Die approbierte Originalversion dieser Dissertation ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Advances in Decomposition Approaches for Mixed Integer Linear Programming

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Martin Riedler, BSc
Registration Number 0828221

to the Faculty of Informatics

at the TU Wien

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

The dissertation has been reviewed by:

Christina Büsing	Markus Leitner	Günther R. Raidl

Vienna, 11th October, 2018

Martin Riedler

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Martin Riedler, BSc
Weindlau 30, 4432 Ernsthofen

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 Quelle als Entlehnung kenntlich gemacht habe.

Wien, 11. Oktober 2018

Martin Riedler

Acknowledgements

First and foremost I want to thank my advisor Günther Raidl for giving me the opportunity to write this thesis. I am grateful for his inspiring suggestions of possible research directions. He provided guidance whenever I needed it and always had further ideas on how to extend or improve the researched algorithms.

Besides my advisor, I would like to thank Christina Büsing and Markus Leitner for taking the time to evaluate this thesis and providing valuable comments.

I thank my co-authors Ivana Ljubić, Markus Leitner, and Mario Ruthmair for collaborating with me on network design problems. Without the tricks and details they told me about mixed integer linear programming and the associated solvers many of the algorithms presented in this thesis would not have been such a success.

Thanks also go to my former and current colleagues at the Algorithms and Complexity group. Discussing open issues was always helpful to identify possible solutions or to view a problem from another perspective. Besides, the non research related discussions were important to relieve stress and to take a break every now and then.

I would like to thank cancer treatment center EBG MedAustron GmbH, Wiener Neustadt, Austria, for their collaboration on particle therapy scheduling. I am grateful for the received funding as well as for the provided insights into the practice of particle therapy patient scheduling.

Last but not least, I want to thank my family for all their support and encouragement in the past years that made this work possible.

Abstract

In this thesis we consider different decomposition approaches for mixed integer linear programming (MILP). We work with well-known techniques from the literature such as cutting plane methods, column generation, and logic-based Benders decomposition but also more recently developed approaches based on iteratively refined relaxations. Moreover, we consider combinations of these algorithms and integrate other techniques such as constraint programming and/or (meta-)heuristics. The aim of this thesis is twofold. First, we want to exploit these methods to solve challenging optimization problems. Thereby we investigate what makes a specific approach effective for dealing with a particular application and which adjustments and extensions are important to improve performance. Second, we use the gained insights to advance the methods. The merits of our discoveries are supported by extensive computational studies that underline the potential of the proposed enhancements.

The first algorithm we consider is based on column generation in combination with cutting planes for solving the network design problem with relays. We address a distance restriction imposed by this problem through transforming the input graph to a so-called communication graph. Thereby we manage to overcome the limitations of a previous column generation model from the literature that suffers from a rather unbalanced decomposition due to shifting too much effort into the pricing subproblem.

The second decomposition approach considered in this thesis is based on logic-based Benders decomposition. Experiments are conducted for a selective variant of the dial-a-ride problem. In terms of the decomposition we achieve a separation into an optimization-based request selection aspect and a feasibility-related routing part. Consequently, we can exploit specialized algorithms to solve each of them as efficiently as possible. The selection problem is tackled by a MILP approach that we strengthen through cutting planes derived from subproblem relaxations. The routing problem is solved through a hybrid of constraint programming and MILP. We enhance the Benders algorithm by heuristic speedup techniques and consider different strategies for computing Benders cuts. An approach deriving Benders cuts from infeasible substructures of minimum cardinality is proven to be highly effective and also promising for other work in this area.

In the remainder of the thesis we focus on algorithms based on extended formulations that are solved through relaxations. The first application we consider is a scheduling problem. We investigate a scenario that demands a very fine-grained time discretization.

Traditional MILP formulations are known to be inefficient in this context due to the large number of time instants that have to be considered. We develop an alternative based on a relaxation obtained by aggregating subsequent time instants into so-called time buckets. Scheduling in terms of this aggregation is less precise and might therefore lead to infeasibilities, however, also helps to substantially decrease the problem size. We retain feasibility and achieve optimality by iteratively splitting time buckets to improve accuracy where necessary. Moreover, we also exploit the relaxation to derive intermediate feasible solutions by (meta-)heuristics. Considerable effort is invested into developing and testing strategies for implementing the refinement step in which the buckets are subdivided. Our most successful algorithm is based on incorporating additional knowledge from the intermediate solutions to address the remaining infeasibilities as effectively as possible. In comparison to standard MILP formulations from the literature we show that our algorithm performs significantly better. Due to its generality our approach is also promising for other problems with similar characteristics.

The second approach based on relaxations is used to solve the directed variant of the network design problem with relays. To consider a modified side constraint we use layered graphs instead of the aforementioned transformation to a communication graph. The idea of layered graphs is to extend a base graph along one or multiple dimensions to state enhanced MILP formulations. In terms of the investigated problem we introduce node copies according to the traversed distance within the graph. To avoid prohibitive graph sizes caused by fractional inputs we consider a dynamic approach. We obtain a relaxation through rounding down all fractional distance values and address potential infeasibilities by cutting planes. Further strengthening inequalities and symmetry breaking constraints are added to enhance the resulting model. Our experiments indicate that infeasibilities in the relaxation are rare, which makes our separation approach effective in practice.

Often the size of layered graphs is prohibitive already for integral inputs. In these situations we can employ a strategy similar to the algorithm considered for high-resolution scheduling. Instead of aggregating time instants we omit node copies and redirect arcs accordingly. The resulting iterative algorithm is based on the observation that only a (small) subset of the nodes is required to obtain optimal solutions. To this end we successively reintroduce some of the omitted node copies until optimality can be proven. Similar to the scheduling problem the success of such an algorithm strongly depends on the strategy according to which the graph is extended in each iteration. We develop new path-based approaches to benefit from the structural knowledge encoded in the layered graph relaxation. A computational study on two benchmark problems shows the effectiveness of these strategies.

