research directions. Some parts are based on the published article [163].

6.1 Motivation

The discussion of the RDCST problem in Chapter 4 shows that MIP models on layered graphs usually provide tight LP relaxation values. A connection cut formulation on a layered graph yields the best LP bounds known so far and in many cases LP solutions are obtained which are already integral with no need for additional branching. However, the crucial disadvantage of these layered graph approaches is the strong dependence on given resource bounds. In case of a large set of achievable path delay values together with a high delay-bound the corresponding layered graph consists of a huge number of layers, nodes and arcs. Thus, a strong MIP formulation on this extended graph often is too large to be tractable in one piece in practice exceeding both time and space limits in many cases. Classical column and row generation methods, e.g. Dantzig-Wolfe [28] and Benders [54] decomposition, may help to deal with the size but only if the MIP structure is suitable for these rather general approaches.

We propose a different method which partly overcomes the problems with possibly large-sized layered graphs but still benefits from the tight dual bounds obtained by the corresponding layered graph models. Our framework approximates the LP relaxation and the optimal integer

value of a complete layered graph formulation by solving a sequence of usually much smaller models. More detailed, ALF starts with a strongly reduced node set in the layered graph redirecting arcs in a way to provide lower and upper bounds to the LP relaxation and the optimal integer value of the complete formulation. According to the obtained solutions the layered graph is iteratively extended, decreasing the gap.

In a second variant of ALF we only approximate the LP relaxation of a formulation on the full layered graph in a first phase. Additionally, during this approximation procedure the framework can optionally provide a sequence of improving primal feasible solutions. After certain stopping criteria are met the final formulation extended by valid inequalities necessary for guaranteeing feasibility is solved in the usual branch-and-cut way supported by the best obtained primal bound.

In principle, when considering the RDCST problem ALF can be seen as a flexible method to discretize delay values which does not fix a priori chosen discretized values but iteratively adapts and refines them to obtain a better approximation of the "real" delay values. Basically, also the layered graph model from Section 4.9 can be interpreted as a discretized model if considering the RDCST problem with real-valued edge delays: If we round all delay values to the next lower integer then the optimal solution of the corresponding layered graph model is a lower bound to the optimal solution of the original problem. On the other hand, if we round up all delays we obtain an upper bound.

6.2 Related Work

There are some existing articles discussing discretization methods for routing and scheduling problems with time constraints: Wang et al. [185, 186] tackle the (multiple) traveling salesman problem with time windows (TSPTW) and suggest a scheme to partition the time windows into equal sub-windows. Similarly to the layered graph transformation in Section 4.8, each sub-window induces a replication of the according original node. These sub-windows are then used to linearize a non-linear MIP formulation for the TSPTW problem. The smaller the sub-windows the larger the corresponding MIP model and the better the approximation of the original model. By considering either the start or the end of a sub-window when connecting two nodes the authors are able to provide either lower or upper bounds to the optimal solution of the original problem. In an iterative way they decrease the width of the sub-windows to monotonically tighten the bounds. Already obtained solutions are utilized in the sense that the time window is additionally split up at the optimal visiting time in the solution of the last iteration. Apart from that, time windows are partitioned in a rather homogeneous and unadapted way without taking previous solutions into account. The repeated solving of MIP models causes a quite high computational overhead.

Dash et al. [37] proposed in 2010 a more sophisticated discretization approach for the TSPTW. They divide each time window into so-called *buckets* not necessarily with equal width. An according time-indexed MIP formulation uses one variable for each bucket and additional valid inequalities to guarantee feasibility: Directed connection cut inequalities (4.27) have to be added to eliminate subtours and infeasible path inequalities similarly to (4.28) are necessary to avoid tours violating the original time windows. In a first phase the buckets are iteratively

refined to improve the dual bound obtained by the LP relaxation of the time bucket formulation. By examining the current optimal LP solution the buckets are further partitioned in a sophisticated way respecting the possibly fractional bucket and arc variables. If the buckets cannot be refined anymore by this procedure the formulation is solved in the usual branch-and-cut way extended by some sets of strong problem-specific variants of the classical subtour elimination and infeasible path inequalities additionally exploiting the bucket structure. However, the in general quite promising results indicate that stronger primal bounds are needed to accelerate the branch-and-bound process especially for the harder instances.

To the best of our knowledge there are no further approaches for network design problems which are at least partly related to our ALF.

6.3 Basics

This section discusses the theoretical foundation of our framework illustrated on the RDCST problem from Chapter 4. However, ALF is not limited to the RDCST problem and the corresponding basics can easily be adapted to related problems, e.g. see Sections 6.6 and 6.7. We use the same notation as in Chapter 4.

We are given a graph $G = (V, E)$ and the corresponding layered graph $G_{\mathrm{L}} = (V_{\mathrm{L}}, A_{\mathrm{L}})$ after applying the transformation and preprocessing described in Section 4.8. In G_{L} we now consider an arc $(u_j, v_l) \in A_{\mathrm{L}},\ d_v^{\min} < l < d_v^{\max}$, and redirect its target to a node $v_k \in V_{\mathrm{L}}$ on a lower layer $k < l$. Since $l > d_v^{\min}$ there always exists a feasible node v_k with $k < l$. We denote the resulting graph $\check{G}_{\mathrm{L}}$.

Lemma 6.3.1. *Let T_{L}^* and $\check{T}_{\mathrm{L}}^*$ be optimal solutions to the transformed RDCST problem on G_{L} and $\check{G}_{\mathrm{L}}$, respectively. Furthermore, we denote by T^* and $\check{T}^*$ the corresponding reverse transformed solutions in G, respectively. Then $c_{\check{T}^*} \leq c_{T^*}$.*

Proof. Redirecting an arc in G_{L} from target v_l to node v_k on a lower layer $k < l$ corresponds to a decrease of the related edge delay in G. Therefore, the solution space may be extended since it may now be easier to satisfy the delay-bound B. The optimal solution T^* still stays feasible but one or more of the new solutions may have less cost than T^*, so $c_{\check{T}^*} \leq c_{T^*}$. □

Figure 6.1b shows layered graph $\check{G}_{\mathrm{L}}$ derived from example graph G in Fig. 6.1a. One arc is redirected to a new target on a lower layer. The optimal solution $\check{T}_{\mathrm{L}}^*$ in $\check{G}_{\mathrm{L}}$ has less cost than the optimal solution T_{L}^* in G_{L}. Thus, $\check{T}^*$ cannot be feasible for the RDCST problem on G otherwise T^* would not be optimal. On the other hand, if $\check{T}^*$ would be feasible Lemma 6.3.2 applies.

Lemma 6.3.2. *If $\check{T}^*$ is feasible for the RDCST problem on G then it is optimal.*

Proof. According to Lemma 6.3.1, $c_{\check{T}^*} \leq c_{T^*}$. If $\check{T}^*$ is feasible for the RDCST problem on G then $c_{\check{T}^*} = c_{T^*}$. Therefore, $\check{T}^*$ is an optimal solution to the original problem. □

Proposition 6.3.3. *The transformed RDCST problem on redirected layered graph $\check{G}_{\mathrm{L}}$ is a relaxation to the RDCST problem on G.*

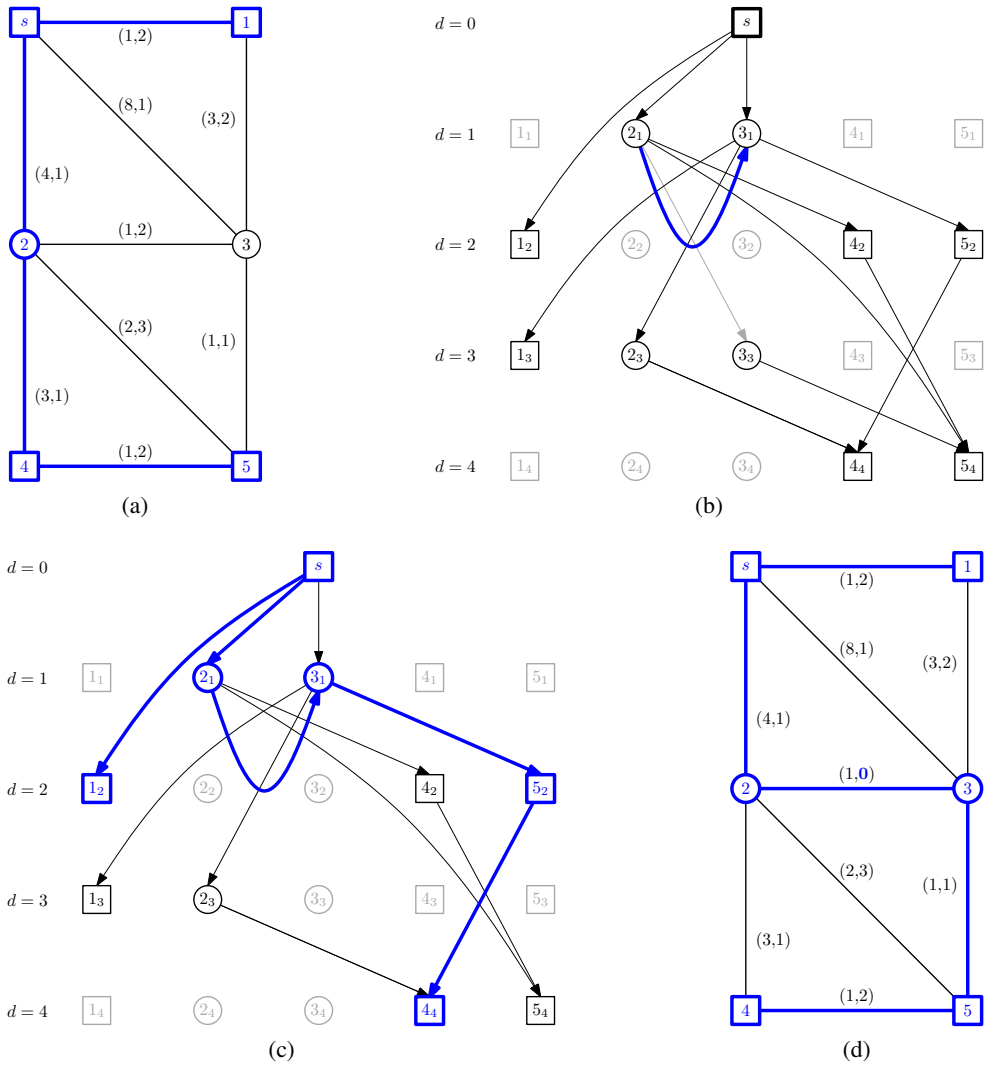


Figure 6.1: (a) Example graph G with edge labels (c_e, d_e) and $B = 4$. Squared nodes denote terminal nodes and blue arcs denote the optimal solution T^* with $c_{T^*} = 9$. (b) Layered graph $\check{G}_{\mathrm{L}}$ derived from G_{L} by redirecting arc $(2_1, 3_3)$ to node 3_1. (c) Optimal solution $\check{T}^*_{\mathrm{L}}$ in $\check{G}_{\mathrm{L}}$. (d) Reverse transformed infeasible solution $\check{T}^*$ in G with $c_{\check{T}^*} = 8$. Edge delay d_{23} is decreased to 0.

If we redirect an arc to a target node on a higher layer instead of a lower one and denote the resulting graph $\hat{G}_{\mathrm{L}}$, Lemma 6.3.4 holds.

Lemma 6.3.4. *Let $\hat{T}_{\mathrm{L}}$ be any feasible solution in $\hat{G}_{\mathrm{L}}$ and $\hat{T}$ the corresponding reverse transformed tree in G. Then $\hat{T}$ is feasible for the RDCST problem on G. Furthermore, $c_{\hat{T}^*} \geq c_{T^*}$.*

Proof. Redirecting an arc in G_{L} to a new target node on a higher layer corresponds to increasing the respective edge delay in G. Therefore, it may be harder to satisfy the delay-bound and the solution space may be pruned. Nevertheless, all feasible solutions $\hat{T}$ stay feasible for the original RDCST problem on G since replacing the modified edge delay by its original smaller one cannot violate the delay-bound. However, former optimal solutions may now be infeasible in $\hat{G}_{\mathrm{L}}$, so $c_{\hat{T}^*} \geq c_{T^*}$. □

Figure 6.2 shows an example for this case. By redirecting arc $(2_1, 4_2)$ to a higher layer the former optimal solution T^*_{L} now is not valid anymore in $\hat{G}_{\mathrm{L}}$. Here, the new optimal solution $\hat{T}^*$ in G has higher cost and therefore provides an upper bound to c_{T^*}.

Note that Lemma 6.3.1, 6.3.2, and 6.3.4, and also Proposition 6.3.3 do not hold for the RDDVCST problem from Chapter 5 since modifying an edge delay in graph G may result in delay-variation changes between terminal nodes, sometimes even leading to infeasibility of an instance. Thus, we are not able to apply ALF to the RDDVCST problem in a straight-forward way, see the discussion in Section 6.8 for further comments on this situation. Generally speaking, the results presented in this section are only valid for problems which define upper bound constraints on resources consumed on paths or tours. They do not hold for problems which relate the resources of different paths in a solution and set limits on these relations, similar to the delay-variation-constraint in the RDDVCST problem.

6.4 Framework

To reduce the size of a layered graph G_{L} we consider a node $v_l \in V_{\mathrm{L}}$, $d_v^{\min} < l < d_v^{\max}$, and redirect all incoming arcs $(u_j, v_l) \in A_{\mathrm{L}}$ to a node v_k on a different layer $k \neq l$. Then we can safely remove v_l together with all outgoing arcs from G_{L} since it cannot be reached from root s anymore and therefore cannot be part of a solution. If we want to obtain a lower bound the layer k of the new target node v_k is set to $\max_{d_v^{\min} \leq i < l}\{i : v_i \in V_{\mathrm{L}}\}$, for an upper bound $k = \min_{l < i \leq d_v^{\max}}\{i : v_i \in V_{\mathrm{L}}\}$. In other words, we want to minimize the difference between the original and the redirected target layer. Repeating this redirection process for further layered graph nodes results in a sequence of layered graphs with monotonically decreasing size. In contrast to this reduction process ALF goes the other way and starts with the smallest possible layered graph providing a feasible solution and extends it iteratively to tighten the bounds. Node set V^0_{L} of the initial layered graph G^0_{L} just contains root node s and nodes $v_{d_v^{\min}}$ and $v_{d_v^{\max}}$ for all $v \in V \setminus \{s\}$ on the lowest and highest feasible layer, respectively. If necessary, arcs are redirected to the next available layers depending on the desired bound.

The next step is to compute optimal LP and integer solutions $T^i_{\mathrm{L,LP}}$ and T^i_{L} of an appropriate model, e.g. model LAY from Section 4.9, on the current layered graph G^i_{L}. For all redirected arcs $(u_k, v_l) \in A^i_{\mathrm{L}}$ with $x^k_{uv} > 0$ in $T^i_{\mathrm{L,LP}}$ or T^i_{L} we extend G^i_{L} by adding the

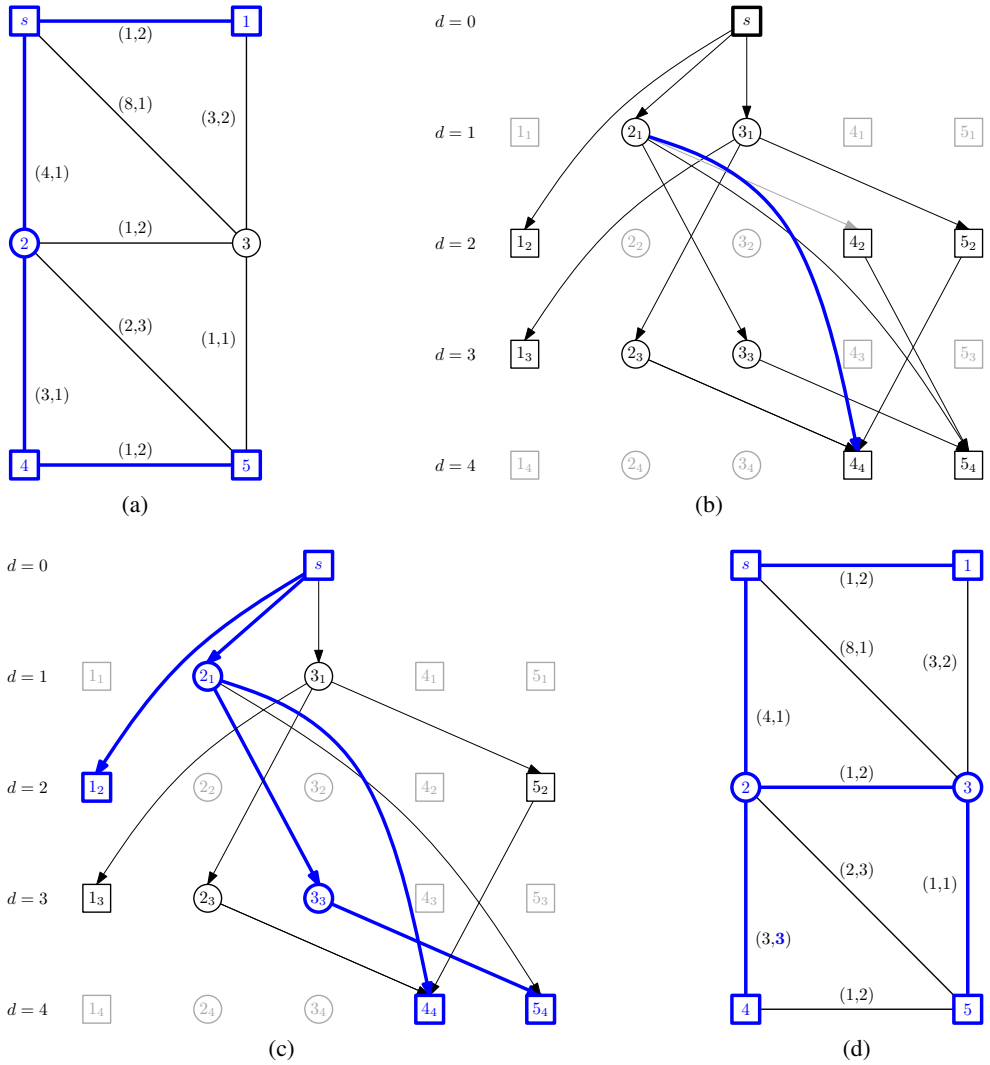


Figure 6.2: (a) Example graph G with edge labels (c_e, d_e) and $B = 4$. Squared nodes denote terminal nodes and blue arcs denote the optimal solution T^* with $c_{T^*} = 9$. (b) Layered graph $\hat{G}_{\mathrm{L}}$ derived from G_{L} by redirecting arc $(2_1, 4_2)$ to node 4_4. (c) Optimal solution $\hat{T}^*_{\mathrm{L}}$ in $\hat{G}_{\mathrm{L}}$. (d) Reverse transformed feasible solution $\hat{T}^*$ in G with $c_{\hat{T}^*} = 10$. Edge delay d_{24} is increased to 3.