Kurzfassung

In dieser Arbeit werden unterschiedliche Zerlegungsansätzen für gemischt-ganzzahlige lineare Optimierung (engl. mixed integer linear programming, MILP) untersucht. Hierzu kommen etablierte Techniken wie das Schnittebenenverfahren, Spaltengenerierung und die Logik-basierte Benders-Zerlegung, sowie neuere Verfahren basierend auf schrittweise verfeinerten Relaxationen zum Einsatz. Insbesondere werden Kombinationen dieser Algorithmen mit anderen Techniken, wie Constraintprogrammierung oder (Meta-)Heuristiken verwendet. Im Rahmen dieser Arbeit werden zwei Hauptziele verfolgt. Zum einen soll gezeigt werden, wie mit diesen Techniken komplexe Optimierungsprobleme gelöst werden. Dabei wird zum einen darauf eingegangen, wodurch sich die unterschiedlichen Verfahren für bestimmte Problemstellungen besonders eignen bzw. welche Anpassungen und Erweiterungen notwendig sind, um deren Effektivität zu steigern. Zum anderen werden die gewonnenen Erkenntnisse genutzt, um die Verfahren weiterzuentwickeln. Die gezogenen Schlussfolgerungen sowie das Potential der entwickelten Verbesserungen werden durch ausführliche Experimente demonstriert.

Als erstes Verfahren kommt Spaltengenerierung in Kombination mit dem Schnittebenenverfahren zur Lösung des Netzwerkentwurfproblems mit Verstärkern zum Einsatz. Eine Nebenbedingung dieses Problems betreffend die zurückgelegten Distanzen im Netzwerk wird dabei mithilfe einer Transformation des Basisgraphen in einen sogenannten Kommunikationsgraphen modelliert. Dadurch wird eine Zerlegung ermöglicht, die im Gegensatz zu einem ähnlichen Ansatz aus der Literatur den Aufwand im Unterproblem reduziert.

Als Zweites wird ein Algorithmus basierend auf Logik-basierter Benders-Zerlegung entwickelt für die Lösung einer selektiven Variante des *Dial-a-Ride* Problems entwickelt. Der Zerlegungsansatz ermöglicht eine Aufteilung in einen Optimierungsaspekt für die Anfrageaufteilung und einen zweiten Aspekt, der die Gültigkeit der entstehenden Routen prüft. Dadurch können spezialisierte Algorithmen angewandt werden. Der Optimierungsaspekt wird durch ein MILP-Modell, gestärkt durch Schnittebenen basierend auf Relaxationen des Subproblems, gelöst. Für das Routenplanungsproblem kommt eine Kombination aus Constraintprogrammierung und MILP zum Einsatz. Der Benders Algorithmus wird durch heuristische Techniken beschleunigt. Darüber hinaus werden unterschiedliche Strategien zur Berechnung der Benders Schnitte untersucht. Am effektivsten zeigte sich eine Variante, die Schnitte aus ungültigen Strukturen kleinster Kardinalität ableitet. Diese erscheint vielversprechend für zukünftige Arbeiten in diesem Gebiet.

Der übrige Teil dieser Arbeit beschäftigt sich mit erweiterten Modellen, die mithilfe von Relaxationen gelöst werden. Als erste Anwendung wird ein Terminplanungsproblem untersucht, bei dem eine besonders feine Zeitauflösung berücksichtigt werden muss. Etablierte MILP-Modelle sind entsprechend den Erkenntnissen der bestehenden Literatur unter solchen Voraussetzung ineffektiv. Als Alternative wird eine Relaxation entwickelt, die aufeinanderfolgende Zeitpunkte in Zeitintervalle aggregiert. Auf diese Weise kann die Problemgröße deutlich reduziert werden. Allerdings kann die dadurch verlorene Genauigkeit zur Verletzung von Nebenbedingungen führen. Um Gültigkeit zu gewährleisten wird die Relaxation durch schrittweises Aufspalten der Intervalle nach und nach verfeinert. Mithilfe von (Meta-)Heuristiken werden aus den ungültigen Lösungen gültige Zwischenlösungen abgeleitet. Ein besonderer Fokus liegt auf der Entwicklung und dem Vergleich von Verfahren zum Aufspalten der Zeitintervalle. Der erfolgreichste Algorithmus nutzt Informationen, die aus den heuristisch berechneten Zwischenlösungen extrahiert werden, um Rückschlüsse auf die Ursachen der verletzten Nebenbedingungen zu ziehen. Das neue Verfahren erreicht dadurch eine deutliche Verbesserung gegenüber den etablierten MILP-Ansätzen der Literatur. Diese neue Technik erscheint auch vielversprechend für andere Probleme mit ähnlichen Eigenschaften.

Mithilfe des nächsten Ansatzes wird die gerichtete Variante des obengenannten Netzwerkentwurfproblems mit Verstärkern gelöst. Um eine abgewandelte Nebenbedingung zu berücksichtigen, kommen sogenannte *layered graphs* anstatt des zuvor verwendeten Kommunikationsgraphen zum Einsatz. *Layered graphs* erweitern den Graphen des Originalproblems anhand einer oder mehrerer Problemdimensionen, um effektivere MILP-Formulierungen zu ermöglichen. Im Rahmen des betrachteten Problems fügen wir Knotenkopien in Bezug auf die zurückgelegten Distanzen im Graphen ein. Um problematische Graphengrößen infolge rationaler Distanzen zu vermeiden, kommt ein dynamisches Verfahren zum Einsatz. Dazu werden rationale Distanzen abgerundet und dadurch verletzte Nebenbedingungen mithilfe von Schnittebenen behandelt. Darüber hinaus wird das resultierende Modell durch weitere Ungleichungen zur Stärkung und Symmetrievermeidung ergänzt. Die durchgeführten Experimente zeigen, dass Nebenbedingungen infolge des Rundens nur selten verletzt werden und die Separierung in der Praxis somit sehr effektiv arbeitet.

In vielen Fällen sind Modelle basierend auf *layered graphs* bereits bei integralen Eingaben zu groß, um effektiv genutzt werden zu können. Als Abhilfe kann ein Verfahren ähnlich des Algorithmus für das obige Terminplanungsproblem eingesetzt werden. Anstatt Zeitpunkte zu aggregieren, werden Knotenkopien weggelassen und Kanten entsprechend umgelenkt. Der entstehende Ansatz basiert auf der Erkenntnis, dass optimale Lösungen meist mit einer (kleinen) Teilmenge der Knoten bestimmt werden können. Der Erfolg des Verfahrens hängt hier stark von der Strategie ab, anhand derer iterativ weitere Knotenkopien in der Relaxation ergänzt werden, um eine optimale Lösung zu berechnen. Zu diesem Zweck werden Pfad-basierte Techniken, gestützt auf strukturelle Informationen des *layered graph*, entwickelt. Die Effektivität dieser neuen Verfahren wird anhand von Experimenten mit zwei unterschiedlichen Optimierungsproblemen demonstriert.

Contents

Abstract **vii**

Kurzfassung **ix**

Contents **xi**

1 Introduction **1**
1.1 Structure of the Thesis 4

2 Methodology **7**
2.1 Basic Definitions 7
2.2 Exact Methods 9
2.3 Heuristic Methods 23
2.4 Hybrid Methods 26

3 Exact Approaches for Network Design Problems with Relays **29**
3.1 Introduction 30
3.2 Solution Properties 34
3.3 Mixed Integer Linear Programming Formulations 38
3.4 Algorithmic Framework 46
3.5 Computational Study 51
3.6 Conclusion 61

4 Solving a selective DARP with LBBD **65**
4.1 Introduction 66
4.2 Formulations 70
4.3 Algorithmic Framework 76
4.4 Computational Study 87
4.5 Conclusion 104

5 ITBRA for High-Resolution RCPSP **109**
5.1 Introduction 110
5.2 Simplified Intraday Particle Therapy Patient Scheduling Problem . . . 112
5.3 Related Work 112

5.4 Reference MILP Models 115
5.5 Time-Bucket Relaxation 118
5.6 Iterative Time-Bucket Refinement Algorithm 124
5.7 Implementation Details 133
5.8 Computational Study 136
5.9 Conclusion 146

6 Exact Approaches for the DNDPR 149
6.1 Introduction 150
6.2 Formulations 154
6.3 Algorithmic Framework 161
6.4 Computational Study 165
6.5 Conclusion 179

7 Strategies for Iteratively Refining Layered Graph Models 183
7.1 Introduction 184
7.2 Mathematical Formalization 186
7.3 Algorithmic Framework 189
7.4 Computational Study 193
7.5 Conclusion and Future Work 197

8 Conclusions 199

Appendix A Additional Result Tables for the Selective DARP 203

Acronyms 209

Bibliography 213

CHAPTER 1

Introduction

Today's society is facing an ever-growing demand for our planet's limited resources. Striving for their most efficient, economical, and sustainable usage appears to be an immediate consequence. Algorithms are a valuable tool to help in this respect. Many tasks that impact our daily life can be modeled as (discrete) optimization problems. Examples are:

- routing problems that aim at minimizing transportation costs and fuel consumption,
- network design problems that, e.g., aim at minimizing costs for constructing and operating optical fiber networks, and
- scheduling problems that aim, e.g., at optimizing the utilization of expensive and/or critical resources or at optimizing production/manufacturing processes and arranging tasks executed by employees.

A well-known tool to model such problems and for deriving algorithms that compute (optimal) solutions is mixed integer linear programming (MILP). This formalism essentially describes on optimization problem in terms of decision variables subject to linear equality and inequality constraints and a linear objective function. In some cases such models can be directly handed over to a general purpose black-box solver to obtain optimal solutions when given enough time. Unfortunately, MILP is $\mathcal{NP}$-hard. This means that in general such problems cannot be solved in polynomial time unless $\mathcal{P} = \mathcal{NP}$. Nevertheless, some $\mathcal{NP}$-hard problems can be addressed by (straightforward) polynomially-sized models and effectively solved by directly applying a MILP solver. However, frequently such an approach does not lead to state-of-the-art results and is not capable of solving larger instances in a reasonable amount of time. Fortunately, several advanced techniques are available to allow for more diverse modeling options and improved computational performance.

Basic MILP models usually involve only a (pseudo-)polynomial number of constraints and variables. More expressive formulations become possible when also allowing exponential size. Using even larger models when already struggling with the small ones sounds counter-intuitive at first. Fortunately, dynamic approaches have been developed to avoid the full size of the large resulting models: *cutting plane* methods for dealing with an exponential number of constraints and *column generation* for dealing with an exponential number of variables. The basic idea is to start with a small subset of the constraints and variables and then iteratively add constraints that are required to establish feasibility and variables to achieve optimality. For details see Section 2.2.3.

Another well-known strategy is to partition the variables of the model into subsets. In a first stage a solution with respect to one part of the variables—the *master problem*—is computed. Based on the restrictions of these trial values a solution with respect to the remaining variables is obtained in a second stage—the *subproblem*. As the decision for the first variable set ignored restrictions imposed on the other variables, it might not be possible to obtain a feasible solution for the subproblem. Information regarding such infeasibilities or otherwise information regarding the influence of the second variable set on the objective function is incorporated into the master problem by means of additional inequalities, called *Benders cuts*. This process is iterated until master and subproblem provide compatible solutions. Several aspects make such a decomposition promising. The master and the subproblem are typically much smaller than the original problem and thus can be solved faster. In particular, either of them might exhibit special structure that makes it substantially easier to solve. Moreover, the subproblem might decouple into several independently solvable problems which allows for parallelization. Besides, it is often the case that one of the two problems turns out to be a known optimization problem that allows benefiting from existing research and possibly available algorithms. Further details on this so-called *Benders decomposition* and its logic-based variant are provided in Section 2.2.4.