Algorithm 6.1: ALF v1

Input: graph $G = (V, E)$, feasible model LAY on G_{L}
Output: an optimal solution T^* to the RDCST problem

$V_{\mathrm{L}}^0 = s \cup \{v_{d_v^{\min}}, v_{d_v^{\max}} : v \in V \setminus \{s\}\}$ `// initial node set`
$LB = 0,\ UB = \infty$ `// global lower and upper bounds`
$redirect = \mathrm{down}$ `// arc redirection` $\in \{\mathrm{down}, \mathrm{up}\}$
$i = 0$
while $LB \neq UB$ **do**
    build G_{L}^i depending on V_{L}^i and *redirect*
    obtain $T_{\mathrm{L,LP}}^i$ and T_{L}^i by solving model LAY on G_{L}^i
    transform T_{L}^i to T^i on G
    if $redirect == \mathrm{down}$ **then** $LB = c_{T^i}$
    if T^i *is feasible* **then**
        if $redirect == \mathrm{up}$ **then** (optionally) improve T^i by heuristics
        if $c_{T^i} < UB$ **then** $UB = c_{T^i},\ T^* = T^i$
    `// extend layered graph` G_{L}^i
    $V_{\mathrm{L}}^{i+1} = V_{\mathrm{L}}^i$
    forall the $(u_k, v_l) \in T_{\mathrm{L,LP}}^i \cup T_{\mathrm{L}}^i$ **do**
        if $l - k \neq d_{uv}$ **then** $V_{\mathrm{L}}^{i+1} = V_{\mathrm{L}}^{i+1} \cup \{v_{k+d_{uv}}\}$
    switch $redirect$, $i = i + 1$
return T^*

node $v_{k+d_{uv}}$ together with related outgoing arcs. If necessary, existing arcs pointing to a node $v_l,\ d_v^{\min} \leq l \leq d_v^{\max}$, are modified to either prevent a former redirection or to reduce the difference between the original and the current target layer. The resulting graph is denoted G_{L}^{i+1}. Applying Lemma 6.3.1 and 6.3.4 we know that $c_{T_{\mathrm{L}}^{i+1}} \geq c_{T_{\mathrm{L}}^i}$ if redirecting to lower layers and $c_{T_{\mathrm{L}}^{i+1}} \leq c_{T_{\mathrm{L}}^i}$ otherwise. These steps are repeated on G_{L}^{i+1} and further graphs until the two bounds match. Algorithm 6.1 shows this iterative solving process.

When redirecting to lower layers we have to consider that the resulting graph G_{L}^i does not necessarily need to be acyclic anymore. Therefore, $T_{\mathrm{L,LP}}^i$ or T_{L}^i of model LAY may be unconnected and contain cycles. Adding violated directed connection cut inequalities prevents these cycles and thus may lift the lower bound. When redirecting the arcs to higher layers we are not faced with this problem, so connection cut inequalities cannot improve generated upper bounds in this case. Additionally, every time we obtain a solution feasible in G we try to improve it by heuristic methods to further tighten the global upper bound.

Proposition 6.4.1. *Algorithm 6.1 terminates.*

Proof. As long as the optimal solution T_{L}^i in graph G_{L}^i is infeasible for the RDCST problem when calculating a lower bound, T_{L}^i must contain at least one redirected arc. Adding an appropriate node v_l to G_{L}^{i+1} prevents this redirection leading to a new solution T_{L}^{i+1}. In worst case

all nodes $v_l \in V_{\mathrm{L}}$ are added to the layered graph resulting in the original graph G_{L} with optimal solution T^*_{L}. Since $|V_{\mathrm{L}}|$ is finite the number of iterations is bounded. □

To adapt ALF to work for related problems one just has to replace model LAY by an appropriate model for the considered problem on the layered graph. To enhance upper bounds some problem specific heuristics could optionally be provided, too.

A disadvantage of the ALF variant described in Algorithm 6.1 may be the repeated time-consuming solving of complete MIPs, similarly to the approach by Wang et al. [186] for the TSPTW. Since the main motivation for ALF was to obtain the tight LP bounds of complete layered graph models, we now primarily focus on approximating this desired LP value in a first phase and then solve the corresponding MIP in a reduced layered graph with additional valid inequalities to ensure feasibility of the formulation for the original problem, similarly as Dash et al. [37] did for the TSPTW. Algorithm 6.2 shows this improved ALF variant.

In most iterations of the first phase we build reduced layered graphs by redirecting arcs to lower layers, solve the LP relaxation of an according model, and extend the layered graph in the same way as described above. Thus, we obtain a series of monotonically increasing lower bounds to the LP relaxation value of the model on the complete layered graph. Since we may not be able to construct the complete layered graph and compute the corresponding LP value because of time and/or memory limits, we actually do not know how good our current approximated LP value is. Therefore, from time to time we also build reduced layered graphs by redirecting arcs to higher layers and compute the according LP values which obviously are upper bounds to the desired LP value. The gap between the best lower and upper LP bounds provides a useful measure to evaluate our current approximate LP bound. Additionally, solving the integer model on these upper bound graphs yields a series of monotonically decreasing feasible primal bounds which can be used to support the presolving of CPLEX and further prune the branch-and-bound tree in the final MIP solving phase. The decision when to obtain upper LP or integer bounds is controlled by a predefined parameter $\eta \in \mathbb{Z}$:

- $\eta = 0$: No upper LP or integer bounds are calculated.
- $\eta < 0$: Every $|\eta|$ iterations we compute an upper LP bound.
- $\eta > 0$: Every η iterations we compute upper LP and integer bounds.

Let $\delta \in [0, 1]$ be a second input parameter. The layered graph extension phase is stopped if at least one of the following conditions is met:

- The gap between the best lower and upper LP bounds is less than or equal to δ.
- The relative increase of the lower LP bound of the last five values is less than or equal to δ.
- The reduced layered graph has not been extended according to the last lower LP bound calculation.

The final layered graph node set is then used in the second phase to compute an optimal solution to the original problem. First, we construct a reduced layered graph $\check{G}_{\mathrm{L}}$ based on the final

Algorithm 6.2: ALF v2

Input: graph $G = (V, E)$, feasible model LAY on G_{L}, parameters $\delta \in [0, 1]$, $\eta \in \mathbb{Z}$
Output: an optimal solution T^* to the RDCST problem

$V_{\mathrm{L}}^0 = s \cup \{v_{d_v^{\min}}, v_{d_v^{\max}} : v \in V \setminus \{s\}\}$ // initial node set
$LB_{\mathrm{LP}} = 0,\ UB_{\mathrm{LP}} = UB = \infty$ // global lower and upper bounds
$gap_{\mathrm{LP}} = 1$ // gap between lower and upper LP bounds
$inc_{\mathrm{LP}} = 1$ // relative increase of lower LP bound of last five values
$redirect = \mathrm{down}$ // arc redirection $\in \{\mathrm{down}, \mathrm{up}\}$
$i = 0$
// phase 1: layered graph extension + primal bounds
while $gap_{\mathrm{LP}} > \delta \wedge inc_{\mathrm{LP}} > \delta$ **do**
  build G_{L}^i depending on V_{L}^i and $redirect$
  obtain $T_{\mathrm{L,LP}}^i$ by solving model LAY_{LP} on G_{L}^i
  if $redirect == \mathrm{down}$ **then** $LB_{\mathrm{LP}} = c_{T_{\mathrm{L,LP}}^i}$, update inc_{LP}
  if $redirect == \mathrm{up}$ **then**
    $UB_{\mathrm{LP}} = c_{T_{\mathrm{L,LP}}^i}$
    if $\eta > 0$ **then**
      obtain T_{L}^i by solving model LAY on G_{L}^i
      transform T_{L}^i to T^i on G, (optionally) improve it by heuristics
      if $c_{T^i} < UB$ **then** $UB = c_{T^i}$
  $V_{\mathrm{L}}^{i+1} = V_{\mathrm{L}}^i$
  forall the $(u_k, v_l) \in T_{\mathrm{L,LP}}^i \cup T_{\mathrm{L}}^i$ **do**
    if $l - k \neq d_{uv}$ **then** $V_{\mathrm{L}}^{i+1} = V_{\mathrm{L}}^{i+1} \cup \{v_{k+d_{uv}}\}$
  if $redirect == \mathrm{down} \wedge i \bmod |\eta| == 0$ **then** $redirect = \mathrm{up}$
  if $redirect == \mathrm{up}$ **then** $redirect = \mathrm{down}$
  $gap_{\mathrm{LP}} = (UB_{\mathrm{LP}} - LB_{\mathrm{LP}})/UB_{\mathrm{LP}}$
  $i = i + 1$
// phase 2: solve final MIP
build $\check{G}_{\mathrm{L}}$ depending on V_{L}^i and $redirect = \mathrm{down}$
obtain $\check{T}_{\mathrm{L}}$ by solving model LAY (+ additional inequalities) on $\check{G}_{\mathrm{L}}$
transform $\check{T}_{\mathrm{L}}$ to T^* on G
return T^*

node set V_{L}^i by redirecting arcs to lower layers. Proposition 6.3.3 tells us that a feasible model for the complete layered graph applied to $\check{G}_{\mathrm{L}}$ results in a relaxation to the original problem. Thus, we need additional valid inequalities to ensure feasibility according to the original problem: Since redirecting arcs to lower layers may annihilate the acyclicity of the layered graph we have to explicitly ensure a connected solution, e.g. by adding directed connection cut inequalities (4.27) in graph G' or their stronger counterparts (4.50) in layered graph G'_{L}. Furthermore, decreasing edge delays may allow longer path-delays in a solution possibly leading to a violation of the delay-bound. These invalid paths can be prevented e.g. by separating the lifted infeasible path inequalities (4.33) which clearly are redundant on the complete layered graph. The small subset (4.34) of inequalities (4.33) is added a priori to the reduced layered graph model while inequalities (4.27), (4.50), and (4.33) are added dynamically in the usual branch-and-cut way.

Depending on the chosen layered graph model for the RDCST problem, it could be necessary to separate violated inequalities already in the first phase when only solving LP relaxations. In many cases it definitely makes sense to add strengthening inequalities in this phase to further tighten the obtained LP bounds. Additionally, found inequalities can be re-used in future iterations by adding them a priori to the model. However, this is only valid for inequalities defined on original graph G', e.g. for directed connection cuts (4.27) and infeasible path inequalities (4.33), and not for sets defined on the layered graph, e.g. connection cuts (4.50) on G'_{L}, since the structure of the current graph G_{L}^i may change dramatically by extending node set V_{L}^i to V_{L}^{i+1}. Unfortunately, we found so far no meaningful way to adapt violated connection cut inequalities (4.50) found in G_{L}^i to be re-used in G_{L}^{i+1}: It is not difficult to feasibly assign new layered graph nodes to a cut-set but in preliminary tests nearly all inequalities generated this way turned out to be redundant without strengthening the LP relaxation.

To conclude, in contrast to the approach by Dash et al. [37] for the TSPTW, the improved variant of ALF provides both a measure to evaluate the quality of the approximated LP bound and a way to obtain decreasing primal bounds without need for problem-specific heuristics.

6.5 Computational Results

In this section we compare ALF to two of the best approaches for the RDCST problem from Chapter 4. In detail we discuss the following solution methods:

- BP: branch-and-price approach by Leitner et al. [113] based on path formulation $PATH$ from Section 4.5 stabilizing column generation by using alternative dual-optimal solutions (algorithm-specific parameter Q is set to 20)

- L3: branch-and-cut based on layered graph formulation LAY in Section 4.9 with inequalities (4.30)–(4.31) added a priori to the model and extended by directed connection cut inequalities (4.50) on layered graph G'_{L}

- A1: basic ALF variant shown in Algorithm 6.1 and based on the same layered graph model as L3

- A2G: improved ALF variant shown in Algorithm 6.2 and based on the same layered graph model as L3 but replacing directed connection cut inequalities (4.50) on layered graph G'_{L} by their weaker counterparts (4.27) on original graph G'
- A2GL: improved ALF variant shown in Algorithm 6.2 and based on the same layered graph model as L3

We selected BP since it performed quite well throughout all instance sets, and L3 because it yields the best LP bounds and is among the leading approaches for the RDCST problem. Additionally, we want to directly compare an approach on a complete layered graph to our adaptive method working only on reduced layered graphs.

We chose the layered graph model of L3 also for A1 since preliminary tests showed that this model performed best within this basic ALF approach. Directed connection cut inequalities (4.50) on layered graph G'_{L} in approach A2GL are only separated in the final MIP phase since it is too time-consuming to add them within each LP relaxation computation in the first layered graph extension phase. Instead of them, we search for violated connection cuts (4.27) on graph G' as in approach A2G since they can easily be re-used in further LP solvings.

6.5.1 Test Instances and Environment

We re-use all benchmark instances for the RDCST problem from Section 4.11.1: the diverse sets from Gouveia et al. [68], the self-generated random instances from Section 3.13.1, and the modified SteinLib instances from Leggieri et al. [111]. Similarly, we also apply all preprocessing methods described in Section 3.3 and 4.3 prior to solving. To provide an initial primal bound for the ALF approaches we use the same heuristic methods as described in Section 4.11.1. Additionally, in case of spanning tree instances all primal bounds found are improved by the VND from Section 3.8. IBM ILOG CPLEX 12.3 is applied to solve the LP and MIP models within ALF, configured in the same way as for the other pure branch-and-cut approaches. We set a memory limit of 3 GB for each test which is for technical reasons less than the 4 GB limit for approaches BP and L3. However, in contrast to them ALF is quite economical in consuming memory due to the much smaller layered graphs and MIP models. Thus, the difference of both limits is negligible. The overall time limit for each test run is set to 10 000 seconds and the time for calculating an upper integer bound in the first phase of A2G and A2GL is bounded by one percent of the residual runtime since these primal bounds should only support the final MIP phase and not dominate the overall solving process. All tests are performed either on a single core of Intel Xeon E5540 processors with 2.53 GHz where eight cores share 24 GB of memory, or on a single core of Intel Xeon E5649 processors with 2.53 GHz where 12 cores share 48 GB of memory. In preliminary tests both systems yielded nearly the same performance within usual limits of tolerances.

6.5.2 Framework Results

Tables 6.1–6.4 show results for all considered instance sets for the RDCST problem. We report obtained average gaps between the best primal and dual bounds, the median runtime to reach these bounds, and the number of instances solved to optimality within the time limit. Dashes in

		average gap in %									median time in seconds									# optimal solutions (out of 5)								
		BP	L3	A1	A2G			A2GL			BP	L3	A1	A2G			A2GL			BP	L3	A1	A2G			A2GL		
Set	$B\backslash\eta$	-	-	-	-3	0	3	-3	0	3	-	-	-	-3	0	3	-3	0	3	-	-	-	-3	0	3	-3	0	3
R5	6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	2	**0**	1	**0**	**0**	1	**0**	1	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	8	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	3	**0**	2	1	1	1	1	1	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	10	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	3	**0**	2	2	2	2	2	2	2	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	12	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	4	**0**	2	1	1	1	1	1	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
C5	6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	2	**0**	1	**0**	**0**	**0**	**0**	1	**0**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	8	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	3	**0**	2	2	2	2	2	1	2	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	10	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	7	**1**	11	8	8	11	6	5	6	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	12	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	7	**3**	17	15	12	14	6	8	7	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
E5	6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	8	**1**	7	7	7	6	3	3	4	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	8	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	18	**3**	21	18	18	28	8	10	12	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	10	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	50	**6**	46	92	196	153	21	27	29	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	12	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	157	**22**	91	750	675	1566	45	53	55	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
R10	10	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	3	**0**	2	1	1	1	1	1	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	15	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	4	**0**	3	4	3	3	4	3	3	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	20	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	4	**0**	7	4	4	5	4	4	4	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	25	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	4	**1**	7	6	7	6	6	7	7	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
C10	10	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	4	**0**	2	2	2	2	2	2	2	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	15	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	4	**2**	5	5	6	5	5	6	5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	20	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**6**	10	14	17	23	17	10	15	12	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	25	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	27	22	36	43	43	32	29	24	**20**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
E10	10	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	7	**1**	7	5	5	6	4	4	5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	15	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	43	**10**	59	118	112	148	22	25	27	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	20	**0.0**	**0.0**	**0.0**	0.3	**0.0**	0.4	**0.0**	**0.0**	**0.0**	405	**83**	392	6198	2432	9196	180	140	254	**5**	**5**	**5**	4	**5**	3	**5**	**5**	**5**
	25	**0.0**	**0.0**	**0.0**	2.8	1.6	2.8	**0.0**	**0.0**	0.3	1425	**317**	1361	-	-	-	651	751	1419	**5**	**5**	**5**	1	2	0	**5**	**5**	4

Table 6.1: Results for instances by Gouveia et al. [68] (B: delay-bound, η: parameter for upper LP and integer bounds, gap: gap between best primal and dual bound, BP: stabilized branch-and-price, L3: layered graph approach with connection cuts on G'_{L}, A1: basic ALF variant, A2G(L): improved ALF variant with connection cuts on G' (G'_{L}), best results are printed bold).