Another technique of exploiting MILP to solve more challenging optimization problems are *relaxations*. The idea of a relaxation is to formulate an easier problem that disregards part of the original problem's constraints or considers a modified objective function in a way that possibly extends the solution space to further (infeasible) solutions but retains all the original solutions and does not overestimate the original solution value (in case of minimization). Such a relaxed problem can often be solved considerably faster. Often it is sufficient to disregard only few constraints to obtain a polynomially solvable relaxation for an originally $\mathcal{NP}$-hard problem. This gives rise to the following strategy. Initially, we solve the relaxation to optimality and then check whether the obtained solution is feasible for the original problem. If this is the case, we can stop and know that the obtained solution is optimal. Otherwise, we derive a more precise relaxation that forbids at least the current infeasible solution. Iterating this procedure eventually leads to an optimal solution—assuming that the solution space is finite. The success of such a method relies on the assumption that solving a (small) series of relaxed problems can be substantially faster than directly solving the original problem.

What the methods described above have in common is that they rely on a decomposition principle. They try to overcome the challenges of the original problem by considering a sequence of smaller or more efficiently solvable subproblems instead. This thesis is concerned with different such decomposition methods. In particular, we strongly focus on MILP-based algorithms as a very popular and versatile tool to obtain optimal solutions for discrete optimization problems. Our aim is twofold. First, we want to exploit these methods to design successful solution strategies for solving challenging optimization problems to optimality. We chose a rather diverse selection of problems to show how such strategies can be used to deal with different applications. To this end we consider network design problems, a routing problem, and a scheduling application. Second, we aim at investigating the properties of these decomposition approaches to explore what makes them effective. The made observations are used to suggest adjustments and additions that help in improving these methods.

We start by considering an application of column generation. Our modeling approach considers a transformation of the input graph into a so-called *communication graph*. Thereby we manage to overcome the limitations of a more straightforward model from the literature that also relies on column generation. Afterwards we consider an approach using Benders decomposition. We achieve a structural decomposition into an optimization problem (master problem) and a feasibility problem (subproblem). This allows employing specialized algorithms to tackle both of them. To this end we solve the master problem with an MILP approach improved by further inequalities derived from so-called *subproblem relaxations*. In terms of the Benders algorithm we use heuristic speed-up techniques to decrease the time spent in the master problem. The subproblem is solved by means of a hybridization of constraint programming (CP) and MILP. With this technique we benefit from the higher speed of the CP approach and use the consistency of the MILP algorithm to counterbalance the few outliers in which the CP approach is significantly slower. Moreover, we consider different strategies for deriving Benders cuts. We compare plain Benders cuts to heuristically strengthened ones, as well as two variants of theoretically strongest cuts. The next contribution is based on relaxations. We consider a scheduling scenario in which a very fine-grained time discretization is required. To reduce the size of the otherwise prohibitive time horizon, we aggregate subsequent time instants into so-called *time buckets*. The MILP formulation based on this aggregation constitutes a relaxation to the original problem. To ensure feasibility we iteratively solve this relaxation and derive a subsequent more fine-grained model whenever infeasibilities are detected. In addition, we also employ (meta-)heuristics that are guided by the solutions to the relaxation and thus yield feasible solutions gradually increasing in quality. Optimality is proven by establishing feasibility of the relaxation or by deriving a feasible solution that matches the relaxation's solution value. Our contribution strongly focuses on a thorough investigation of different strategies for implementing the refinement process in which the time buckets are subdivided to improve the precision of the relaxation. This step is particularly important as it impacts the size of the intermediate MILP models and the total number of iterations until optimality can be proven, which are the dominant performance indicators of the algorithm. A similar approach is investigated for network design problems. Instead of aggregating time instants we reduce the number of nodes in an originally large graph by redirecting arcs. Some of the omitted nodes are then incrementally reintroduced to refine the associated relaxation as done for the scheduling problem above. The main difference comes from the fact that the underlying graph conveys more structural information. We exploit this additional knowledge to develop specialized refinement strategies based on graph algorithms.

1.1 Structure of the Thesis

We begin with a short introduction to the used methodological concepts. In this respect Chapter 2 summarizes the basics of exact and heuristic solution approaches that serve as foundation for the algorithms considered in this thesis.

Chapters 3 to 7 are devoted to the developed algorithms. All of them have either been published, are accepted for publication, or are currently submitted to high-class scientific journals or conferences in the respective field. Detailed information is provided at the beginning of each chapter.

In Chapter 3 we propose algorithms to obtain optimal solutions for the network design problem with relays (NDPR). This problem deals with the design of networks in which certain nodes have to communicate. Due to signal degradation a distance limit is imposed for sequences of traversed communication links. If signals need to be transmitted farther, expensive regeneration equipment has to be deployed. The goal of the problem is to select a set of communication links and a subset of nodes at which regeneration equipment is deployed such that a given set of node pairs can communicate at minimal total construction cost. We tackle the problem by algorithms based on exponentially sized MILP models, improved by strengthening inequalities. Different from an approach in the existing literature that prices entire commodity connections, we rely on a more fine-grained strategy based on a preceding graph transformation. This makes it possibly to reduce the difficulty of the subproblem and thereby improve the balance of the decomposition.

Chapter 4 considers a selective variant of the dial-a-ride problem (DARP). In the classical DARP customers have to be transported from pick-up to drop-off locations by a set of vehicles with limited capacity. Departure and arrival are restricted by time windows. Moreover, client satisfaction is taken into account by limiting the time a customer is allowed to be on board of a vehicle to avoid long detours. The most common setting considers minimization of the total routing cost while assuming that the number of available vehicles is sufficient to service all customers. We consider a different scenario in which we drop the assumption that all transportation requests must be accepted. Therefore, we aim at maximizing the number of served requests. This appears to be particularly relevant for sponsored systems which often occur in practice. We use logic-based Benders decomposition (LBBD) as solution method to decompose the problem into a selection aspect for assigning customers to vehicles and a routing aspect that checks whether the assignment permits a feasible tour for each vehicle. Through this structural decomposition we can employ specialized algorithms with individual improvements. Different strategies for strengthening Benders cuts are suggested and extensively compared.