		average gap in %									median time in seconds									# optimal solutions (out of 5)								
		BP	L3	A1	A2G			A2GL			BP	L3	A1	A2G			A2GL			BP	L3	A1	A2G			A2GL		
Set	$B\backslash\eta$	-	-	-	-3	0	3	-3	0	3	-	-	-	-3	0	3	-3	0	3	-	-	-	-3	0	3	-3	0	3
R100	100	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**4**	5	9	**4**	6	5	**4**	6	**4**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	150	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**4**	10	5	6	5	8	5	6	6	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	200	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	6	19	6	**5**	10	7	6	9	6	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	250	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	6	62	8	6	9	**5**	6	9	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
C100	100	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**8**	228	35	34	46	27	29	25	24	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	150	**0.0**	0.9	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**10**	627	69	162	231	71	79	80	72	**5**	4	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	200	**0.0**	2.1	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**13**	7747	67	544	169	201	146	188	121	**5**	3	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	250	**0.0**	3.9	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**14**	-	175	59	62	57	58	55	53	**5**	0	**5**	**5**	**5**	**5**	**5**	**5**	**5**
E100	100	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**16**	593	370	991	962	661	146	159	159	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	150	**0.0**	2.1	**0.0**	1.9	1.7	2.0	**0.0**	**0.0**	**0.0**	**40**	9162	672	-	6249	-	419	650	415	**5**	3	**5**	2	3	2	**5**	**5**	**5**
	200	**0.0**	10.4	0.3	5.0	5.3	5.6	0.4	0.3	**0.0**	**245**	-	4051	-	-	-	5448	4843	3471	**5**	0	4	0	0	0	4	4	**5**
	250	**0.0**	11.7	0.7	8.1	7.3	5.5	2.6	1.4	1.5	**425**	-	-	-	-	-	6508	5889	6399	**5**	0	2	0	0	0	3	4	3
R1000	1000	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	12	245	12	**6**	8	10	7	10	9	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	1500	**0.0**	2.4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**18**	9586	37	67	73	55	33	64	34	**5**	3	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	2000	**0.0**	0.9	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	20	3548	**9**	21	20	10	21	17	10	**5**	4	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	2500	**0.0**	20.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	25	6099	10	7	8	7	**6**	7	**6**	**5**	4	**5**	**5**	**5**	**5**	**5**	**5**	**5**
C1000	1000	**0.0**	2.7	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	17	6018	19	10	12	11	**9**	10	10	**5**	3	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	1500	**0.0**	6.8	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**29**	-	108	121	122	80	74	75	59	**5**	0	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	2000	**0.0**	13.2	**0.0**	**0.0**	0.1	**0.0**	**0.0**	**0.0**	**0.0**	**39**	-	325	1820	1549	833	552	710	309	**5**	0	**5**	**5**	4	**5**	**5**	**5**	**5**
	2500	**0.0**	64.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**33**	-	215	424	635	337	280	228	200	**5**	0	**5**	**5**	**5**	**5**	**5**	**5**	**5**
E1000	1000	**0.0**	8.9	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**26**	-	145	2010	1105	1710	89	147	123	**5**	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	1500	**0.0**	16.1	**0.0**	4.7	3.4	2.6	0.1	**0.0**	**0.0**	**50**	-	625	-	-	3377	939	1256	636	**5**	0	4	1	2	3	4	**5**	**5**
	2000	**0.0**	33.4	0.2	4.4	6.9	4.2	0.8	**0.0**	0.1	**88**	-	3153	-	-	-	7614	2213	2376	**5**	0	4	1	0	1	3	**5**	4
	2500	**0.0**	-	0.9	8.8	5.6	4.4	5.3	6.6	3.6	**218**	-	5952	-	-	-	3929	-	-	**5**	0	3	0	0	0	3	1	2

Table 6.2: Results for instances by Gouveia et al. [68] (B: delay-bound, η: parameter for upper LP and integer bounds, gap: gap between best primal and dual bound, BP: stabilized branch-and-price, L3: layered graph approach with connection cuts on G'_{L}, A1: basic ALF variant, A2G(L): improved ALF variant with connection cuts on G' (G'_{L}), best results are printed bold).

		average gap in %									median time in seconds									# optimal solutions (out of 30)								
		BP	L3	A1	A2G			A2GL			BP	L3	A1	A2G			A2GL			BP	L3	A1	A2G			A2GL		
Set	$B\backslash\eta$	-	-	-	-3	0	3	-3	0	3	-	-	-	-3	0	3	-3	0	3	-	-	-	-3	0	3	-3	0	3
T10	16	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	1	**0**	**0**	**0**	**0**	**0**	**0**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
	30	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	2	**1**	11	8	9	9	8	9	8	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
	50	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**3**	7	41	40	44	43	38	44	40	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
	100	**0.0**	**0.0**	**0.0**	0.7	0.7	0.8	**0.0**	**0.0**	**0.0**	**5**	76	136	145	177	151	117	172	142	**30**	**30**	**30**	28	29	28	**30**	**30**	**30**
T30	16	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	2	**0**	1	1	1	1	1	1	1	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
	30	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	9	**2**	19	10	11	9	9	10	9	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
	50	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**20**	34	173	60	68	61	65	78	60	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
	100	**0.0**	**0.0**	0.2	1.8	0.8	0.5	2.8	0.3	**0.0**	**47**	362	580	329	453	283	283	356	297	**30**	**30**	29	25	27	27	27	29	**30**
T50	16	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	5	**0**	2	1	1	1	1	1	1	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
	30	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	0.1	**0.0**	0.1	19	**4**	37	16	15	13	13	14	14	**30**	**30**	**30**	**30**	**30**	**30**	29	**30**	29
	50	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	44	**33**	282	63	74	52	57	66	45	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
	100	**0.0**	**0.0**	1.4	1.2	0.7	1.1	2.8	3.9	0.5	**202**	768	2271	542	446	407	587	582	356	**30**	**30**	22	25	25	25	26	26	27
T70	16	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	8	**0**	2	1	1	1	1	1	1	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
	30	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	31	**4**	46	17	20	16	16	15	14	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
	50	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	79	50	227	58	73	54	78	71	**49**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
	100	**0.1**	0.9	1.3	0.7	1.9	1.0	3.3	2.2	0.3	**241**	873	2356	398	424	370	464	715	350	28	**29**	25	26	25	25	26	27	26
T99	16	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	15	**0**	2	1	2	2	1	2	2	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
	30	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	66	**4**	42	14	18	13	13	16	14	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
	50	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	0.1	**0.0**	**0.0**	150	**44**	238	63	64	57	67	63	60	**30**	**30**	**30**	**30**	**30**	**30**	29	**30**	**30**
	100	0.3	0.6	0.5	1.4	1.1	0.5	0.9	0.9	**0.1**	526	719	1622	357	453	333	502	574	**323**	28	28	26	25	26	27	28	28	**29**

Table 6.3: Results for random instances from Section 3.13.1 (B: delay-bound, η: parameter for upper LP and integer bounds, gap: gap between best primal and dual bound, BP: stabilized branch-and-price, L3: layered graph approach with connection cuts on G'_{L}, A1: basic ALF variant, A2G(L): improved ALF variant with connection cuts on G' (G'_{L}), best results are printed bold).

		average gap in %									median time in seconds									# opt. solutions (out of 18/10/5/5/5)								
		BP	L3	A1	A2G			A2GL			BP	L3	A1	A2G			A2GL			BP	L3	A1	A2G			A2GL		
Set	$\overline{B}\backslash\eta$	-	-	-	-3	0	3	-3	0	3	-	-	-	-3	0	3	-3	0	3	-	-	-	-3	0	3	-3	0	3
B-Ran	314	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**
	427	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**
B-Cor	40	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**
	54	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**
C-Ran	397	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	18	**2**	5	3	3	3	3	3	3	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
	541	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	0.1	0.7	1.5	0.1	48	106	10	**9**	11	11	19	**9**	**9**	**10**	**10**	9	9	9	9	9	9	9
C-Cor	50	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	0.1	1.0	**0.0**	23	**2**	7	5	4	5	6	5	11	**10**	**10**	**10**	**10**	9	**10**	9	9	**10**
	68	**0.0**	**0.0**	**0.0**	**0.0**	0.9	**0.0**	0.6	0.9	0.1	48	21	9	**3**	**3**	4	4	4	5	**10**	**10**	**10**	**10**	9	**10**	9	9	9
D-Ran	554	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	76	**3**	9	4	**3**	4	4	5	5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	755	**0.0**	0.1	**0.0**	**0.0**	1.0	**0.0**	0.1	1.9	0.1	145	1725	19	8	**7**	8	9	29	10	**5**	4	**5**	**5**	4	**5**	4	3	3
D-Cor	66	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	113	**2**	8	4	3	5	4	3	5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	90	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	199	32	6	**4**	**4**	**4**	**4**	**4**	**4**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
Berlin52-Ran	19	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	26	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
Berlin52-Cor	165	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	1	7	1	**0**	1	1	**0**	1	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	225	**0.0**	3.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	4	5155	3	3	3	**2**	3	3	4	**5**	4	**5**	**5**	**5**	**5**	**5**	**5**	**5**
Brazil58-Ran	20	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
	27	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	1	**0**	2	1	1	1	1	1	1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
Brazil58-Cor	3979	**0.0**	39.4	**0.0**	**0.0**	**0.0**	**0.0**	0.2	**0.0**	**0.0**	44	-	15	**5**	6	10	**5**	6	8	**5**	0	**5**	**5**	**5**	**5**	4	**5**	4
	5425	**0.0**	-	**0.0**	**0.0**	**0.0**	**0.0**	0.7	1.5	**0.0**	**67**	-	324	253	165	203	7111	2743	4102	**5**	0	**5**	**5**	**5**	**5**	3	4	4

Table 6.4: Results for instances by Leggieri et al. [111] ($\overline{B}$: average delay-bound, η: parameter for upper LP and integer bounds, gap: gap between best primal and dual bound, BP: stabilized branch-and-price, L3: layered graph approach with connection cuts on G'_{L}, A1: basic ALF variant, A2G(L): improved ALF variant with connection cuts on G' (G'_{L}), best results are printed bold).

gap and time columns represent 100% and 10 000 seconds, respectively. Parameter δ controlling some of the stopping criteria for the layered graph extension phase in A2G and A2GL is set to 0.01 which turned out to work well in preliminary tests, see Section 6.4 and Algorithm 6.2 for details. Furthermore, parameter η controlling the calculation of upper LP and integer bounds is varied in $\{-3, 0, 3\}$. Some tests were performed to initialize the first layered graph in a more sophisticated way, e.g. based on the initial heuristic solutions, but following the proposed trivial way mostly yielded the best results.

In general, lower absolute values of η result in more frequent upper bound computations and thus higher runtime overhead in the first phase. In principle, we only need upper LP bounds to measure the quality of our lower LP bounds. However, in many cases a synergy effect could be observed: The layered graph extensions arising from LP solutions when redirecting arcs to higher layers often result in a faster convergence of the lower LP bound. Currently, we are not sure about the reasons for this effect but this will be analyzed in more detail in future work.

Additionally, it would be enough to compute an upper integer bound immediately before entering the final MIP phase since we know that this bound has to be the best one over all reduced layered graphs. But then we would have to solve a complete MIP model on a possibly already large layered graph without a previously known primal bound used for pruning the branch-and-bound tree. Indeed we can use the initial heuristic primal bound which, however, may be quite weak. On the other hand, if we obtain upper bounds from time to time on smaller layered graphs then first we have more possibilities to improve them by heuristics and second we can utilize the best bound in further MIP computations. In many cases, this approach could significantly accelerate the overall primal bound calculations, although we have to repeatedly solve growing MIPs. Finally, in case of limited runtime obtaining upper integer bounds usually yields tight gaps already in the first layered graph extension phase.

To summarize, when comparing different values of η for A2G and A2GL, in most cases it is beneficial to compute upper primal bounds ($\eta = 3$) but there are still situations where these bounds are not needed in the final MIP phase and thus the calculations of them only produce runtime overhead. Similarly, due to the synergy effect discussed above it is also preferable to calculate upper LP bounds ($\eta = -3$) instead of completely ignoring them ($\eta = 0$). Finally, we could not observe any significant improvements by further increasing $|\eta|$.

One would expect that the basic ALF variant A1 produces too much overhead by iteratively solving MIPs quickly increasing in size. This can be clearly observed e.g. in the results for the random instances in Table 6.3. However, this is mostly not the case for the Gouveia instances in Tables 6.1 and 6.2 where A1 can obviously compete with A2G and A2GL. Here, the same argumentation as above holds: A1 benefits from the steadily tightening series of lower and upper bounds obtained by the MIPs. Especially computing lower integer bounds clearly provides even stronger bounds on the optimal value than just lower LP bounds.

Compared to L3 on the full layered graph ALF can dramatically improve the performance on instances with a large set of achievable delay values and huge bounds, as shown in Tables 6.2 and 6.4, in many cases even by orders of magnitude. Additionally, ALF consumes substantially less memory since the graphs it works on are significantly smaller than the full layered graphs, see Table 6.5. However, BP is still superior on the instances in Table 6.2. As results in Tables 6.1 and 6.3 indicate, for instances with low delay-bounds the framework causes too much computa-

tional overhead and does not pay off in the end. Finally, the most robust and best performance of ALF is observed for the instances by Leggieri et al. [111] in Table 6.4. Here, all ALF approaches outperform BP and L3 in all cases except for the Brazil58-Cor instances with $\overline{B} = 5425$ where also the reduced layered graphs become quite large due to the huge delay-bounds.

Table 6.5 provides statistics of the ALF experiments on some selected instance sets: average numbers of solved LPs and MIPs, average sizes of preprocessed original graphs G' and full layered graphs G_{L} used in approach L3, and average sizes of reduced layered graphs G_{L}^i in the last iteration of ALF relative to G_{L} in percent. Since A2G and A2GL only differ by the sets of valid inequalities added in the final MIP phase the presented values are the same for both. According to Lemma 6.3.2, a solution to the LP relaxation on a reduced layered graph when redirecting arcs to lower layers is optimal if it is feasible for the original problem; this explains the zeroes in the columns of solved MIPs for A2G(L). In general, the number of ALF iterations stays quite low, both for the basic A1 and the improved A2G(L). Even if A1 solved in average up to 32 MIPs for Brazil58 instances the according runtimes are quite moderate and especially much better than solving one MIP on the full layered graph, see Table 6.4. The reason for this immediately becomes obvious when looking at the reduced layered graph sizes of ALF approaches. Particularly, for instances with large delay-bounds, e.g. R1000, E1000, and Brazil58-Cor, the reduced layered graphs are tiny compared to their full counterparts. This indicates that actually only a small part of the complete layered graph is relevant to prove optimality. However, instance sets R5 and E5 with low delay-bounds represent the opposite behavior: Here, ALF sometimes needs more than 90% of G_{L} and thus it makes more sense to just solve one MIP on G_{L} which is clearly visible in Table 6.1.

6.6 Case Study: Quota-Constrained Rooted Delay-Constrained Steiner Tree Problem

We now consider the application of ALF to the *Quota-Constrained Rooted Delay-Constrained Steiner Tree (QCRDCST) Problem*, a variant of the RDCST problem from Chapter 4 where not all terminal nodes R need to be connected. Each terminal node $v \in R$ is assigned a node prize $p_v \geq 0$, and the objective is to identify a delay-constrained solution tree T^* yielding minimum total costs, while the sum of prizes of connected terminal nodes must be at least equal to a given quota value Q, i.e.

$$\sum_{v \in (T^* \cap R)} p_v \geq Q. \tag{6.1}$$

To the best of our knowledge, this problem variant has not been considered before. However, the variant without considering delay-constraints – the prize-collecting Steiner tree (PCST) problem with a quota-constraint – has been studied in many articles: Johnson et al. [93] improve an existing primal-dual 2-approximation algorithm for the PCST problem. Haouari et al. [83] present a hybrid approach combining Lagrangian relaxation and a genetic algorithm to obtain lower and upper bounds to the optimal solution, respectively. In [81] a generalized variant of the PCST problem is considered and solved by Lagrangian relaxation comparing different subgradient strategies. Three exact MIP formulations are presented in [82] based on MTZ con-

		avg. # LPs				avg. # MIPs				$\overline{\lvert V\rvert}$	$\overline{\lvert V_\mathrm{L}\rvert}$	$\overline{\lvert V_\mathrm{L}^i\rvert/\lvert V_\mathrm{L}\rvert}$ in %				$\overline{\lvert A\rvert}$	$\overline{\lvert A_\mathrm{L}\rvert}$	$\overline{\lvert A_\mathrm{L}^i\rvert/\lvert A_\mathrm{L}\rvert}$ in %			
		A1	A2G(L)			A1	A2G(L)					A1	A2G(L)					A1	A2G(L)		
Set	$\overline{B}\backslash\eta$	-	-3	0	3	-	-3	0	3			-	-3	0	3			-	-3	0	3
R5	6	7	6	5	6	3	0	0	1	41	194	76.7	78.5	78.4	78.5	585	1574	75.7	77.6	77.4	77.6
	8	9	8	8	8	5	0	1	1	41	273	73.7	74.2	76.4	74.2	652	2792	70.6	71.2	73.8	71.4
	10	12	9	10	9	6	1	1	2	41	352	68.1	68.6	71.2	68.7	652	4030	65.4	65.6	68.5	65.6
	12	12	11	9	10	5	0	0	1	41	432	59.4	61.3	60.1	60.3	652	5273	57.0	58.7	57.5	57.4
E5	6	7	5	6	5	7	1	1	2	41	197	91.3	89.3	91.8	89.3	914	2328	90.9	88.6	91.8	88.6
	8	8	8	8	8	7	1	1	3	41	276	91.0	92.8	93.7	93.1	1083	4369	89.7	92.2	93.1	92.5
	10	8	9	9	9	7	1	1	3	41	355	88.0	90.7	92.1	90.8	1083	6448	85.5	88.9	90.5	89.1
	12	9	9	10	9	7	1	1	3	41	434	86.9	89.4	91.4	89.9	1083	8531	85.5	88.1	90.5	88.5
R1000	1000	13	14	16	14	12	1	1	4	41	32068	1.9	2.3	2.6	2.3	721	335931	1.8	2.2	2.4	2.2
	1500	12	16	16	16	10	1	1	4	41	52068	1.6	2.2	3.0	2.3	751	694494	1.5	2.1	2.7	2.1
	2000	12	15	16	15	8	1	1	3	41	72068	0.8	1.2	1.3	1.1	751	1059390	0.8	1.2	1.3	1.1
	2500	12	13	15	13	6	0	0	2	41	92068	0.6	0.9	1.0	0.9	751	1424290	0.6	0.9	1.0	0.9
E1000	1000	14	12	13	13	13	1	1	4	41	32149	3.5	3.2	4.2	3.5	1128	444632	3.2	3.1	4.0	3.4
	1500	13	13	14	13	12	1	1	4	41	52149	3.6	3.3	4.4	3.5	1197	1014190	3.3	3.0	4.0	3.2
	2000	12	13	12	13	11	1	1	4	41	72149	2.9	3.0	2.7	2.8	1197	1598790	2.8	2.8	2.6	2.7
	2500	10	12	13	12	9	1	1	4	41	92149	2.1	2.3	1.6	2.3	1197	2183390	2.1	2.3	1.6	2.3
T50	16	10	8	8	8	6	1	1	1	99	781	46.6	47.3	48.0	47.4	743	2914	51.1	52.2	52.9	52.3
	30	13	11	12	11	11	1	1	3	100	2161	34.4	35.2	37.0	35.3	1627	19020	33.5	34.5	36.3	34.5
	50	15	13	13	13	13	1	1	3	100	4136	27.8	28.5	30.7	28.9	2295	57733	26.0	26.8	29.0	27.2
	100	15	16	16	16	14	1	1	4	100	9081	17.2	23.3	25.5	23.3	3109	192886	15.7	21.5	23.8	21.6
D-Ran	554	24	10	7	10	8	1	1	1	528	45502	12.8	12.4	12.1	12.4	1365	66802	22.4	21.9	21.5	21.9
	755	30	13	9	13	11	1	1	1	657	144159	2.9	2.4	2.4	2.4	1750	270689	4.7	4.3	4.3	4.3
D-Cor	66	18	11	8	11	2	0	0	0	552	10554	20.6	19.6	19.4	19.6	1427	17759	29.8	29.5	29.3	29.5
	90	21	15	14	15	2	0	0	1	670	24149	8.2	8.3	8.5	8.3	1794	47470	10.7	10.8	11.4	10.8
Brazil58-Ran	20	8	6	5	6	2	0	0	0	55	318	49.3	47.8	48.8	47.8	233	630	64.2	62.7	63.7	62.7
	27	8	7	7	7	6	1	1	1	57	697	32.4	32.2	34.4	32.2	422	2336	38.0	38.2	40.2	38.2
Brazil58-Cor	3979	30	13	14	13	20	1	1	3	58	102400	0.5	0.2	0.2	0.2	909	1065070	0.8	0.3	0.3	0.4
	5425	40	15	13	14	32	1	1	4	58	179983	0.9	0.2	0.1	0.2	968	2266490	1.1	0.3	0.2	0.2

Table 6.5: Statistics for several instance sets ($\overline{B}$: average delay-bound, η: parameter for upper LP and integer bounds, $\overline{\lvert V\rvert}$, $\overline{\lvert A\rvert}$, $\overline{\lvert V_\mathrm{L}\rvert}$, $\overline{\lvert A_\mathrm{L}\rvert}$: avg. sizes of graphs G' and G_L, $\overline{\lvert V_\mathrm{L}^i\rvert/\lvert V_\mathrm{L}\rvert}$, $\overline{\lvert A_\mathrm{L}^i\rvert/\lvert A_\mathrm{L}\rvert}$: avg. reduced layered graph sizes in last ALF iteration relative to full layered graph G_L, A1: basic ALF variant, A2G(L): improved ALF variants).

straints for cycle elimination and a reformulation technique originally introduced by Sherali and Adams [170].

6.6.1 Layered Graph Model

We revise and adapt the layered graph model LAY from Section 4.9 to the QCRDCST problem. The transformation to layered graph G_{L} as described in Section 4.8 can be used in the same way here, too. However, we additionally have to model the quota constraint. For this purpose we introduce binary node variables $y_v,\ \forall v \in V \setminus \{s\}$, to decide whether to include a node in the solution or not. The other sets of variables are adopted in a straight-forward way: We use binary variables $x_{uv},\ \forall (u,v) \in A$, to model original arcs in directed graph G'. Continuous variables $y^l_v,\ \forall v_l \in V_{\mathrm{L}} \setminus \{s\}$, and $x^k_{uv},\ \forall (u_k, v_l) \in A_{\mathrm{L}}$, represent nodes and arcs in layered graph G_{L}, respectively. Model $LAYQ$ is then defined as follows:

$$\min \quad \sum_{(u,v)\in A} c_{uv} x_{uv} \tag{6.2}$$

$$\text{s.t.} \quad \sum_{(u_k,v_l)\in A_{\mathrm{L}}} x^k_{uv} = y^l_v \qquad \forall v_l \in V_{\mathrm{L}} \setminus \{s\} \tag{6.3}$$

$$\sum_{(u_k,v_l)\in A_{\mathrm{L}}, u\neq w} x^k_{uv} \geq x^l_{vw} \qquad \forall (v_l, w_j) \in A^g_{\mathrm{L}} \tag{6.4}$$

$$\sum_{v_l \in V_L} y^l_v = y_v \qquad \forall v \in V \setminus \{s\} \tag{6.5}$$

$$x^0_{sv} = x_{sv} \qquad \forall (s,v) \in A \tag{6.6}$$

$$\sum_{(u_k,v_l)\in A_{\mathrm{L}}} x^k_{uv} = x_{uv} \qquad \forall (u,v) \in A,\ u \neq s \tag{6.7}$$

$$\sum_{v\in R} p_v y_v \geq Q \tag{6.8}$$

$$x^k_{uv} \geq 0 \qquad \forall (u_k, v_l) \in A_{\mathrm{L}} \tag{6.9}$$

$$y^l_v \geq 0 \qquad \forall v_l \in V_{\mathrm{L}} \setminus \{s\} \tag{6.10}$$

$$x_{uv} \in \{0,1\} \qquad \forall (u,v) \in A \tag{6.11}$$

$$y_v \in \{0,1\} \qquad \forall v \in V \setminus \{s\} \tag{6.12}$$

Indegree constraints (6.3) in G_{L} restrict the number of incoming arcs to a layered graph node v_l in dependency of y^l_v to at most one. Since G_{L} is acyclic, inequalities (6.4) are enough to ensure connectivity. Equalities (6.5), (6.6) and (6.7) link layered graph nodes and arcs to original nodes and arcs, respectively. Finally, inequality (6.8) forces the inclusion of terminal nodes with node prizes summing up to at least Q.

Additionally, directed connection cut inequalities on G' and G'_{L} have to be adapted to deal with the fact that there only has to be a connection to a terminal node if it is selected to be included in the solution. Thus, we incorporate node variables $y_w,\ w \in V \setminus \{s\}$, to form the

following inequalities:

$$\sum_{(u,v)\in A,\ u\in W,\ v\notin W} x_{uv} \geq y_w \qquad \forall W \subset V,\ s \in W,\ w \in \overline{W} \cap R \tag{6.13}$$

$$\sum_{(u_k,v_l)\in A'_{\mathrm{L}},\ u_k\in W_{\mathrm{L}},\ v_l\in\overline{W_{\mathrm{L}}}} x^k_{uv} \geq y_w \qquad \forall W_{\mathrm{L}} \subset V'_{\mathrm{L}},\ s \in W_{\mathrm{L}},\ \hat{w} \in \overline{W_{\mathrm{L}}} \cap R_{\mathrm{L}} \tag{6.14}$$

Note that since the right side of these inequalities can be less than one they are weaker than their counterparts for the RDCST problem. Similarly to Section 4.7, we a priori add a subset of inequalities (6.13) to model $LAYQ$ to provide a "hot-start" to the branch-and-cut algorithm:

$$\sum_{(s,v)\in A} x_{sv} \geq 1 \tag{6.15}$$

$$x_{uv} + x_{vu} \leq y_u \qquad \forall\{u,v\} \in E,\ u \neq s \tag{6.16}$$

$$x_{uv} + x_{vu} \leq y_v \qquad \forall\{u,v\} \in E,\ v \neq s \tag{6.17}$$

6.6.2 Computational Results

In this section we want to compare the performance of ALF to other solution approaches for the QCRDCST problem. The following methods are considered:

- BP: stabilized branch-and-price approach by Leitner et al. [113] based on path formulation $PATH$ from Section 4.5 slightly adapted to work with the quota-constraint
- L2: branch-and-cut based on layered graph formulation $LAYQ$ in previous Section 6.6.1 with inequalities (6.15)–(6.17) added a priori to the model and extended by directed connection cut inequalities (6.13) on original graph G'
- A1: basic ALF variant shown in Algorithm 6.1 and based on formulation $LAYQ$
- A1G: A1 extended by connection cut inequalities (6.13) on graph G'
- A1GL: A1 extended by connection cut inequalities (6.14) on graph G'_{L}
- A2G: improved ALF variant shown in Algorithm 6.2 and based on the same layered graph model as L2
- A2GL: A2G replacing directed connection cut inequalities (6.13) on graph G' by their stronger counterparts (6.14) on layered graph G'_{L}

The computational environment is exactly the same as for the previous ALF experiments on the RDCST Problem, see Section 6.5.1 for details. The only difference is that here we set a general memory limit of 3 GB, also for BP and L2. We derived set QD of benchmark instances from set D of preprocessed instances for the PCST problem [119] originally created by Canuto et al. [24]. Instances from subset A are used where terminal prizes are chosen randomly from

						gap in %												time in seconds											
						BP	L2	A1	A1G	A1GL	A2G			A2GL			BP	L2	A1	A1G	A1GL	A2G			A2GL		
Instance	B	$\|V\|$	$\|A\|$	$\|R\|$	$Q\backslash\eta$	-	-	-	-	-	-3	0	3	-3	0	3	-	-	-	-	-	-3	0	3	-3	0	3
D13-10	30	966	9009	143	30	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**36**	138	243	174	152	177	146	152	149	194
					60	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**48**	251	473	234	111	140	109	207	186	101
	41	966	9047	143	30	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	208	64	53	72	71	91	**51**	69	81	75
					60	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**115**	299	1088	2374	207	152	224	232	160	254
D13-100	228	965	8965	143	30	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	438	**55**	111	74	192	82	58	189	71	71
					60	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	760	417	1275	3614	162	168	**155**	363	252	164
	342	966	9044	143	30	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	100	113	101	125	107	**92**	134	116	119
					60	-	-	**0.0**	**0.0**	0.6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	700	1205	-	262	278	**212**	1853	2504	624
D13-1000	2230	966	8953	143	30	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	35	43	35	40	36	**26**	48	40	30
					60	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	167	511	341	139	**124**	185	160	136	152
	3409	966	9029	143	30	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	161	133	155	171	**112**	151	206	125	153
					60	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	402	3148	7725	445	403	**188**	1181	2006	223
D14-10	29	946	8905	207	30	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**37**	277	550	989	135	139	102	169	170	144
					60	-	**0.0**	**0.0**	2.9	2.4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**183**	2160	-	-	295	358	327	4346	879	404
	41	946	8914	207	30	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	87.9	**0.0**	-	541	471	565	6360	238	**196**	236	4776	-	369
					60	-	**0.0**	**0.0**	**0.0**	1.4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	374	369	4026	-	280	284	**186**	2438	609	203
D14-100	260	946	8912	207	30	-	-	**0.0**	3.8	1.9	2.8	2.8	**0.0**	1.9	3.7	**0.0**	-	-	3445	-	-	-	-	717	-	-	**487**
					60	-	-	**0.0**	8.3	5.6	**0.0**	**0.0**	**0.0**	**0.0**	0.4	**0.0**	-	-	4566	-	-	7360	597	594	5036	-	**411**
	381	946	8915	207	30	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	3.2	**0.0**	-	-	612	443	924	502	375	**248**	3356	-	339
					60	-	-	**0.4**	4.0	5.8	5.5	**0.4**	**0.4**	72.3	72.2	0.9	-	-	-	-	-	-	-	-	-	-	-
D14-1000	2209	946	8842	207	30	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	77	151	129	**35**	39	**35**	40	40	46
					60	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	299	3453	3432	**93**	150	96	94	189	102
	3194	946	8911	207	30	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	634	1455	757	872	335	**263**	4872	841	299
					60	-	-	**0.0**	1.8	1.3	**0.0**	**0.0**	**0.0**	1.8	**0.0**	**0.0**	-	-	335	-	-	283	332	**245**	-	2234	768
D15-10	26	832	8198	348	30	-	**0.0**	**0.0**	2.8	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**95**	513	-	3644	327	365	207	993	787	182
					60	-	**0.0**	**0.0**	6.4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**60**	481	-	7583	238	282	342	403	276	305
	37	832	8210	348	30	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	581	285	867	3186	**231**	739	295	1890	1111	235
					60	-	**0.0**	**0.0**	4.9	3.2	**0.0**	0.7	**0.0**	74.9	74.9	0.4	-	**947**	2790	-	-	6923	-	1013	-	-	-
D15-100	218	832	8174	348	30	-	**0.0**	**0.0**	5.4	3.6	**0.0**	**0.0**	**0.0**	90.9	90.9	0.9	-	**1075**	3722	-	-	2220	1400	1748	-	-	-
					60	-	**0.0**	**0.0**	8.4	3.1	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	881	605	-	-	349	483	243	2388	5180	**222**
	323	832	8205	348	30	-	-	**0.0**	**0.0**	2.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	1157	2127	-	637	768	**381**	1156	1126	746
					60	-	-	**0.0**	5.9	2.6	1.5	74.9	**0.0**	74.9	74.9	0.8	-	-	6461	-	-	-	-	**2746**	-	-	-
D15-1000	2415	832	8192	348	30	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	130	139	218	82	95	**65**	110	100	81
					60	-	-	**0.0**	4.2	0.7	**0.0**	**0.0**	**0.0**	77.4	77.4	0.4	-	-	**595**	-	-	6681	757	718	-	-	-
	3385	832	8201	348	30	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	437	431	977	499	510	**322**	2049	1889	345
					60	-	-	**0.0**	1.9	1.5	**0.0**	**0.0**	**0.0**	75.2	75.2	0.4	-	-	1721	-	-	1094	1193	**919**	-	-	-
D18-10	18	944	18148	111	30	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**20**	28	41	26	60	88	35	69	100	40
					60	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**51**	534	501	3252	971	1199	255	1893	4004	286
	27	944	19125	111	30	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	579	81	48	**40**	206	265	75	407	360	74
					60	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	77.5	**0.0**	**0.0**	-	572	669	1061	2655	1412	1400	**242**	6235	-	590
D18-100	139	944	18016	111	30	-	94.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	168	123	**86**	624	744	209	747	1072	215
					60	-	77.6	3.4	6.9	8.6	**0.0**	3.4	**0.0**	77.2	77.2	1.8	-	-	-	-	-	9677	-	**2209**	-	-	-
	222	944	19146	111	30	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	138	117	127	1642	2027	**89**	2456	2476	1316
					60	-	-	**2.0**	3.9	5.9	3.9	3.9	3.9	77.4	77.2	3.9	-	-	-	-	-	-	-	-	-	-	-
D18-1000	1315	944	17830	111	30	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	44	35	50	160	202	29	117	154	**26**
					60	-	-	**0.0**	3.4	8.6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	2181	-	-	1111	866	**695**	2963	1911	742
	2143	944	19146	111	30	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	-	110	52	**49**	204	281	148	1541	741	160
					60	-	-	2.0	2.0	7.8	3.9	3.9	**0.0**	76.6	76.6	**0.0**	-	-	-	-	-	-	-	**1148**	-	-	1234
D19-10	17	897	17483	147	30	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**30**	92	80	90	157	145	49	93	130	88
					60	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	**51**	532	1553	1396	344	426	183	369	412	316
	25	897	18210	147	30	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	-	247	1289	**56**	66	380	536	86	460	650	380
					60	-	**0.0**	1.7	1.7	1.7	14.9	**0.0**	1.7	77.6	77.6	1.7	-	**1030**	-	-	-	-	2890	-	-	-	-
D19-100	169	897	18094	147	30	-	-	**0.0**	**0.0**	**0.0**	94.5	94.5	11.1	94.5	94.5	**0.0**	-	-	6223	**601**	2734	-	-	-	-	-	901
					60	-	-	4.7	11.9	7.8	6.2	79.0	**3.1**	78.6	78.6	**3.1**	-	-	-	-	-	-	-	-	-	-	-
	274	897	18164	147	30	-	-	6.7	**0.0**	6.3	94.6	94.6	**0.0**	94.6	94.6	**0.0**	-	-	-	**135**	-	-	-	9658	-	-	3259
					60	-	-	**5.1**	6.7	6.8	78.3	78.3	**5.1**	78.3	78.3	**5.1**	-	-	-	-	-	-	-	-	-	-	-
D19-1000	1474	897	17140	147	30	-	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	94.8	**0.0**	-	-	908	**395**	548	904	3280	975	833	-	3694
					60	-	-	**5.8**	8.7	10.1	78.8	7.1	**5.8**	78.8	78.8	**5.8**	-	-	-	-	-	-	-	-	-	-	-
	2296	897	18111	147	30	-	-	6.7	**0.0**	**0.0**	**0.0**	94.8	**0.0**	94.4	94.8	6.7	-	-	-	**185**	1508	2469	-	471	-	-	-
					60	-	-	5.0	5.0	8.4	78.7	78.7	**1.7**	78.7	78.7	3.4	-	-	-	-	-	-	-	-	-	-	-

Table 6.6: Results for QCRDCST problem instances (B: average delay-bound, $|V|, |A|, |R|$: sizes of graph G', Q: quota value in %, η: parameter for upper LP and integer bounds, gap: gap between best primal and dual bound, BP: stabilized branch-and-price, L2: layered graph approach with connection cuts on G', A1: basic ALF variant without cuts, A1G(L): A1 with connection cuts on G' (G'_{L}), A2G(L): improved ALF variant with connection cuts on G' (G'_{L}), best results are printed bold).

interval $[1, 10]$[1]. For each original instance I, three different QCRDCST problem instances I-α have been derived with edge delays d_e, $\forall e \in E$, chosen uniformly at random from $[1, \alpha]$ with $\alpha \in \{10, 100, 1000\}$. The Steiner node with the smallest index is used as root node. Reasonable lower and upper bounds $B_{\min}$ and $B_{\max}$ for B are determined as follows:

$$B_{\min} = \max_{v \in R} \sum_{e \in P^d(s,v)} d_e, \tag{6.18}$$

$$B_{\max} = \max_{v \in R} \sum_{e \in P^c(s,v)} d_e, \tag{6.19}$$

where $P^d(s, v)$ and $P^c(s, v)$ denote the shortest-delay-path and the minimum-cost-path from s to v, respectively. For each instance, we consider two different delay-bounds $B = \lceil B_{\min} + b_{\mathrm{r}} \cdot (B_{\max} - B_{\min}) \rceil$ with $b_{\mathrm{r}} \in \{0.3, 0.6\}$, and for each delay-bound two different quota values $Q = \lceil q_{\mathrm{r}} \cdot \sum_{v \in R} p_v \rceil$ with $q_r \in \{0.3, 0.6\}$. Table 6.6 compares relative optimality gaps between the best primal and dual bounds in percent and runtimes in seconds on the most difficult instances from set QD. In general, the instance graphs are huge but sparse as shown in the first columns of Table 6.6.

One of the most eye-catching results is that BP is not able to provide any reasonable bounds for the considered instances. Here, the memory limit of 3 GB in most cases does not allow to even setup the path model because of the $|R| \cdot |A|$ constraints (4.15) coupling path with arc variables, even though the constraint matrix usually is quite sparse. Note that when increasing the memory limit to 4 GB as done in our article [113] BP can solve some of these instances to optimality. However, also in this case BP's runtimes are significantly higher than those of the ALF approaches.

We selected L2 among the model variants on the full layered graph since separating directed connection cuts on original graph G' turned out to be a reasonable trade-off between adding no more valid inequalities and searching for violated connection cut inequalities on layered graph G'_{L}. Due to the weakness of connection cut variants for the QCRDCST problem in the latter case a huge number of them is added to the model, much more than for the RDCST problem. Altogether, inequalities (6.14) are not able to improve the overall performance, here. However, as we have already seen in previous results for the RDCST problem, approaches on the full layered graph suffer from large MIP models in case of huge delay-bounds. Thus, L2 is superior for extremely tight B values but clearly not competitive for I-100 and I-1000 instances.