In Chapter 5 we consider a scheduling scenario motivated by a real-world application in particle therapy for cancer treatment. A complicating circumstance is that tasks have to be scheduled in high time resolution to use the expensive particle beam as efficiently as possible. This leads to problems with well-known standard approaches for such problems. To overcome these issues, we suggest a so-called time-bucket relaxation that aggregates consecutive time instants to reduce the problem size. This relaxation is then iteratively refined to eventually obtain an optimal solution. In an extensive computational study we compare our novel algorithm to well-known approaches from the literature. Moreover, we provide a theoretical discussion of the potential modeling alternatives. This work emerged in collaboration with the cancer treatment center MedAustron[1] located in Wiener Neustadt, Austria.

The next two chapters both use models based on so-called *layered graphs (LGs)*. Such models are based on an extended formulation that considers some problem dimension explicitly. In particular, LGs are often useful to deal with otherwise challenging modeling aspects or even encode certain constraints implicitly This leads to larger but more efficiently solvable models. Of course size may become an issue if the incurred overhead or the initial problem is too large. In Chapter 6 we use LGs to model the directed network design problem with relays (DNDPR)—the directed counterpart of the NDPR. Distances are made explicit by introducing for each node copies with respect to the distances at which it might be feasibly reached. This allows to implicitly enforce the distance limit by omitting node copies beyond it. The placement of regeneration equipment is modeled through arcs that start at a higher layer and return to a node copy at layer zero. Even for rather small distance limits such an approach may become inefficient when dealing with non-integral distances that have to be addressed through expensive scaling. We overcome these issues by an optimality preserving combination of rounding and cutting planes.

Chapter 7 considers a general framework which is designed to avoid the full size of LGs for an important class of network design problems. The basic idea is to start with a subgraph that is substantially smaller but might not be sufficient to enforce all restrictions. This graph is then iteratively extended until an optimal solution is obtained—or a strong bound that can be used to verify optimality of a heuristic solution. We focus on the step in which the graph is extended and investigate the impact of employing different strategies. A comparison of our newly suggested techniques to those from the literature shows that (1) this component has a strong influence on the success of such an iterative approach and (2) advanced techniques can provide significant improvements to the algorithm.

Finally, the thesis is concluded in Chapter 8. We summarize the main findings and give an outlook on future research directions.

---

[1] https://www.medaustron.at

CHAPTER 2

Methodology

This chapter provides the terminology and the foundations of the algorithmic concepts used throughout this work. We start with some basic definitions before introducing exact and heuristic solution methods. Due to the fact that maximization problems can be converted to equivalent minimization problems (by multiplying the objective function by minus one) we restrict the discussion to minimization problems.

2.1 Basic Definitions

In this section we provide some basic terminology mainly following Papadimitriou and Steiglitz [133], Bertsimas and Tsitsiklis [17], and Wolsey [172]. We start by formalizing the concept of an optimization problem to define global and local optimality. Afterwards, we give definitions for bounds, relaxations and convexity.

Definition 2.1 ([133, p. 4]). *An* instance *of an optimization problem is a pair* (F, c) *with set* F *being the* domain *or* feasible set *and* c *the* objective function *or* cost function *defined as* $c\colon F \to \mathbb{R}$.

Definition 2.2 ([133, p. 4]). *For an instance* (F, c) *an element* $f \in F$ *such that* $c(f) \leq c(f'),\ \forall f' \in F$*, is called* global optimum.

The goal of optimization is to identify global optima. For brevity global optima are sometimes just called optima or optimal solutions if the precise meaning is unambiguous.

Definition 2.3 ([133, p. 4]). *An* optimization problem *is a set of instances.*

Definition 2.4 ([133, p. 7]). *A* neighborhood *of an optimization problem with instances* (F, c) *is a mapping* $N\colon F \to 2^F$.

Definition 2.5 ([133, p. 8])**.** *Given an instance (F, c), an element $f \in F$ is called* local optimum *or* locally optimal *with respect to neighborhood N if $c(f) \leq c(f')$, $\forall f' \in N(f)$.*

Definition 2.6 ([133, p. 10])**.** *Given feasible set F and a neighborhood N, if whenever $f \in F$ is locally optimal with respect to N it is also globally optimal, we say the neighborhood N is exact.*

Definition 2.7 ([172, p. 24])**.** *Value $p \in \mathbb{R}$ is called a* primal bound *for instance (F, c) with global optimum f^* iff $c(f^*) \leq p$.*

Definition 2.8 ([172, p. 24])**.** *Value $d \in \mathbb{R}$ is called a* dual bound *for instance (F, c) with global optimum f^* iff $d \leq c(f^*)$.*

Remark 2.1 ([172, p. 24])**.** *In minimization problems dual bounds are also called lower bounds and primal bounds are also called upper bounds. Conversely, in maximization problems dual bounds are called upper bounds and primal bounds are called lower bounds.*

Each of these bounds is called *tight* if it matches the value of the global optimum. Bounds are useful to prove optimality: If an element $f \in F$ for instance (F, c) has been identified and a dual bound d is known such that $c(f) = d$, then f must be a global optimum. Primal bounds are often obtained by heuristics, see Section 2.3 A common way to derive dual bounds is to solve so-called relaxations.

Definition 2.9 ([172, pp. 24–25])**.** *An instance (F', c') is a* relaxation *of instance (F, c) iff $F' \supseteq F$ and $c'(f) \leq c(f)$, $\forall f \in F$.*

This guarantees that an optimal solution to a relaxed instance provides a dual bound to the original instance.

Theorem 2.1 ([172, p. 26])**.** *Let (F, c) be an instance and (F', c') an associated relaxation with optimal solution $f^* \in F'$. If $f^* \in F$, then f^* is an optimal solution to the original instance.*

Definition 2.10 ([17, p. 44])**.** *For $\mathbf{x}^1, \ldots, \mathbf{x}^k$ vectors in $\mathbb{R}^n$ and $\lambda_1, \ldots, \lambda_k$ non-negative scalars with $\sum_{i=1}^k \lambda_i = 1$.*

(a) Vector $\sum_{i=1}^k \lambda_i x^i$ is called convex combination *of the vectors $\mathbf{x}^1, \ldots, \mathbf{x}^k$.*

(b) The set of all convex combinations of the vectors $\mathbf{x}^1, \ldots, \mathbf{x}^k$ is called convex hull*, denoted by* $\operatorname{conv}(\mathbf{x}^1, \ldots, \mathbf{x}^k)$.

Definition 2.11 ([17, p. 43])**.** *A set $S \subseteq \mathbb{R}^n$ is* convex *if it contains all convex combinations of points $x, y \in S$.*

Lemma 2.1 ([17, p. 44])**.** *The intersection of convex sets is convex.*

Definition 2.12 ([17, p. 15])**.** *Let $S \subseteq \mathbb{R}^n$ and $c\colon S \to \mathbb{R}$. Function c is convex in S if*

$$c(\lambda x + (1-\lambda)y) \leq c(\lambda x) + c((1-\lambda)y), \quad \forall x, y \in S, \lambda \in [0,1].$$

If a function is convex in $\mathbb{R}^n$, we simply call it convex.

Theorem 2.2 ([133, p. 14])**.** *Let (F, c) be an instance with convex set $F \subseteq \mathbb{R}^n$ and convex function c. The neighborhood based on Euclidean distance*

$$N_\epsilon(x) = \{y \in F : ||x - y|| \leq \epsilon\}$$

is exact for $\epsilon > 0$.

2.2 Exact Methods

Exact solution methods compute global optima. Solutions for polynomially solvable problems are usually directly constructed by problem-specific algorithms. Problems that are $\mathcal{NP}$-hard are often solved via enumeration schemes that prove optimality via converging sequences of primal and dual bounds.

We start with linear programming (LP) as important basis for mixed integer linear programming (MILP). In particular, we establish that solutions can be obtained efficiently as a consequence of optimizing over a convex set. Moreover, we give some basics regarding duality theory that will be required in the later sections. Then, we explain how LP can be used to solve more expressive MILP problems. Afterwards, we discuss techniques to deal with exponentially sized models. These first three sections are mainly based on Bertsimas and Tsitsiklis [17], Nemhauser and Wolsey [130], Schrijver [157], and Wolsey [172]. The fourth section is devoted to Benders decomposition (BD) and in particular logic-based Benders decomposition (LBBD), primarily following Hooker and Ottosson [91]. Finally, we give a short introduction to constraint programming (CP) based on Rossi et al. [153].

2.2.1 Linear Programming

An LP problem (as stated in [17, p. 3]) looks as follows:

$$\begin{aligned}
\min \quad & \mathbf{c}'\mathbf{x} && && (2.1)\\
\text{subject to} \quad & \mathbf{a}_i'\mathbf{x} \geq b_i && \forall i \in M_1, && (2.2)\\
& \mathbf{a}_i'\mathbf{x} \leq b_i && \forall i \in M_2, && (2.3)\\
& \mathbf{a}_i'\mathbf{x} = b_i && \forall i \in M_3, && (2.4)\\
& x_j \geq 0 && \forall j \in N_1, && (2.5)\\
& x_j \leq 0 && \forall j \in N_2. && (2.6)
\end{aligned}$$

Vector $\mathbf{x} = (x_1, \dots, x_n)$ is the set of *decision variables* (or just variables) with $\mathbf{x} \in \mathbb{R}^n$. Multiplied by the cost vector $\mathbf{c} = (c_1, \dots, c_n)$, with $\mathbf{c} \in \mathbb{R}^n$, we obtain the *objective* *function* $c'x$—the linear form of the more general concept introduced in the previous section. Finite disjoint index sets M_1, M_2, and M_3 are associated with coefficient vectors $\mathbf{a}_i \in \mathbb{R}^n$ and scalars $b_i \in \mathbb{R}$. Index set N_1 and N_2 are disjoint subsets of $\{1, \ldots, n\}$ that indicate the subsets of non-negative and non-positive variables, respectively. A variable whose index belongs neither to N_1 nor to N_2 is called *unrestricted* or *free.* Formulas (2.2) to (2.6) are the set of *constraints.* An assignment of the decision variables that satisfied all the constraints is called *feasible solution* (or just solution if clear from the context). The set of all feasible solutions forms the *feasible region* (cf. feasible set as introduced above). A feasible solution $\mathbf{x}^*$ that minimizes the objective function is called *optimal solution* with solution value $\mathbf{c'x}$. When solving an LP problem to optimality, there are four possible outcomes:

(a) The problem is infeasible because the feasible set is empty.

(b) There exists a unique optimal solution.

(c) Multiple optimal solutions exist.

(d) For every real number K there exists a feasible solution with value less than K. In this case we say that the solution value is *unbounded* and associate a cost of $-\infty$.

As already mentioned earlier, maximization problems can be easily transferred to equivalent minimization problems, i.e., $\max \mathbf{c'x} = \min -\mathbf{c'x}$. Moreover, we can replace equality constraints $a_i'x = b_i$ by two inequalities $\mathbf{a}_i'\mathbf{x} \leq b_i$ and $\mathbf{a}_i'\mathbf{x} \geq b_i$. The opposite conversion requires the addition of variables. Thereby we can transform $\mathbf{a}_i'\mathbf{x} \leq b_i$ to $\mathbf{a}_i'\mathbf{x} + s_i = b_i$ and $\mathbf{a}_i'\mathbf{x} \geq b_i$ to $\mathbf{a}_i'\mathbf{x} - r_i = b_i$ with *slack variable* $s_i \in \mathbb{R}$ and *surplus variable* $r_i \in \mathbb{R}$. Free variables x_j can be replaced by the difference of two non-negative variables: $x_j^+ - x_j^-$ with $x_j^+, x_j^- \geq 0$. Finally, constraints of the form $\mathbf{a}_i'\mathbf{x} \leq b_i$ can be equivalently stated as $(-\mathbf{a}_i)'\mathbf{x} \geq -b_i$. This includes in particular the non-negativity restrictions. Consequently, LP problems can be formulated exclusively in terms of constraints of the form $\mathbf{a}_i'\mathbf{x} \geq b_i$. Using matrix notation we can write:

$$\begin{aligned} \min \quad & \mathbf{c'x} && (2.7) \\ \text{subject to} \quad & \mathbf{Ax} \geq \mathbf{b}, && (2.8) \end{aligned}$$

with $\mathbf{A} \in \mathbb{R}^{m \times n}$ a matrix and $\mathbf{b} \in \mathbb{R}^m$ a vector.