For A2G and A2GL approaches setting $\delta = 0.001$ turned out to work best in preliminary tests. Note that this value is one magnitude lower than the δ used for the RDCST problem. Therefore, it seems to be advantageous to spend more effort on approximating the optimal LP value of the full layered graph model before entering the final MIP phase. This can be explained by the fact that by additionally considering node prizes and the quota constraint the integrality gaps between the LP relaxation and the optimal integer value grow compared to the RDCST problem. Thus, it is essential to provide dual bounds as strong as possible to keep the branch-and-bound tree in the final MIP computation small.

Furthermore, it turned out to be disadvantageous to already separate connection cuts on original graph G' in the layered graph extension phase in A2G and A2GL. As already mentioned

[1] `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/steininfo.html`

in Section 6.6.1 the connection cut inequalities for the QCRDCST problem are weaker than the ones for the RDCST problem. Thus, they are not able to contribute that much to the obtained lower LP bounds while the usually vast number of added violated inequalities causes a significant computational overhead.

Note that our initial heuristic originally created for the RDCST problem is not able to consider node prizes and quota constraints and thus simply connects all terminal nodes to the root node. The obtained solution is obviously feasible but may provide a rather bad primal bound. One could argue that this delay-constrained shortest path heuristic could be adapted to the QCRDCST problem with little effort by implementing some greedy decisions based on the node prizes. However, we explicitly want to show the advantage of providing primal bounds by solving MIPs on the reduced layered graphs with redirection to higher layers, without need for problem-specific heuristic methods. This feature enables approaches A2G and A2GL with $\eta = 3$ to reach the best results in most of the cases. A1G and A1GL are able to obtain quite good results, too, mainly due to the same reason as above.

To summarize, when considering the QCRDCST problem on these large sparse graphs ALF significantly outperforms BP and L2, in many cases by orders of magnitude. The secret of its success lies in the fact that it starts with some small amount of approximate information about a given instance and refines it mainly by elements which are helpful as guidance towards optimality. In contrast, BP and L2 begin with all available information even if some of it might be redundant and not necessary for reaching the goal.

6.7 Case Study: Vehicle Routing Problem with Time Windows

In this second case study we examine the application of ALF to a combinatorial optimization problem from a different field of research: the vehicle routing problem with time windows (VRPTW). We will see that it is quite intuitive to use a layered graph approach to solve the VRPTW. Providing such a layered graph transformation ALF can be applied to the problem without modification.

First of all, we formally define the problem: Given a directed graph $G = (V, A)$ with node set V and arc set A. Node 0 denotes the depot and each node $v \in V \setminus \{0\}$ represents a client. Each arc $a \in A$ is assigned a travel cost $c_a \in \mathbb{R}_0^+$ and a travel time $t_a \in \mathbb{Z}^+$. For each client $v \in V \setminus \{0\}$ a demand $q_v \in \mathbb{Z}^+$ and a time window $[a_v, b_v],\ a_v, b_v \in \mathbb{Z}^+,\ a_v \leq b_v$, are given. Furthermore, a fleet of unlimited vehicles with capacity $Q \in \mathbb{Z}^+$ is available at the depot. The objective of the VRPTW is to find a set of routes with minimal total travel costs such that each client is visited exactly once within its time window, each route starts and ends at the depot, and the total demand of clients served within one route must not exceed the vehicle capacity Q. An optional service time at a client is added to the travel times on all outgoing arcs. It is allowed for a vehicle to arrive at a client v before a_v but then has to wait until a_v.

This classical problem together with many variants has been exhaustively tackled by numerous authors in literature in terms of both heuristic and exact approaches. We now only scratch the surface and discuss some recent methods solving the VRPTW to proven optimality. Surveys of state-of-the-art exact approaches are presented by Kallehauge [94, 95] and Desaulniers et al. [40]. Usually, the most successful algorithms use a set partitioning formulation solved by

branch-and-price, extended by several classes of strong valid inequalities. However, the current state-of-the-art method differs in some sense from previous approaches: The quite general framework by Baldacci et al. [9] first computes tight dual solutions by using Lagrangian relaxation and column generation while periodically applying a fast and efficient heuristic to obtain primal bounds. Then, suboptimal routes are eliminated based on their reduced costs with respect to the best primal and dual bounds. In the final phase all residual routes are enumerated and used within a set partitioning formulation extended by so-called subset-row inequalities [91] to determine an optimal solution by a standard MIP solver. Using this approach Baldacci et al. are able to solve all but one instances of the well-known Solomon[2] set and significantly outperform all other methods proposed so far. Letchford et al. [116] provide some projection results for vehicle routing problems and discuss many different sets of strengthening valid inequalities. To the best of the author's knowledge the only layered graph approaches for vehicle routing problems are proposed by Godinho et al. [60, 61] for the capacitated vehicle routing problem with unit demands: Basically, they model the flow to and through each client and limit the vehicle load on paths and circuits on a layered graph, respectively, where the layers correspond to the vehicle loads. By further disaggregating the flow variables they obtain a strong but computationally impractical five-index formulation.

6.7.1 Transformation to Layered Capacity and Time Graphs

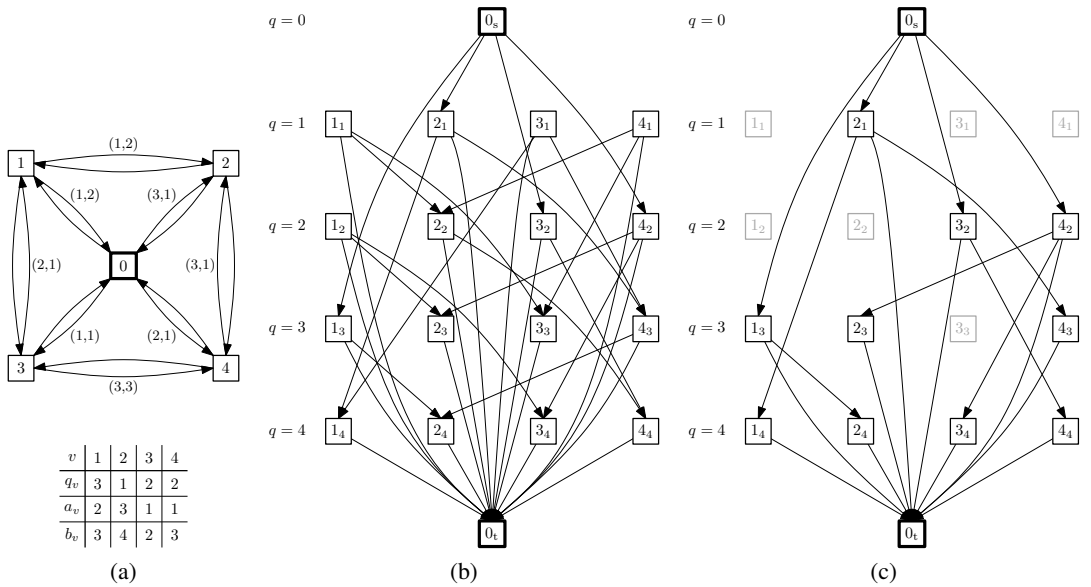
The VRPTW involves two kinds of resource constraints which have to be satisfied: the capacity and time restrictions. Thus, our first layered graph approach models each of the two resource constraints on a separate layered graph. Before we start with the transformation we apply the preprocessing methods described in [94] on graph G: The time windows may induce precedence relations between the clients which enable us to remove infeasible arcs. Additionally, by applying several rules the time windows can possibly be tightened which is quite beneficial for our approach.

Similarly to the layered graph transformation for the RDCST problem in Section 4.8, we transform input graph $G = (V, A)$ to a layered digraph $G_{\mathrm{Lc}} = (V_{\mathrm{Lc}}, A_{\mathrm{Lc}})$ modeling the vehicle capacity restrictions. Node set $V_{\mathrm{Lc}} = \{0_{\mathrm{s}}, 0_{\mathrm{t}}\} \cup \{v_l \mid v \in V \setminus \{0\},\ 1 \leq l \leq Q\}$ consists of depot nodes representing the start and end of a tour and duplicated nodes for all clients for all possible vehicle loads. Node $v_l \in V_{\mathrm{Lc}} \setminus \{0_{\mathrm{s}}, 0_{\mathrm{t}}\}$ denotes the state in which a vehicle arrives at client v having already satisfied a demand of l including demand q_v. Arc set $A_{\mathrm{Lc}} = A^{\mathrm{s}}_{\mathrm{Lc}} \cup A^{\mathrm{g}}_{\mathrm{Lc}} \cup A^{\mathrm{t}}_{\mathrm{Lc}}$ comprises

- start depot arcs $A^{\mathrm{s}}_{\mathrm{Lc}} = \{(0_{\mathrm{s}}, v_{q_v}) \mid (0, v) \in A\}$,
- general arcs $A^{\mathrm{g}}_{\mathrm{Lc}} = \{(u_l, v_{l+q_v} \mid (u, v) \in A,\ u, v \neq 0,\ 1 \leq l \leq Q - q_v\}$, and
- end depot arcs $A^{\mathrm{t}}_{\mathrm{Lc}} = \{(v_l, 0_{\mathrm{t}}) \mid (v, 0) \in A,\ 1 \leq v \leq Q\}$.

To model the time window constraints of the clients we construct a second layered digraph $G_{\mathrm{Lt}} = (V_{\mathrm{Lt}}, A_{\mathrm{Lt}})$. Node set $V_{\mathrm{Lt}} = \{0_{\mathrm{s}}, 0_{\mathrm{t}}\} \cup \{v_l \mid v \in V \setminus \{0\},\ a_v \leq l \leq b_v\}$ consists of start

[2] `http://web.cba.neu.edu/~msolomon/problems.htm`



v	1	2	3	4
q_v	3	1	2	2
a_v	2	3	1	1
b_v	3	4	2	3

(a) (b) (c)

Figure 6.3: (a) Graph G with depot 0, arc labels (c_a, t_a), vehicle capacity $Q = 4$, and table with client demands and time windows. (b) Corresponding layered capacity graph G_{Lc} (arc costs are omitted). (c) Preprocessed graph G_{Lc}.

and end depot nodes and duplicated nodes for all clients for all possible discrete time values within the corresponding time window. Node v_l represents the state in which a vehicle starts servicing a client v at time l. Arc set $A_{\mathrm{Lt}} = A^{\mathrm{s}}_{\mathrm{Lt}} \cup A^{\mathrm{g}}_{\mathrm{Lt}} \cup A^{\mathrm{t}}_{\mathrm{Lt}}$ comprises

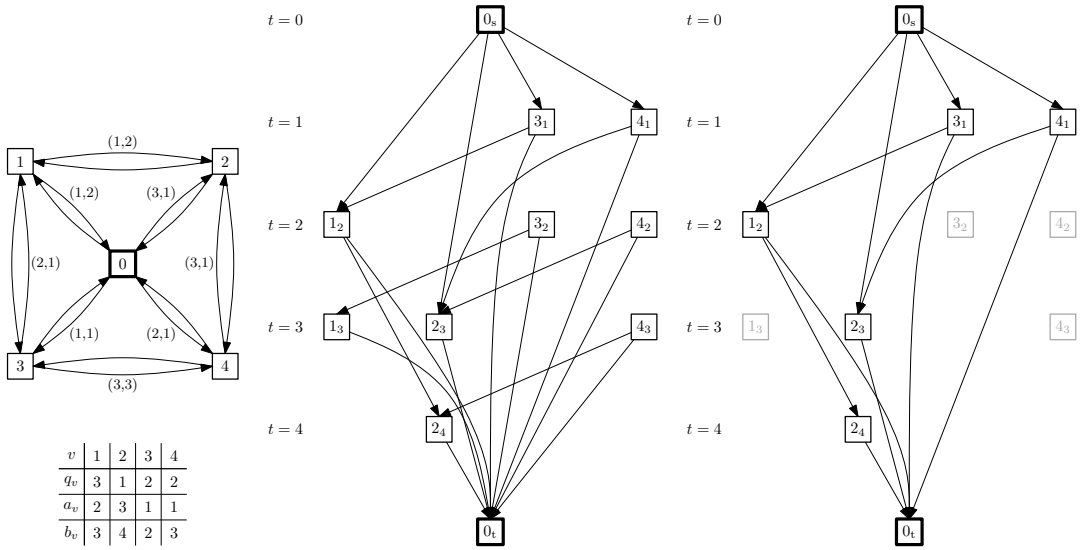
- start depot arcs $A^{\mathrm{s}}_{\mathrm{Lt}} = \{(0_{\mathrm{s}}, v_{\max\{t_{0v}, a_v\}}) \mid (0, v) \in A,\ t_{0v} \leq b_v\}$,
- general arcs $A^{\mathrm{g}}_{\mathrm{Lt}} = \{(u_l, v_{\max\{l+t_{uv}, a_v\}}) \mid (u, v) \in A,\ u, v \neq 0,\ a_u \leq l \leq b_u,\ l + t_{uv} \leq b_v\}$, and
- end depot arcs $A^{\mathrm{t}}_{\mathrm{Lt}} = \{(v_l, 0_{\mathrm{t}}) \mid (v, 0) \in A,\ a_v \leq l \leq b_v\}$.

The preprocessing rules stated in Section 4.8 for layered graphs for the RDCST problem are also valid for G_{Lc} and G_{Lt}. Figures 6.3 and 6.4 illustrate this transformation to G_{Lc} and G_{Lt} for a small example graph G, respectively. The corresponding optimal solution is shown in Fig. 6.5.

6.7.2 MIP Model on Two Layered Graphs

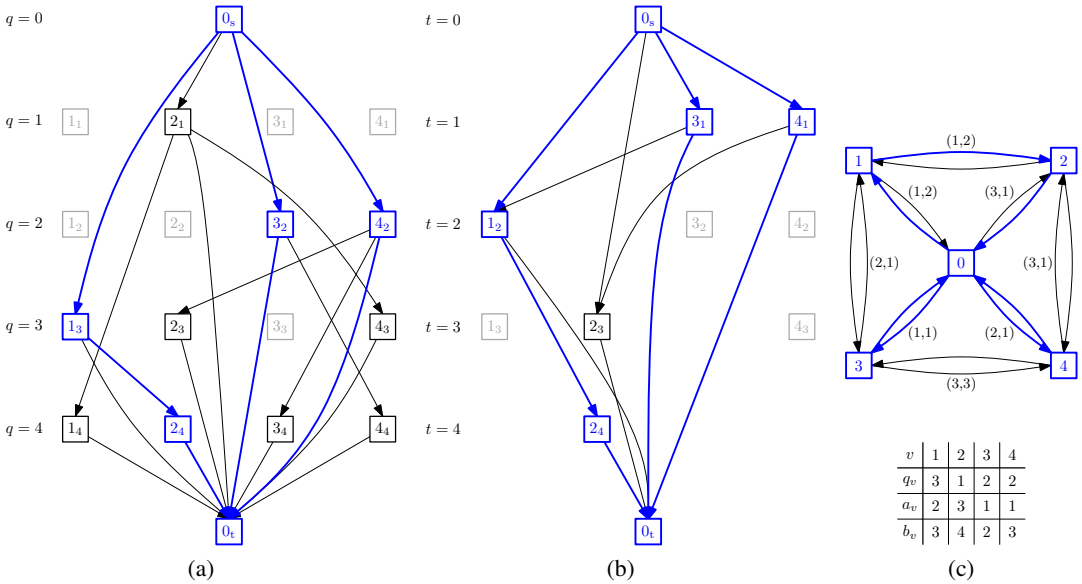
Basically, on each of the two layered graphs G_{Lc} and G_{Lt} we separately model a vehicle routing problem without capacity and time window constraints. Then, we link the two subproblems via the original arc variables.

We use binary variables $x_{uv},\ \forall (u, v) \in A$, for the arcs in the original graph G. Additionally, non-negative variables $\dot{x}^k_{ij}$ and $\ddot{x}^m_{uv}$ represent arcs $(i_k, j_l) \in A_{\mathrm{Lc}}$ and $(u_m, v_n) \in A_{\mathrm{Lt}}$,



v	1	2	3	4
q_v	3	1	2	2
a_v	2	3	1	1
b_v	3	4	2	3

Figure 6.4: (a) Graph G with depot 0, arc labels (c_a, t_a), and table with client demands and time windows. (b) Corresponding layered time graph G_{Lt} (arc costs are omitted). (c) Preprocessed graph G_{Lt}.



v	1	2	3	4
q_v	3	1	2	2
a_v	2	3	1	1
b_v	3	4	2	3

Figure 6.5: (a) Optimal solution denoted by blue arcs in layered capacity graph G_{Lc}, (b) in layered time graph G_{Lt}, and (c) in original graph G.

respectively. Non-negative variables $\dot{y}_u^k$ and $\ddot{y}_v^l$ model layered nodes $u_k \in V_{\mathrm{Lc}}$ and $v_l \in V_{\mathrm{Lt}}$, respectively. Formulation $VRP1$ is defined as follows:

$$\min \quad \sum_{(u,v)\in A} c_{uv} x_{uv} \tag{6.20}$$

$$\text{s.t.} \quad x_{uv} + x_{vu} \leq 1 \qquad \forall (u,v) \in A,\ u,v \neq 0 \tag{6.21}$$

$$\sum_{l} \dot{y}_v^l = 1 \qquad \forall v \in V \setminus \{0\} \tag{6.22}$$

$$\sum_{(u_k,v_l)\in A_{\mathrm{Lc}}} \dot{x}_{uv}^k = \dot{y}_v^l = \sum_{(v_l,w_m)\in A_{\mathrm{Lc}}} \dot{x}_{vw}^l \qquad \forall v_l \in V_{\mathrm{Lc}} \setminus \{0_{\mathrm{s}}, 0_{\mathrm{t}}\} \tag{6.23}$$

$$\sum_{(u_k,v_l)\in A_{\mathrm{Lc}},\ u\neq w} \dot{x}_{uv}^k \geq \dot{x}_{vw}^l \qquad \forall (v_l, w_m) \in A_{\mathrm{Lc}} \tag{6.24}$$

$$\sum_{l} \ddot{y}_v^l = 1 \qquad \forall v \in V \setminus \{0\} \tag{6.25}$$

$$\sum_{(u_k,v_l)\in A_{\mathrm{Lt}}} \ddot{x}_{uv}^k = \ddot{y}_v^l = \sum_{(v_l,w_m)\in A_{\mathrm{Lt}}} \ddot{x}_{vw}^l \qquad \forall v_l \in V_{\mathrm{Lt}} \setminus \{0_{\mathrm{s}}, 0_{\mathrm{t}}\} \tag{6.26}$$

$$\sum_{(u_k,v_l)\in A_{\mathrm{Lt}},\ u\neq w} \ddot{x}_{uv}^k \geq \ddot{x}_{vw}^l \qquad \forall (v_l, w_j) \in A_{\mathrm{Lt}} \tag{6.27}$$

$$\sum_{(u_k,v_l)\in A_{\mathrm{Lc}}} \dot{x}_{uv}^k = \sum_{(u_k,v_l)\in A_{\mathrm{Lt}}} \ddot{x}_{uv}^k = x_{uv} \qquad \forall (u,v) \in A \tag{6.28}$$

$$\dot{x}_{uv}^k \geq 0 \qquad \forall (u_k, v_l) \in A_{\mathrm{Lc}} \tag{6.29}$$

$$\dot{y}_v^l \geq 0 \qquad \forall v_l \in V_{\mathrm{Lc}} \tag{6.30}$$

$$\ddot{x}_{uv}^k \geq 0 \qquad \forall (u_k, v_l) \in A_{\mathrm{Lt}} \tag{6.31}$$

$$\ddot{y}_v^l \geq 0 \qquad \forall v_l \in V_{\mathrm{Lt}} \tag{6.32}$$

$$x_{uv} \in \{0,1\} \qquad \forall (u,v) \in A \tag{6.33}$$

Inequalities (6.21) forbid short cycles between two nodes with one exception: A vehicle is allowed to start at the depot, visit only one client and return again to the depot. Equalities (6.22) and (6.25) select exactly one layered graph node in G_{Lc} and G_{Lt} corresponding to one particular client, respectively. The flow conservation for vehicles is guaranteed by equalities (6.23) and (6.26). According to [61], inequalities (6.24) and (6.27) which are the counterparts of (4.42) for the RDCST problem are able to strengthen the model. Finally, equalities (6.28) link arcs on G_{Lc} and G_{Lt}, respectively, to arcs on the original graph G. In principle, these coupling constraints make it possible to eliminate x_{uv} variables from the model if we declare $\dot{x}_{uv}^k$ and $\ddot{x}_{uv}^k$ Boolean. However, as noted in Section 4.9 for the RDCST problem, x_{uv} variables can be beneficial for branching. Note that the classical assignment constraints are implicitly contained in the model by combining equalities (6.22), (6.23), (6.25), (6.26), and (6.28).

Many sets of strengthening inequalities for different MIP formulations for the vehicle routing problem have been proposed in the past, among them several sets of capacity inequalities [116,

132] which are stronger variants of the directed connection cut inequalities (4.27):

$$\sum_{(u,v)\in A,\ u\notin W,\ v\in W} x_{uv} \geq k(W) \qquad \forall W \subset V,\ 0 \notin W, \tag{6.34}$$

where $k(W)$ denotes the minimum number of vehicles necessary to serve all clients in node set W. To find an optimal value for $k(W)$ one has to solve a one-dimensional bin packing problem [52] which nowadays can be done quite efficiently even for a huge number of items. However, as described in [132] often lower bounds for k are used, such as

$$k(W) \geq \lceil \sum_{v\in W} q_v/Q \rceil \qquad \forall W \subseteq V,\ W \neq \emptyset. \tag{6.35}$$

Clearly, the higher the right side of inequalities (6.34) the stronger the inequality. Thus, a lower bound for k is valid but weaker than the optimal value. The right side can even be lifted by also considering the clients outside set W. However, the higher the right side the more difficult the corresponding separation problem, cf. [132]. Since not even the directed connection cut inequalities are contained in the description $VRP1$ all these capacity inequalities are able to improve the according LP bound. It is still open and part of future work which of the other known sets of valid inequalities are already included in $VRP1$ and which can possibly further tighten it.

6.7.3 Transformation to Layered Capacity-Time Graph

Note that each of the two layered graphs models a set of solutions which does not respect the other resource constraint. Thus, it seems to be quite natural to find a way to combine both layered graphs. Our approach is based on extending the layered capacity graph G_{Lc} by an additional dimension to allow a simultaneous consideration of the time restrictions. This results in a tuple (v, q, t) defining one specific layered graph node: a vehicle visits client v with load q (including q_v) at time t which is a quite natural description of a vehicle state.

We transform graph $G = (V, A)$ to a layered capacity-time digraph $G_{\mathrm{L}} = (V_{\mathrm{L}}, A_{\mathrm{L}})$. Node set $V_{\mathrm{L}} = \{0_{\mathrm{s}}, 0_{\mathrm{t}}\} \cup \{v_{qt} \mid v \in V \setminus \{0\},\ 1 \leq q \leq Q,\ a_v \leq t \leq b_v\}$ consists of start and end depot nodes and duplicated nodes for all clients for all possible vehicle loads and time values within the corresponding time window. Arc set $A_{\mathrm{L}} = A_{\mathrm{L}}^{\mathrm{s}} \cup A_{\mathrm{L}}^{\mathrm{g}} \cup A_{\mathrm{L}}^{\mathrm{t}}$ includes

- start depot arcs $A_{\mathrm{L}}^{\mathrm{s}} = \{(0_{\mathrm{s}}, v_{q_v,\max\{t_{0v},a_v\}}) \mid (0, v) \in A,\ t_{0v} \leq b_v\}$,
- general arcs $A_{\mathrm{L}}^{\mathrm{g}} = \{(u_{qt}, v_{q+q_v,\max\{t+t_{uv},a_v\}}) \mid (u, v) \in A,\ u, v \neq 0,\ 1 \leq q \leq Q - q_v,\ a_u \leq t \leq b_u,\ t + t_{uv} \leq b_v\}$, and
- end depot arcs $A_{\mathrm{Lt}} = \{(v_{qt}, 0_{\mathrm{t}}) \mid (v, 0) \in A,\ 1 \leq q \leq Q,\ a_v \leq t \leq b_v\}$.

Based on Fig. 6.3 and 6.4 it is not hard to imagine an according combined three-dimensional layered graph G_{L}. Note that an arc always crosses both the capacity and time dimension since all travel times and demand values are positive.

6.7.4 MIP Model on the Combined Layered Graph

An according MIP model on layered graph G_{L} can be written similarly as $VRP1$. We use binary variables $x_{uv},\ \forall (u,v) \in A$, to model arcs in graph G. Additionally, non-negative variables x_{uv}^{qt} represent arcs $(u_{qt}, v_{kl}) \in A_{\mathrm{L}}$, and variables y_v^{qt} model layered nodes $v_{qt} \in V_{\mathrm{L}}$. Formulation $VRP2$ is defined as follows:

$$\min \quad \sum_{(u,v)\in A} c_{uv} x_{uv} \tag{6.36}$$

$$\text{s.t.} \quad x_{uv} + x_{vu} \leq 1 \qquad \forall (u,v) \in A,\ u,v \neq 0 \tag{6.37}$$

$$\sum_{q,t} y_v^{qt} = 1 \qquad \forall v \in V \setminus \{0\} \tag{6.38}$$

$$\sum_{(u_{kl},v_{qt})\in A_{\mathrm{L}}} x_{uv}^{kl} = y_v^{qt} = \sum_{(v_{qt},w_{mn})\in A_{\mathrm{L}}} x_{vw}^{qt} \qquad \forall v_{qt} \in V_{\mathrm{L}} \setminus \{0_{\mathrm{s}}, 0_{\mathrm{t}}\} \tag{6.39}$$

$$\sum_{(u_{kl},v_{qt})\in A_{\mathrm{L}},\ u\neq w} x_{uv}^{kl} \geq x_{vw}^{qt} \qquad \forall (v_{qt}, w_{mn}) \in A_{\mathrm{L}} \tag{6.40}$$

$$\sum_{(u_{kl},v_{qt})\in A_{\mathrm{L}}} x_{uv}^{kl} = x_{uv} \qquad \forall (u,v) \in A \tag{6.41}$$

$$x_{uv}^{kl} \geq 0 \qquad \forall (u_{kl}, v_{qt}) \in A_{\mathrm{L}} \tag{6.42}$$

$$y_v^{qt} \geq 0 \qquad \forall v_{qt} \in V_{\mathrm{L}} \tag{6.43}$$

$$x_{uv} \in \{0,1\} \qquad \forall (u,v) \in A \tag{6.44}$$

This model is a straight-forward adaption of model $VRP1$. It is easy to see that the optimal LP relaxation value of model $VRP2$ is at least as high as the one of model $VRP1$, and there are instances where it is higher.

6.7.5 ALF for the VRPTW

Especially the three-dimensional layered graph G_{L} but also the two independent graphs G_{Lc} and G_{Lt} may involve excessively many nodes and arcs. Thus, ALF seems to be a natural extension to the approaches on the full layered graphs. In fact ALF can be applied without modification to the VRPTW. When considering formulation $VRP1$ as base model for ALF then we can redirect arcs independently in G_{Lc} and G_{Lt} provided that we use the same direction – to lower or to higher layers. In the combined layered graph G_{L} we have to ensure that an arc is redirected to the same direction for both resource dimensions otherwise we are not able to say whether the obtained bound is a lower or upper one.

The infeasible path inequalities needed in the final MIP phase of the improved ALF variant in Algorithm 6.2 can be strengthened by utilizing the fact that also the out-degree of a client node is exactly one. Kallehauge et al. [96] apply e.g. so-called tournament inequalities and further variants to the VRPTW. When redirecting arcs to lower layers we already know that the according layered graph is not acyclic anymore. Thus, the following inequality which in fact is a capacity inequality (6.34) with $W = V \setminus \{0\}$ and redundant for the full layered graph model

strengthens the obtained lower LP bounds:

$$\sum_{(0,v)\in A} x_{0v} \geq k(V \setminus \{0\}). \tag{6.45}$$

Basically, it ensures that enough vehicles leave the depot to serve all client demands.

6.7.6 Preliminary Results

Until now we only performed some preliminary tests as proof-of-concept on the Solomon instances which comprise complete graphs with 25-100 nodes, differently clustered clients and several time window widths. The state-of-the-art framework by Baldacci et al. [9] is able to solve all instances except the R208 graph with 100 nodes to optimality. Our preliminary results are far away from this performance but nevertheless look promising. We observed large integrality gaps between the LP relaxation value of models $VRP1$ or even $VRP2$ and the optimal integer value. This indicates a need for using additional strengthening valid inequalities from the literature since we currently only separate the simple directed connection cut inequalities (4.27) and tournament inequalities to prevent infeasible paths. The next step will be to possibly derive new stronger variants of existing sets of valid inequalities utilizing the more refined structure of layered graphs.

When comparing formulations $VRP1$ and $VRP2$ on the full layered graph the approach on two separated graphs outperforms the model on the combined graph in nearly all cases. Here, the three-dimensional layered graph mostly is too large to be tractable within reasonable time and memory limits. However, when ALF is applied the size of the layered graphs is not crucial anymore and thus the approach on the tighter model $VRP2$ is far superior to $VRP1$. In general, ALF provides significantly better results than solving the full layered graph models.

6.8 Future Work

The unexpected synergy effect between the reduced layered graphs when redirecting to lower and higher layers as discussed in Section 6.5.2 is definitely worth to be further analyzed. Also efficient primal heuristics could accelerate the detection of good primal bounds. A further promising method to reduce graph sizes is preprocessing based on reduced costs, cf. [111]. Since we usually have quite tight gaps, even within the first layered graph extension phase of the improved ALF variant, we might be able to utilize them to eliminate or fix nodes and arcs in original graph G and layered graph G_{L}, respectively.

As already mentioned in Section 6.3, ALF cannot be applied in a straight-forward way to the RDDVCST problem because of the additional delay-variation-constraint relating resources on different paths. However, if we decompose the problem as described in Section 5.9 by considering each possible variation interval independently then we are able to use ALF to solve these subproblems since basically we reduce the RDDVCST problem to a set of RDCST problems. On the other hand, if we find a way to implicitly encode the delay-variation-constraint in the structure of a layered graph then ALF may also work. But whether and how such a layered graph construction can be achieved, respectively, is still an open question.

We observed especially for the VRPTW problem that for some instances even the reduced layered graph becomes too large. Thus, we could think about a meaningful way to again reduce the layered graph node set, e.g. by removing all nodes which have rarely been included in a fractional or integer solution. However, at the same time we would lose the guarantee of monotonically improving bounds. Or is it maybe possible to efficiently decide which layered graph nodes have become redundant again within the last iterations?

Clearly, a hot topic is to find new state-of-the-art exact approaches for vehicle routing problems and maybe solve the last open Solomon instance for the VRPTW. Obviously, the so far leading extremely sophisticated methods are hard to outperform. Maybe ALF is a possible direction to reach this goal. Especially multi-dimensional layered graphs which simultaneously consider several resources in combination with ALF to deal with their huge sizes seem to be quite promising.

Last but not least, we plan to adapt and apply ALF to further optimization problems and to analyze its behavior. The so far performed tests indicate that our framework particularly is successful on large sparse graphs which are quite common in practical applications in network design and routing.

CHAPTER 7

Conclusions

In this thesis we considered several combinatorial optimization problems in the area of network design where different QoS constraints have to be satisfied. Basically, these limitations concern the total delays on the paths from a central server to a set of connected clients. They define upper bounds and restrict the variation between different paths. We tackled these combinatorial optimization problems by applying various (meta-)heuristics as well as exact approaches based on (mixed) integer programming.

We introduced construction heuristics based on Kruskal's minimum spanning tree algorithm and on the multilevel refinement paradigm for the rooted delay-constrained minimum spanning tree problem. On average the Kruskal-based heuristic avoiding some drawbacks of previous heuristics produces faster and better results compared to other approaches especially for tight delay-bounds. The runtime is almost independent of the delay-constraint and the cost- and delay-values of the instances. On the contrary, the multilevel heuristic is in general a better starting point for subsequent improvement by the presented local search techniques. Among the applied metaheuristics the general variable neighborhood search and the genetic algorithm outperform the other methods in terms of solution quality when considering a fixed time limit. The first is able to quickly find high quality solutions whereas the second with its balanced mix of diversification and intensification built on a fast and diverse solution construction usually needs some time to converge but then yields even better solutions in many cases. Both approaches benefit from an embedded efficient variable neighborhood descent in two sophisticated neighborhood structures. Additionally, we discussed methods to detect duplicates in the genetic algorithm by either solution hashing or a complete trie-based archive. Hashing works well and is able to improve final solution quality, and in contrast to the solution archive the time overhead is negligible. The trie-based archive can be beneficial for instances with low delay-bounds and/or if the number of revisits is very high then providing new unvisited solutions.

For solving the rooted delay-constrained Steiner tree problem we presented various exact modeling approaches based on integer programming. A path-cut formulation with a small number of variables but an exponentially-sized set of directed connection cut and strengthened infeasible path inequalities stays quite compact during an according branch-and-cut algorithm but

in general examines a rather high number of branch-and-bound nodes due to its weaker LP relaxation bounds. However, the independence of given delay-bounds and the light-weight model are beneficial for several types of instances. A branch-and-price approach stabilized by alternative dual-optimal solutions is quite robust and well-performing throughout all instances sets making it often the preferred choice. However, a transformation to an appropriate layered graph and a MIP model utilizing the special structure of this graph, mainly its acyclicity, provides the strongest dual bounds and excellent results in many cases except on instances with huge sets of achievable path delay values and high bounds since the size of the layered graph heavily depends on these properties.

We tackled the rooted delay- and delay-variation-constrained Steiner tree problem by using two different MIP models based on multi-commodity-flows and a layered graph transformation. Furthermore, we proposed sets of valid inequalities for the second model particularly targeting the bounding of the delay-variation and provided an efficient separation method. Experimental results clearly show the superiority of layered graph models with or without delay-variation cuts. Nevertheless, the generally still relatively large integrality gaps of the layered graph models ask for investigating also other modeling approaches. We believe that finding a way to implicitly encode the delay-variation-constraint within a layered graph structure is highly promising to further close the gap.

Our adaptive layers framework (ALF) tries to overcome the problems with huge layered graphs by computing lower and upper bounds to the optimal LP and integer value on reduced layered graphs. It obtains in general rather small gaps and shows robust performance throughout all test sets. Besides consuming significantly less memory it even yields tight bounds in cases where it is not possible to compute LP relaxations of the model on the full layered graph in reasonable time. Only on complete graphs and for low delay-bounds ALF produces too much overhead due to repeated model solving. On large sparse graphs ALF is able to dramatically outperform all other approaches, sometimes by orders of magnitude. This is also observed in a case study on a different network design problem considering node prizes and a quota constraint. A second case study documents ALF's versatility by applying it to the vehicle routing problem with time windows for which two promising alternative modeling approaches on layered graphs are suggested.

Bibliography

[1] E. H. L. Aarts and J. K. Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[2] T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[3] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[4] R. K. Ahuja, O. Ergun, J. B. Orlin, and A. P. Punnen. A Survey of Very Large-Scale Neighborhood Search Techniques. *Discrete Applied Mathematics*, 123(1-3):75–102, 2002.

[5] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[6] M. Aissa and A. B. Mnaouer. A new delay-constrained algorithm for multicast routing tree construction. *International Journal of Communication Systems*, 17(10):985–1000, 2004.

[7] N. Ascheuer, M. Fischetti, and M. Grötschel. A polyhedral study of the asymmetric traveling salesman problem with time windows. *Networks*, 36(2):69–79, 2000.

[8] N. Ascheuer, M. Fischetti, and M. Grötschel. Solving the Asymmetric Travelling Salesman Problem with time windows by branch-and-cut. *Mathematical Programming*, 90(3):475–506, 2001.

[9] R. Baldacci, E. Bartolini, A. Mingozzi, and R. Roberti. An exact solution framework for a broad class of vehicle routing problems. *Computational Management Science*, 7(3):229–268, 2010.

[10] S. M. Banik, S. Radhakrishnan, and C. N. Sekharan. Multicast Routing with Delay and Delay Variation Constraints for Collaborative Applications on Overlay Networks. *IEEE Transactions on Parallel and Distributed Systems*, 18(3):421–431, 2007.

[11] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-And-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, 46(3):316–329, 1998.

[12] J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

[13] J. E. Beasley. A heuristic for Euclidean and rectilinear Steiner problems. *European Journal of Operational Research*, 58(2):284–292, 1992.

[14] M. Berlakovich. Multilevel Heuristiken für das Rooted Delay-Constrained Minimum Spanning Tree Problem. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Vienna, Austria, July 2010.

[15] M. Berlakovich, M. Ruthmair, and G. R. Raidl. A Multilevel Heuristic for the Rooted Delay-Constrained Minimum Spanning Tree Problem. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I*, volume 6927 of *LNCS*, pages 256–263. Springer, 2012.

[16] D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98(1):49–71, 2003.

[17] D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[18] D. Bertsimas and A. Thiele. Robust and data-driven optimization: modern decision-making under uncertainty. In *INFORMS Tutorials in Operations Research: Models, Methods, and Applications for Innovative Decision Making*. INFORMS, 2006.