Geometrical Properties

In the following we provide a geometrical interpretation for LP problems. We will see that they can be interpreted as multidimensional convex shapes. Moreover, we show that optimization can be restricted to the corners of these shapes which is important for deriving solution algorithms.

Definition 2.13 ([17, p. 42])**.** *For* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *and* $\mathbf{b} \in \mathbb{R}^m$ *a set that can be described as* $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{Ax} \geq \mathbf{b}\}$ *is called* polyhedron*.*

Definition 2.14 ([17, p. 43])**.** *A set* $S \subseteq \mathbb{R}^n$ *is called* bounded *if the absolute value of every component of every element of* S *is bounded by some constant.*

Definition 2.15 ([17, p. 43])**.** *A bounded polyhedron is called* polytope*.*

Theorem 2.3 ([17, p. 44])**.**

(a) Every polyhedron is a convex set.

(b) The convex hull of a finite number of vectors is a convex set.

Definition 2.16 ([17, p. 46])**.** *Let* P *be a polyhedron. Vector* $\mathbf{x} \in P$ *is called* extreme point *of* P *if there do not exist two vectors* $\mathbf{y}, \mathbf{z} \in P$*, both different from* $\mathbf{x}$*, and a scalar* $\lambda \in [0, 1]$ *such that* $\mathbf{x} = \lambda\mathbf{y} + (1 - \lambda)\mathbf{z}$*.*

Definition 2.17 ([17, p. 63])**.** *A polyhedron* $P \subseteq \mathbb{R}$ *contains a* line *if there exists a vector* $\mathbf{x} \in P$ *and a non-zero vector* $d \in \mathbb{R}^n$ *such that* $\mathbf{x} + \lambda\mathbf{d} \in P$ *for all scalars* $\lambda \in \mathbb{R}$*.*

Theorem 2.4 ([17, p. 63])**.** *Given a non-empty polyhedron* P*. Then, the following are equivalent:*

(a) Polyhedron P *has at least one extreme point.*

(b) Polyhedron P *does not contain a line.*

Theorem 2.5 ([17, p. 65])**.** *Consider minimization of* $\mathbf{c}'\mathbf{x}$ *subject to polyhedron* P*. If* P *has at least one extreme point and there exists an optimal solution, then there exists an optimal solution which is an extreme point of* P*.*

Algorithms

One of the most well-known algorithms for solving LP problems is the *simplex method* proposed by Dantzig in 1947, see [46]. Its basic idea is to start at an extreme point of the polyhedron and then to traverse the surface of the polyhedron in cost-reducing direction towards the next extreme point. Observe that the number of extreme points in a polyhedron is finite for a finite number of constraints. Because we are optimizing over a convex set, this guarantees that we eventually obtain an optimal solution with this procedure. Although the number of extreme points is finite, it can still be exponential in the number of variables and constraints. Due to the existence of problems for which all extreme points are visited, the simplex algorithm has exponential worst-case complexity. For the original variant by Dantzig this was shown by the *Klee-Minty cube* in [99]. Despite of the worst-case complexity, simplex algorithms are widely used due to their excellent practical performance.

Khachiyan showed in 1979 that the *ellipsoid method* can solve LP problems in polynomial time, see [98]. This result, however, was mostly of theoretical relevance as no practically efficient implementations of this algorithm could be developed.

Interior point methods finally led to algorithms with polynomial worst-case behavior that are suitable for practical applications. In particular *barrier methods* are known to be effective in practice. Interior point methods date back to the work by Karmarkar [97] in 1984. Opposed to simplex algorithms interior point methods move along the interior of the polyhedron.

Modern state-of-the-art solvers such as IBM ILOG CPLEX Optimizer[1] or Gurobi[2] offer efficient implementations of both simplex and interior point methods. Having several options available can be important in practice as certain problem characteristics can lead to one of the methods performing significantly better.

For the technical details and an in-depth discussion of these LP algorithms we refer to [17], [130], and [157].

Duality

In the following we give a brief introduction to duality theory. Duality gives not only important insights for LP but also serves as basis for some of the methods discussed in the remainder of this chapter.

Let A be a matrix with rows $\mathbf{a}_i'$ and columns $\mathbf{A}_j$. Using the notation introduced at the beginning of this section, we formulate the following pair of LP problems as stated in [17, p. 142]:

$$\begin{array}{rlllrll}
\min & \mathbf{c}'\mathbf{x} & & \qquad & \max & \mathbf{p}'\mathbf{b} & \\
\text{subject to} & \mathbf{a}_i'\mathbf{x} \geq b_i & \forall i \in M_1, & & \text{subject to} & p_i \geq 0 & \forall i \in M_1, \\
& \mathbf{a}_i'\mathbf{x} \leq b_i & \forall i \in M_2, & & & p_i \leq 0 & \forall i \in M_2, \\
& \mathbf{a}_i'\mathbf{x} = b_i & \forall i \in M_3, & & & p_i \text{ free} & \forall i \in M_3, \\
& x_j \geq 0 & \forall j \in N_1, & & & \mathbf{p}'\mathbf{A}_j' \leq c_j & \forall j \in N_1, \\
& x_j \leq 0 & \forall j \in N_2, & & & \mathbf{p}'\mathbf{A}_j' \geq c_j & \forall j \in N_2, \\
& x_j \text{ free} & \forall j \in N_3, & & & \mathbf{p}'\mathbf{A}_j' = c_j & \forall j \in N_3.
\end{array}$$

Thereby we call the minimization problem *primal problem* and the maximization problem *dual* problem. Each constraint of the primal becomes a variable in the dual with the same inequality symbol. Equality constraints transfer to free variables. Each variable becomes a constraint with the opposite inequality symbol. Free variables transfer to equality constraints.

Theorem 2.6 ([17, p. 144])**.** *If we transform the dual problem into an equivalent minimization problem and then form its dual, we obtain a problem equivalent to the initial primal problem.*

---

[1] https://www.ibm.com/analytics/cplex-optimizer (accessed 09/2018)

[2] http://www.gurobi.com (accessed 09/2018)

More compactly this is often stated as “the dual of the dual is the primal”.