[19] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[20] D. Bertsimas and R. Weismantel. *Optimization over Integers*. Dynamic Ideas, 2005.

[21] C. Blum, M. J. B. Aquilera, A. Roli, and M. Sampels, editors. *Hybrid Metaheuristics: An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence (SCI)*. Springer, 2008.

[22] C. Blum and A. Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[23] C. Blum and A. Roli. Hybrid Metaheuristics: An Introduction. In C. Blum, M. J. B. Aquilera, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics: An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence (SCI)*, pages 1–30. Springer, 2008.

[24] S. A. Canuto, M. G. C. Resende, and C. C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 38(1):50–58, 2001.

[25] B. V. Cherkassky and A. V. Goldberg. On Implementing the Push-Relabel Method for the Maximum Flow Problem. *Algorithmica*, 19(4):390–410, 1997.

[26] S. Chopra, E. R. Gorres, and M. R. Rao. Solving the Steiner tree problem on a graph using branch and cut. *ORSA Journal on Computing*, 4(3):320–335, 1992.

[27] S. Chopra and M. Rao. The Steiner tree problem I: Formulations, compositions and extension of facets. *Mathematical Programming*, 64(1):209–229, 1994.

[28] W. Chung. Dantzig-Wolfe Decomposition. In *Wiley Encyclopedia of Operations Research and Management Science*. Wiley, 2010.

[29] Cisco Systems. Cisco Visual Networking Index: Forecast and Methodology, 2010-2015. Technical report, Cisco Systems, 2011.

[30] R. K. Congram, C. N. Pots, and S. L. van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.

[31] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms second edition*. The MIT Press, 2nd edition, 2001.

[32] G. Dahl, L. Gouveia, and C. Requejo. On formulations and methods for the hop-constrained minimum spanning tree problem. In *Handbook of Optimization in Telecommunications*, chapter 19, pages 493–515. Springer, 2006.

[33] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T. C. Koppmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347. Wiley, 1951.

[34] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1998.

[35] G. B. Dantzig and P. Wolfe. The decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

[36] C. Darwin. *On the origin of species by means of natural selection of the preservation of favored races in the struggle for life*. Murray, 1859.

[37] S. Dash, O. Günlük, A. Lodi, and A. Tramontani. A Time Bucket Formulation for the Traveling Salesman Problem with Time Windows. *INFORMS Journal on Computing*, 24(1):132–147, 2010.

[38] K. Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. Wiley, 2001.

[39] G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors. *Column Generation*. Springer, 2005.

[40] G. Desaulniers, J. Desrosiers, and S. Spoorendonk. The Vehicle Routing Problem with Time Windows: State-of-the-Art Exact Solution Methods. In *Wiley Encyclopedia of Operations Research and Management Science*, pages 1–11. Wiley, 2010.

[41] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[42] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(1):29–41, 1996.

[43] M. Dorigo and T. Stützle. Ant Colony Optimization: Overview and Recent Advances. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 227–263. Springer, 2010.

[44] S. Dreyfus and R. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1972.

[45] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct methods for sparse matrices*. Oxford University Press, USA, 1989.

[46] I. Dumitrescu and N. Boland. Improved preprocessing, labeling and scaling algorithms for the Weight-Constrained Shortest Path Problem. *Networks*, 42(3):135–153, Aug. 2003.

[47] T. Feo and M. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.

[48] T. A. Feo and M. G. C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[49] M. Fernandes, L. Gouveia, and S. Voß. Determining hop-constrained spanning trees with repetitive heuristics. *Journal of Telecommunications and Information Technology*, 4:16–22, 2007.

[50] M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management science*, 50(12):1861–1871, 2004.

[51] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[52] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

[53] M. Gendreau and J. Y. Potvin, editors. *Handbook of Metaheuristics*, volume 146. Springer, 2nd edition, 2010.

[54] A. M. Geoffrion. Generalized benders decomposition. *Journal of optimization theory and applications*, 10(4):237–260, 1972.

[55] N. Ghaboosi and A. T. Haghighat. A path relinking approach for Delay-Constrained Least-Cost Multicast routing problem. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, volume 34, pages 383–390. IEEE Computer Society, May 2007.

[56] E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

[57] P. C. Gilmore and R. E. Gomory. A Linear Programming Approach to the Cutting-Stock Problem (Part I). *Operations Research*, 9:849–859, 1961.

[58] P. C. Gilmore and R. E. Gomory. A Linear Programming Approach to the Cutting-Stock Problem (Part II). *Operations Research*, 11:363–888, 1963.

[59] F. Glover and G. A. Kochenberger. *Handbook of metaheuristics*. Springer, 2003.

[60] M. T. Godinho, L. Gouveia, and T. L. Magnanti. Combined route capacity and route length models for unit demand vehicle routing problems. *Discrete Optimization*, 5(2):350–372, 2008.

[61] M. T. Godinho, L. Gouveia, T. L. Magnanti, P. Pesneau, and J. Pires. Inequalities Implied by a Time-Dependent Model for the Unit Demand Vehicle Routing Problem. Technical report, Technical Report 10, Centro de Investigacão Operacional, Faculdade de Ciências da Universidade de Lisboa, 2009.

[62] M. X. Goemans and Y.-S. Myung. A catalog of Steiner tree formulations. *Networks*, 23(1):19–28, 1993.

[63] J. Gottlieb, B. A. Julstrom, G. R. Raidl, and F. Rothlauf. Prüfer numbers: A poor representation of spanning trees for evolutionary search. In S. et al. L., editor, *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, pages 343–350. Morgan Kaufmann, 2001.

[64] L. Gouveia. Using the Miller-Tucker-Zemlin constraints to formulate a minimal spanning tree problem with hop constraints. *Computers & Operations Research*, 22(9):959–970, 1995.

[65] L. Gouveia. Multicommodity flow models for spanning trees with hop constraints. *European Journal of Operational Research*, 95(1):178–190, 1996.

[66] L. Gouveia. Using Variable Redefinition for Computing Lower Bounds for Minimum Spanning and Steiner Trees with Hop Constraints. *Informs Journal on Computing*, 10(2):180–188, 1998.

[67] L. Gouveia. Using hop-indexed models for constrained spanning and Steiner tree models. In B. Sanso and P. Soriano, editors, *Telecommunications Network Planning*, pages 21–32. Kluwer Academic Publishers, 1999.

[68] L. Gouveia, A. Paias, and D. Sharma. Modeling and solving the rooted distance-constrained minimum spanning tree problem. *Computers & Operations Research*, 35(2):600–613, 2008.

[69] L. Gouveia, P. Patrício, and A. De Sousa. Lexicographical minimization of routing hops in telecommunication networks. In *Proceedings of the International Network Optimization Conference*, pages 216–229. Springer, 2011.

[70] L. Gouveia, L. G. Simonetti, and E. Uchoa. Modeling hop-constrained and diameter-constrained minimum spanning tree problems as Steiner tree problems over layered graphs. *Mathematical Programming*, 128(1):123–148, 2011.

[71] C. Gruber. Ein Lösungsarchiv mit Branch-and-Bound-Erweiterung für das Generalized Minimum Spanning Tree Problem. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Sept. 2011.

[72] M. Gruber. *Exact and Heuristic Approaches for Solving the Bounded Diameter Minimum Spanning Tree Problem*. PhD thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, May 2009.

[73] M. Gruber and G. R. Raidl. Exploiting Hierarchical Clustering for Finding Bounded Diameter Minimum Spanning Trees on Euclidean Instances. In F. Rothlauf, editor, *GECCO 2009: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 263–270. ACM Press, 2009.

[74] M. Gruber and G. R. Raidl. (Meta-)Heuristic Separation of Jump Cuts in a Branch&Cut Approach for the Bounded Diameter Minimum Spanning Tree Problem. In V. Maniezzo, T. Stützle, and S. Voß, editors, *Matheuristics – Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*, pages 209–230. Springer, 2009.

[75] M. Gruber, J. van Hemert, and G. R. Raidl. Neighborhood Searches for the Bounded Diameter Minimum Spanning Tree Problem Embedded in a VNS, EA, and ACO. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1187–1194, New York, New York, USA, 2006. ACM Press.

[76] L. Guo and I. Matta. QDMR: an efficient QoS dependent multicast routing algorithm. In *Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium*, pages 213–222. IEEE, 1999.

[77] B. K. Haberman and G. N. Rouskas. Cost, delay, and delay variation conscious multicast routing. Technical report, North Carolina State University, 1996.

[78] P. Hansen and N. Mladenovic. An introduction to variable neighborhood search. In S. Voß, S. Martello, I. H. Osman, and C. Roucairol, editors, *MIC-97: meta-heuristics international conference*, pages 433–458. Kluwer Academic Publishers, 1999.

[79] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[80] P. Hansen, N. Mladenović, J. Brimberg, and J. A. M. Pérez. Variable Neighborhood Search. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 61–86. Springer, 2010.

[81] M. Haouari, S. B. Jayeb, and H. D. Sherali. The prize collecting Steiner tree problem: Models and Lagrangian dual optimization approaches. *Computational Optimization and Applications*, 40(1):13–39, 2008.

[82] M. Haouari, S. B. Layeb, and H. D. Sherali. Strength of Three MIP Formulations for the Prize Collecting Steiner Tree Problem with a Quota Constraint. *Electronic Notes in Discrete Mathematics*, 36:495–502, 2010.

[83] M. Haouari and J. C. Siala. A hybrid Lagrangian genetic algorithm for the prize collecting Steiner tree problem. *Computers & Operations Research*, 33:1274–1288, 2006.

[84] D. P. Heyman and M. J. Sobel. *Stochastic models in operations research, Vol. II: Stochastic optimization*, volume 2. Dover Publications, 2003.

[85] J. H. Holland. *Adaptation in natural and artificial systems*. MIT Press, 1992.

[86] T.-S. Hsu, K.-H. Tsai, D.-W. Wang, and D. T. Lee. Steiner Problems on Directed Acyclic Graphs. *Computing and Combinatorics*, pages 21–30, 1996.

[87] B. Hu, M. Leitner, and G. R. Raidl. Combining Variable Neighborhood Search with Integer Linear Programming for the Generalized Minimum Spanning Tree Problem. *Journal of Heuristics*, 14(5):473–499, 2008.

[88] B. Hu and G. R. Raidl. Variable Neighborhood Descent with Self-Adaptive Neighborhood-Ordering. In C. Cotta, A. J. Fernandez, and J. E. Gallardo, editors, *Proceedings of the 7th EU/MEeting on Adaptive, Self-Adaptive, and Multi-Level Metaheuristics*, Malaga, Spain, 2006.

[89] F. K. Hwang and D. S. Richards. Steiner tree problems. *Networks*, 22(1):55–89, 1992.

[90] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner tree problem*. North Holland, 1992.

[91] M. Jepsen, B. Petersen, S. Spoorendonk, and D. Pisinger. Subset-Row Inequalities Applied to the Vehicle-Routing Problem with Time Windows. *Operations Research*, 56(2):497–511, 2008.

[92] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, 9(3):256–278, 1974.

[93] D. S. Johnson, M. Minkoff, and S. Phillips. The prize collecting steiner tree problem: theory and practice. In *Proceedings of the 11th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 760–769. Society for Industrial and Applied Mathematics, 2000.

[94] B. Kallehauge. *On the vehicle routing problem with time windows*. PhD thesis, Technical University of Denmark, Informatics and Mathematical Modelling, Scientific Computing, 2006.

[95] B. Kallehauge. Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers & Operations Research*, 35(7):2307–2330, 2008.

[96] B. Kallehauge, N. Boland, and O. B. G. Madsen. Path inequalities for the vehicle routing problem with time windows. *Networks*, 49(4):273–293, 2007.

[97] N. Karmakar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[98] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[99] L. Khachiyan. A polynomial algorithm in linear programming (english translation). *Soviet Mathematics Doklady*, 20:191–194, 1979.

[100] T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32(3):207–232, 1998.

[101] T. Koch, A. Martin, and S. Voß. SteinLib: An updated library on Steiner tree problems in graphs, 2001.

[102] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos. Multicasting for multimedia applications. In *Proceedings of the 11th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOMM)*, volume 3, pages 2078–2085. IEEE, 1992.

[103] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking*, 1(3):286–292, 1993.

[104] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos. Two Distributed Algorithms for Multicasting Multimedia Information. In *Proceedings of the 2nd International Conference on Computer Communications and Networks (ICCCN)*, pages 343–349, San Diego, CA, USA, 1993.

[105] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15(2):141–145, 1981.

[106] J. Kratica. Improving Performances of the Genetic Algorithm by Caching. *Computers and Artificial Intelligence*, 18(3):271–283, 1999.

[107] J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[108] Z. Kun, W. Heng, and L. Feng-yu. Distributed multicast routing for delay and delay variation-bounded Steiner tree using simulated annealing. *Computer Communications*, 28(11):1356–1370, 2005.

[109] H.-Y. Lee and C.-H. Youn. Scalable multicast routing algorithm for delay-variation constrained minimum-cost tree. In *IEEE International Conference on Communications*, volume 3, pages 1343–1347. IEEE Press, 2000.

[110] V. Leggieri, M. Haouari, and C. Triki. An Exact Algorithm for the Steiner Tree Problem with Delays. In *Electronic Notes in Discrete Mathematics*, volume 36, pages 223–230. Elsevier, 2010.

[111] V. Leggieri, M. Haouari, and C. Triki. The Steiner Tree Problem with Delays: A compact formulation and reduction procedures. *Discrete Applied Mathematics*, 2011.

[112] M. Leitner. *Solving Two Network Design Problems by Mixed Integer Programming and Hybrid Optimization Methods*. PhD thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Vienna, Austria, May 2010.

[113] M. Leitner, M. Ruthmair, and G. R. Raidl. On Stabilized Branch-and-Price for Constrained Tree Problems. Technical Report TR 186–1–11–01, Vienna University of Technology, Vienna, Austria, 2011. accepted with revisions to Networks (INOC 2011 special issue).

[114] M. Leitner, M. Ruthmair, and G. R. Raidl. Stabilized Branch-and-Price for the Rooted Delay-Constrained Steiner Tree Problem. In J. Pahl, T. Reiners, and S. Voß, editors, *Network Optimization: 5th International Conference, INOC 2011*, volume 6701 of *LNCS*, pages 124–138, Hamburg, Germany, 2011. Springer.

[115] M. Leitner, M. Ruthmair, and G. R. Raidl. Stabilized Column Generation for the Rooted Delay-Constrained Steiner Tree Problem. In *Proceedings of the VII ALIO/EURO – Workshop on Applied Combinatorial Optimization*, pages 250–253, Porto, Portugal, 2011.

[116] A. N. Letchford and J.-J. Salazar-González. Projection results for vehicle routing. *Mathematical Programming*, 105(2):251–274, 2006.

[117] I. Ljubic and S. Gollowitzer. Layered Graph Approaches to the Hop Constrained Connected Facility Location Problem. Technical report, University of Vienna, Austria, 2010.

[118] I. Ljubic and S. Gollowitzer. Modelling the hop constrained connected facility location problem on layered graphs. In *Electronic Notes in Discrete Mathematics*, volume 36, pages 207–214. Elsevier, 2010.

[119] I. Ljubic, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Mathematical Programming*, 105(2):427–449, 2006.

[120] A. Lodi. Mixed integer programming computation. *50 Years of Integer Programming 1958-2008*, pages 619–645, 2010.

[121] C. P. Low and Y. J. Lee. Distributed multicast routing, with end-to-end delay and delay variation constraints. *Computer Communications*, 23(9):848–862, 2000.

[122] M. E. Lübbecke and J. Desrosiers. Selected Topics in Column Generation. *Operations Research*, 53(6):1007–1023, 2005.

[123] N. Maculan. The Steiner tree problem in graphs. *Annals of Discrete Mathematics*, 31:185–212, 1987.

[124] T. L. Magnanti and L. A. Wolsey. Optimal trees. *Handbooks in operations research and management science*, 7:503–615, 1995.

[125] V. Maniezzo, T. Stützle, and S. Voß, editors. *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*. Springer, 2009.

[126] P. Manyem and M. F. M. Stallmann. Some approximation results in multicasting. Technical Report TR-96-03, North Carolina State University, 1996.

[127] G. Mendel. Versuche über Pflanzen-Hybriden (Experiments on plant hybridization). In *Verhandlungen des naturforschenden Vereins Brünn (Proceedings of the Natural History Society of Brünn)*, volume 4, pages 3–47, 1866.

[128] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*, volume 19. Springer, 1992.

[129] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.

[130] P. Moscato. Memetic Algorithms: A Short Introduction. In D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price, editors, *New Ideas in Optimization*, pages 219–234. McGraw Hill, 1999.

[131] P. Moscato and C. Cotta. A Modern Introduction to Memetic Algorithms. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 141–183. Springer, 2010.

[132] D. Naddef and G. Rinaldi. Branch-and-cut algorithms for the capacitated VRP. *The vehicle routing problem*, 9:53–84, 2002.

[133] L. Nastansky, S. M. Selkow, and N. F. Stewart. Cost-minimal trees in directed acyclic graphs. *Mathematical Methods of Operations Research*, 18(1):59–67, 1974.

[134] G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience, 1988.

[135] C. A. S. Oliveira and P. M. Pardalos. A survey of combinatorial optimization problems in multicast routing. *Computers & Operations Research*, 32(8):1953–1981, 2005.

[136] C. A. S. Oliveira, P. M. Pardalos, and M. G. C. Resende. Optimization problems in multicast tree construction. In *Handbook of Optimization in Telecommunications*, chapter 25, pages 5–35. Springer Science + Business Media, 2006.

[137] M. Padberg and T.-Y. Sung. An analytical comparison of different formulations of the travelling salesman problem. *Mathematical Programming*, 52(1):315–357, 1991.

[138] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Dover Publications, 1998.

[139] M. Parsa, Q. Zhu, and J. J. Garcia-Luna-Aceves. An iterative algorithm for delay-constrained minimum-cost multicasting. *IEEE/ACM Transactions on Networking*, 6(4):461–474, 1998.

[140] J. C. Picard and M. Queyranne. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research*, 26(1):86–110, 1978.

[141] S. Pirkwieser and G. R. Raidl. A Column Generation Approach for the Periodic Vehicle Routing Problem with Time Windows. In G. Bigi, A. Frangioni, and M. G. Scutellà, editors, *Proceedings of the International Network Optimization Conference 2009*, Pisa, Italy, 2009.

[142] M. Poggi de Aragão, E. Uchoa, and R. F. Werneck. Dual heuristics on the exact solution of large Steiner problems. *Electronic Notes in Discrete Mathematics*, 7:150–153, 2001.

[143] T. Polzin and S. Vahdati Daneshmand. A comparison of Steiner tree relaxations. *Discrete Applied Mathematics*, 112(1-3):241–261, 2001.

[144] M. Prandtstetter and G. R. Raidl. An Integer Linear Programming Approach and a Hybrid Variable Neighborhood Search for the Car Sequencing Problem. *European Journal of Operational Research*, 191(3):1004–1022, 2008.

[145] R. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957.

[146] H. Prüfer. Neuer beweis eines satzes über permutationen. *Archiv für Mathematik und Physik*, 27:142–144, 1918.

[147] J. Puchinger and G. R. Raidl. Models and Algorithms for Three-Stage Two-Dimensional Bin Packing. *European Journal of Operational Research*, 183(3):1304–1327, 2007.

[148] J. Puchinger and G. R. Raidl. Bringing Order into the Neighborhoods: Relaxation Guided Variable Neighborhood Search. *Journal of Heuristics*, 14(5):457–472, 2008.

[149] J. Puchinger, G. R. Raidl, and S. Pirkwieser. MetaBoosting: Enhancing Integer Programming Techniques by Metaheuristics. In V. Maniezzo, T. Stützle, and S. Voss, editors, *Matheuristics – Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*, pages 71–102. Springer, 2009.

[150] R. Qu, Y. Xu, and G. Kendall. A Variable Neighborhood Descent Search Algorithm for Delay-Constrained Least-Cost Multicast Routing. In T. Stützle, editor, *Learning and Intelligent Optimization*, volume 5851 of *LNCS*, pages 15–29. Springer, 2009.

[151] G. R. Raidl. A Unified View on Hybrid Metaheuristics. In F. Almeida, M. J. Blesa Aguilera, C. Blum, J. M. Moreno Vega, M. Perez, A. Roli, and M. Sampels, editors, *Proceedings of the Hybrid Metaheuristics Workshop*, volume 4030 of *LNCS*, pages 1–12. Springer, 2006.

[152] G. R. Raidl and B. Hu. Enhancing Genetic Algorithms by a Trie-Based Complete Solution Archive. In P. Cowling and P. Merz, editors, *Evolutionary Computation in Combinatorial Optimisation – EvoCOP 2010*, volume 6022 of *LNCS*, pages 239–251. Springer, 2010.

[153] G. R. Raidl and J. Puchinger. Combining (Integer) Linear Programming Techniques and Metaheuristics for Combinatorial Optimization. In C. Blum, M. J. B. Augilera, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics – An Emergent Approach for Combinatorial Optimization*, volume 114 of *Studies in Computational Intelligence*, pages 31–62. Springer, 2008.

[154] G. R. Raidl, J. Puchinger, and C. Blum. Metaheuristic Hybrids. In M. Gendreau and J. Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 469–496. Springer, 2010.

[155] C. R. Reeves. Genetic Algorithms. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 109–139. Springer, 2010.

[156] M. G. C. Resende and P. M. Pardalos. *Handbook of optimization in telecommunications*, volume 10. Springer, 2006.

[157] C. C. Ribeiro, E. Uchoa, and R. F. Werneck. A Hybrid GRASP with Perturbations for the Steiner Problem in Graphs. *INFORMS Journal on Computing*, 14(3):228–246, 2003.

[158] G. Robins and A. Zelikovsky. Tighter bounds for graph Steiner tree approximation. *SIAM Journal on Discrete Mathematics*, 19(1):122, 2006.

[159] G. N. Rouskas and I. Baldine. Multicast routing with end-to-end delay and delay variation constraints. In *Proceedings of IEEE Conference on Computer Communications*, volume 1, pages 353–360. IEEE, 1996.

[160] G. N. Rouskas and I. Baldine. Multicast routing with end-to-end delay and delay variation constraints. *IEEE Journal on Selected Areas in Communications*, 15(3):346–356, 1997.

[161] M. Ruthmair and G. R. Raidl. A Kruskal-Based Heuristic for the Rooted Delay-Constrained Minimum Spanning Tree Problem. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Proceedings of the 12th International Conference on Computer Aided Systems Theory*, volume 5717 of *LNCS*, pages 713–720. Springer, 2009.

[162] M. Ruthmair and G. R. Raidl. Variable Neighborhood Search and Ant Colony Optimization for the Rooted Delay-Constrained Minimum Spanning Tree Problem. In R. Schaefer et al., editors, *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part II*, volume 6239 of *LNCS*, pages 391–400. Springer, 2010.

[163] M. Ruthmair and G. R. Raidl. A Layered Graph Model and an Adaptive Layers Framework to Solve Delay-Constrained Minimum Tree Problems. In O. Günlük and G. Woeginger, editors, *Proceedings of the 15th Conference on Integer Programming and Combinatorial Optimization (IPCO XV)*, volume 6655 of *LNCS*, pages 376–388. Springer, 2011.

[164] M. Ruthmair and G. R. Raidl. A Memetic Algorithm and a Solution Archive for the Rooted Delay-Constrained Minimum Spanning Tree Problem. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I*, volume 6927 of *LNCS*, pages 351–358. Springer, 2012.

[165] M. Ruthmair and G. R. Raidl. On Solving the Rooted Delay- and Delay-Variation-Constrained Steiner Tree Problem. In *Proceedings of the 2nd International Symposium on Combinatorial Optimization*, LNCS. Springer, 2012 (to appear).

[166] H. F. Salama, D. S. Reeves, and Y. Viniotis. An Efficient Delay-Constrained Minimum Spanning Tree Heuristic. In *Proceedings of the 5th International Conference on Computer Communications and Networks*. IEEE Press, 1996.

[167] H. F. Salama, D. S. Reeves, and Y. Viniotis. The Delay-Constrained Minimum Spanning Tree Problem. In *Proceedings of the 2nd IEEE Symposium on Computers and Communications*, pages 699–703. IEEE Computer Society, 1997.

[168] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

[169] T. Seidl. A Multilevel Refinement Approach to the Rooted Delay-Constrained Steiner Tree Problem. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Vienna, Austria, Sept. 2011.

[170] H. D. Sherali and W. P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990.

[171] P.-R. Sheu and S.-T. Chen. A fast and efficient heuristic algorithm for the delay- and delay variation-bounded multicast tree problem. *Computer Communications*, 25(8):825–833, 2002.

[172] P.-R. Sheu, H.-Y. Tsai, and S.-C. Chen. An Optimal MILP Formulation for the Delay- and Delay Variation-Bounded Multicast Tree Problem. *Journal of Internet Technology*, 8(3):321–328, 2007.

[173] N. Skorin-Kapov and M. Kos. The application of Steiner trees to delay constrained multicast routing: a tabu search approach. In *Proceedings of the 7th International Conference on Telecommunications*, volume 2, pages 443–448, 2003.

[174] N. Skorin-Kapov and M. Kos. A GRASP heuristic for the delay-constrained multicast routing problem. *Telecommunication Systems*, 32(1):55–69, 2006.

[175] A. Sramko. Enhancing a Genetic Algorithm by a Complete Solution Archive Based on a Trie Data Structure. Master's thesis, Vienna University of Technology, Institute of Computer Graphic s and Algorithms, Vienna, Austria, Feb. 2009.

[176] T. Stützle and H. H. Hoos. MAX–MIN ant system. *Future Generation Computer Systems*, 16:889–914, 2000.

[177] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Mathematica Japonica*, 24(6):573–577, 1980.

[178] F. Vanderbeck. Implementing Mixed Integer Column Generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, chapter 12, pages 331–358. Springer, 2005.

[179] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[180] R. Viertl. *Einführung in die Stochastik: mit Elementen der Bayes-Statistik und der Analyse unscharfer Information*. Springer, 2003.

[181] S. Voß. The Steiner Tree Problem with Hop Constraints. *Annals of Operations Research*, 86(0):321–345, 1999.

[182] S. Voß, S. Martello, I. H. Osman, and C. Roucairol, editors. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, 1999.

[183] C. Walshaw. Multilevel Refinement for Combinatorial Optimisation Problems. *Annals of Operations Research*, 131(1-4):325–372, 2004.

[184] C. Walshaw. Multilevel refinement for combinatorial optimisation: Boosting metaheuristic performance. In C. Blum, M. J. B. Aquilera, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics: An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence (SCI)*, pages 261–289. Springer, 2008.

[185] H. Wang, Z. Shi, and S. Li. Multicast routing for delay variation bound using a modified ant colony algorithm. *Journal of Network and Computer Applications*, 32(1):258–272, 2009.

[186] X. Wang and A. C. Regan. Local truckload pickup and delivery with hard time window constraints. *Transportation Research Part B: Methodological*, 36(2):97–112, 2002.

[187] B. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988.

[188] D. Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[189] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

[190] L. A. Wolsey. *Integer programming*. Wiley, New York, 1998.

[191] R. T. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28(3):271–287, 1984.

[192] Y. Xu and R. Qu. A GRASP approach for the Delay-constrained Multicast routing problem. In *Proceedings of the 4th Multidisplinary International Scheduling Conference*, pages 93–104, Dublin, Ireland, 2009.

[193] Y. Xu and R. Qu. A hybrid scatter search meta-heuristic for delay-constrained multicast routing problems. *Applied Intelligence*, 36(1):229–241, 2010.

[194] A. Zelikovsky. A series of approximation algorithms for the acyclic directed Steiner tree problem. *Algorithmica*, 18(1):99–110, 1997.

[195] Q. Zhang and Y. W. Leung. An orthogonal genetic algorithm for multimedia multicast routing. *IEEE Transactions on Evolutionary Computation*, 3(1):53–62, 1999.

[196] Q. Zhu, M. Parsa, and J. J. Garcia-Luna-Aceves. A source-based algorithm for delay-constrained minimum-cost multicasting. In *Proceedings of 14th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 377–385. IEEE, 1995.

APPENDIX A

Curriculum Vitae

A.1 Personal Information

Name:	Mario Ruthmair
Date and place of birth:	July 11, 1980, St.Pölten, Austria
Nationality:	Austria
Resident:	Herbeckstraße 80/1, 1180 Vienna, Austria
Family status:	unmarried, no children
Languages:	German (native), English (fluent)
E-mail address:	ruthmair@ads.tuwien.ac.at

A.2 Education

since 2008/05:	PhD (Doctorate) Studies in computer science at the Vienna University of Technology, Austria; under the supervision of Günther R. Raidl and Ulrich Pferschy.
1998/10 – 2006/06:	Studies of computer science at the Vienna University of Technology, Austria; graduation to "Diplom-Ingenieur" (MSc equivalent) passed with distinction.
1990/09 – 1998/06:	Secondary School, St.Pölten, Austria; general qualification for university entrance passed with distinction.
1986/09 – 1990/06:	Elementary School, Weißenkirchen/Perschling, Austria.

A.3 Professional Activities

since 2008/07:	One-man business Ruthmair e.U. (IT Consulting and Services), Vienna, Austria.
since 2008/05:	Research and teaching assistant, Algorithms and Data Structures Group, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria.
2002/12 – 2007/12:	Part/Full time employment as IT technician and consultant at Syncomp Data Systems, St.Pölten, Austria.
2002/09:	Internship at Siemens Austria (PSE), Vienna, Austria.
1999/10 – 2000/09:	Civilian national service (instead of military service) at Arbeiter-Samariterbund, St.Pölten, Austria.
1999/08:	Internship at the municipal office Weißenkirchen/Perschling

A.4 International Organizational and Reviewing Activities

- Reviewer for:
  - European Journal of Operational Research, since 2012.
  - Information Sciences, since 2011.
  - Soft Computing, since 2011.
- (Sub)referee for:
  - Workshop Heuristic Problem Solving at Eurocast 2011 – 13th International Conference on Computer Aided Systems Theory.
  - Workshop Heuristic Problem Solving at Eurocast 2009 – 12th International Conference on Computer Aided Systems Theory.
- Member of the Local Organizing Committee, 7th International Workshop on Hybrid Metaheuristics, Vienna, 2010.

A.5 Teaching Activities

Contributions as lecturer to the following courses:

- Algorithms and Data Structures (basic course)
- Algorithmics (randomized, approximation, and graph algorithms, basics of linear and integer programming)
- Efficient Algorithms (selected algorithms in text search, cryptography, and random sampling)
- Mathematical Programming (extended topics on integer programming)

A.6 List of Publications

A.6.1 Refereed Conference and Workshop Papers

1. Mario Ruthmair and Günther R. Raidl. *A Kruskal-Based Heuristic for the Rooted Delay-Constrained Minimum Spanning Tree Problem.* In A. Quesada-Arencibia et al., editors, Extended Abstracts of the 12th International Conference on Computer Aided Systems Theory, pages 244-246, 2009.

2. Mario Ruthmair and Günther R. Raidl. *A Kruskal-Based Heuristic for the Rooted Delay-Constrained Minimum Spanning Tree Problem.* In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, Proceedings of the 12th International Conference on Computer Aided Systems Theory, volume 5717 of LNCS, pages 713-720. Springer, 2009.

3. Jakob Walla, Mario Ruthmair, and Günther R. Raidl. *Solving a Video-Server Load Re-Balancing Problem by Mixed Integer Programming and Hybrid Variable Neighborhood Search.* In M. J. Blesa et al., editors, Hybrid Metaheuristics 2009, volume 5818 of LNCS, pages 84-99. Springer, 2009.

4. Mario Ruthmair and Günther R. Raidl. *Variable Neighborhood Search and Ant Colony Optimization for the Rooted Delay-Constrained Minimum Spanning Tree Problem.* In R. Schaefer et al., editors, Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part II, volume 6239 of LNCS, pages 391-400. Springer, 2010.

5. Mario Ruthmair, Andreas Hubmer, and Günther R. Raidl. *Using a Solution Archive to Enhance Metaheuristics for the Rooted Delay-Constrained Minimum Spanning Tree Problem.* In A. Quesada-Arencibia et al., editors, Extended Abstracts of the 13th International Conference on Computer Aided Systems Theory, pages 285-287, 2011.

6. Martin Berlakovich, Mario Ruthmair, and Günther R. Raidl. *A Multilevel Heuristic for the Rooted Delay-Constrained Minimum Spanning Tree Problem.* In A. Quesada-Arencibia et al., editors, Extended Abstracts of the 13th International Conference on Computer Aided Systems Theory, pages 247-249, 2011.

7. Mario Ruthmair and Günther R. Raidl. *A Layered Graph Model and an Adaptive Layers Framework to Solve Delay-Constrained Minimum Tree Problems.* In O. Günlük and G.J. Woeginger, editors, Proceedings of the 15th Conference on Integer Programming and Combinatorial Optimization (IPCO XV), volume 6655 of LNCS, pages 376-388. Springer, 2011.

8. Markus Leitner, Mario Ruthmair, and Günther R. Raidl. *Stabilized Branch-and-Price for the Rooted Delay-Constrained Steiner Tree Problem.* In J. Pahl, T. Reiners, and S. Voß, editors, Network Optimization: 5th International Conference, INOC 2011, volume 6701 of LNCS, pages 124-138, Hamburg, Germany, 2011. Springer.

9. Markus Leitner, Mario Ruthmair, and Günther R. Raidl. *Stabilized Column Generation for the Rooted Delay-Constrained Steiner Tree Problem.* In Proceedings of the VII ALIO/EURO - Workshop on Applied Combinatorial Optimization, pages 250-253, Porto, Portugal, 2011.

10. Mario Ruthmair and Günther R. Raidl. *A Memetic Algorithm and a Solution Archive for the Rooted Delay-Constrained Minimum Spanning Tree Problem.* In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I, volume 6927 of LNCS, pages 351-358. Springer, 2012.

11. Martin Berlakovich, Mario Ruthmair, and Günther R. Raidl. *A Multilevel Heuristic for the Rooted Delay-Constrained Minimum Spanning Tree Problem.* In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I, volume 6927 of LNCS, pages 256-263. Springer, 2012.

12. Mario Ruthmair and Günther R. Raidl. *On Solving the Rooted Delay- and Delay-Variation-Constrained Steiner Tree Problem.* In Proceedings of the 2nd International Symposium on Combinatorial Optimization, LNCS. Springer, 2012 (to appear).

13. Thorsten Krenek, Mario Ruthmair, and Günther R. Raidl. *(Hybrid) Metaheuristics to Fuel Consumption Optimization of Hybrid Electric Vehicles.* In C. Di Chio et al., editors, Proceedings of the European Conference on the Applications of Evolutionary Computation, volume 7248 of LNCS, pages 376-385. Springer, 2012.

A.6.2 Research Reports

1. Markus Leitner, Mario Ruthmair, and Günther R. Raidl. *On Stabilized Branch-and-Price for Constrained Tree Problems.* Technical Report TR 186-1-11-01, Vienna University of Technology, Vienna, Austria, 2011. accepted with revisions to Networks (INOC 2011 special issue).

A.6.3 Thesis

1. Mario Ruthmair. *Gateway zur Übertragung von Audiodaten von einem RTSP-Server in ein IEEE1394-Netzwerk.* Master's thesis, Vienna University of Technology, Institute of Computer Technology, Vienna, Austria, March 2006.

A.6.4 Co-Supervised Thesis

1. Martin Berlakovich. *Multilevel Heuristiken für das Rooted Delay-Constrained Minimum Spanning Tree Problem.* Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Vienna, Austria, July 2010.

2. Thorsten Krenek. *Verbrauchsminimierung eines Hybridfahrzeuges im Neuen Europäischen Fahrzyklus.* Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Vienna, Austria, July 2011.

3. Thomas Seidl. *A Multilevel Refinement Approach to the Rooted Delay-Constrained Steiner Tree Problem.* Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Vienna, Austria, September 2011.

4. Jakob Walla. *Exakte und heuristische Optimierungsmethoden zur Lösung von Video Server Load Re-Balancing.* Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Vienna, Austria, April 2009.

A.7 Posters and Presentations

1. *On Solving the Rooted Delay- and Delay-Variation-Constrained Steiner Tree Problem.* ISCO 2012, Athens, Greece, 2012.

2. *An Adaptive Layers Framework for Resource-Constrained Network Design Problems.* INFORMS 2012, Boca Raton, Florida, USA, 2012.

3. *A Layered Graph Model and an Adaptive Layers Framework to Solve Delay-Constrained Minimum Tree Problems.* IPCO 2011, Armonk, New York, USA, 2011.

4. *A Memetic Algorithm and a Solution Archive for the Rooted Delay-Constrained Minimum Spanning Tree Problem.* EUROCAST 2011, Las Palmas, Spain, 2011.

5. *Variable Neighborhood Search and Ant Colony Optimization for the Rooted Delay-Constrained Minimum Spanning Tree Problem.* PPSN 2010, Krakòw, Poland, 2010.

6. *A Kruskal-Based Heuristic for the Rooted Delay-Constrained Minimum Spanning Tree Problem.* EUROCAST 2009, Las Palmas, Spain, 2009.