Primal and dual are related by two important theorems.

Theorem 2.7 (Weak duality, [17, p. 146])**.** *Let* $\mathbf{c}$ *and* $\mathbf{p}$ *be feasible solutions to the primal and the dual problem, respectively, then*

$$\mathbf{p}'\mathbf{b} \leq \mathbf{c}'\mathbf{x}.$$

This means that, opposed to relaxations, each feasible solution to the dual problem provides a valid dual bound for the primal problem. Observe that unboundedness in either problem implies infeasibility for the other. The reverse implication, however, is not true because both problems might be infeasible.

Theorem 2.8 (Strong duality, [17, p. 148])**.** *If a linear programming problem has an optimal solution, so does its dual, and the respective solution values are equal.*

2.2.2 Mixed Integer Linear Programming

In the previous section we have seen that LP problems can be solved efficiently, i.e., in polynomial time. While the used formalism is quite general, it still misses a rather important feature: integral variables. The benefit of such variables (in particular the binary ones) is that they allow modeling choice, i.e., whether a particular solution component is chosen or not. This, however, comes at a cost: MILP is $\mathcal{NP}$-hard, see, e.g., [66]. This means that in general we cannot expect so solve such problems in polynomial time unless $\mathcal{P} = \mathcal{NP}$. An MILP problem in generic form (see [172, p. 3]) reads as follows:

$$\begin{aligned} \max \quad & \mathbf{c}'\mathbf{x} + \mathbf{h}'\mathbf{y} && (2.9)\\ \text{subject to} \quad & \mathbf{A}\mathbf{x} + \mathbf{G}\mathbf{y} \geq \mathbf{b}, && (2.10)\\ & \mathbf{x} \geq 0, && (2.11)\\ & \mathbf{y} \in \mathbb{Z}^p_{\geq 0}, && (2.12) \end{aligned}$$

with $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{G} \in \mathbb{R}^{m\times p}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{h} \in \mathbb{R}^p$, $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{y} \in \mathbb{R}^p$. If all variables are integer we call the associated problem integer linear programming (ILP) problem.

Definition 2.18 (LP relaxation, [172, p. 25])**.** *Given an MILP problem we obtain the so-called* linear programming relaxation *(LP relaxation) by dropping the integrality restrictions.*

The LP relaxation provides a dual bound to the solution value of its associated MILP problem. However, in general it does not provide a feasible solution due to violating the integrality restrictions.

Definition 2.19 ([172, p. 12])**.** *A polyhedron* $P \subseteq \mathbb{R}^{n+p}$ *is a* formulation *for a set* $X \subseteq \mathbb{R}^n \times \mathbb{Z}^p$ *iff* $X = P \cap (\mathbb{R}^n \times \mathbb{Z}^p)$*.*

Consequently, there exist infinitely many formulations for each MILP problem. Now suppose we are given a formulation P for a (bounded) set X with $\operatorname{conv}(X) = P$. Then it suffices to solve the LP problem for P to obtain an optimal solution for X with respect to a given objective function as all extreme points of the convex hull satisfy the integrality restrictions. We call such formulations *ideal*. The problem, however, is that it is usually difficult to find a formulation equivalent to the convex hull. In most cases such formulations involve a very high, i.e., exponential, number of constraints. This is to be expected from the complexity point of view because a simple characterization (i.e., with a polynomial number of constraints and variables) of the convex hull of an $\mathcal{NP}$-hard problem would imply that $\mathcal{NP} = \mathcal{P}$.

Knowing that an ideal formulation is the best we can hope for, it makes sense to compare formulations with respect to their closeness to the convex hull.

Definition 2.20 ([172, p. 16])**.** *Given a set $X \subseteq \mathbb{R}^n$ and two associated formulations P_1 and P_2. We say that P_1 is at least as strong as P_2 if $P_1 \subseteq P_2$ and stronger if $P_1 \subsetneq P_2$. If neither $P_1 \subseteq P_2$ nor $P_2 \subseteq P_1$ we say that the two formulations are* incomparable.

Corollary 2.1. *Let P_1 and P_2 be two formulations whose optimal solution values to their LP relaxations are z_1 and z_2, respectively. If P_1 is stronger than P_2, then $z_1 \geq z_2$.*

Definition 2.21 ([172, p. 114])**.** *An inequality $\mathbf{a}'\mathbf{x} \geq b$ is called* valid *for set $P \subseteq \mathbb{R}^n$ iff it is satisfied for all $x \in P$.*

According to [172, p. 114] valid inequalities $\mathbf{a_1}'\mathbf{x} \geq b_1$ and $\mathbf{a_2}'\mathbf{x} \geq b_2$ are said to be equivalent if there exists a scalar $\lambda > 0$ such that $(\mathbf{a}_2, b_2) = \lambda(\mathbf{a}_1, b_1)$. If the two valid inequalities are not equivalent and there exists a scalar $\mu > 0$ such that $\mathbf{a}_2 \geq \mu\mathbf{a}_1$ and $b_2 \leq \mu b_1$, then the first inequality is said to be stronger than the second or to dominate it.

Definition 2.22 ([172, p. 141])**.** *A valid inequality $\mathbf{a}'\mathbf{x} \geq b$ is called* redundant *if there exist $k \geq 2$ valid inequalities in the description of a polyhedron P dominating $\mathbf{a}'\mathbf{x} \geq b$. Formally this means there exist inequalities $\mathbf{a}^{i\prime}\mathbf{x} \geq b^i$, for P, and weights $u_i > 0$, $i = 1, \dots, k$, such that $(\sum_{i=1}^{k} u_i \mathbf{a}^{i\prime})\mathbf{x} \geq \sum_{i=1}^{k} b^i$ dominates $\mathbf{a}'\mathbf{x} \geq b$.*

Definition 2.23. *Let $P \subseteq \mathbb{R}^n$ be a formulation for set X and inequality $\mathbf{a}'\mathbf{x} \geq b$ be valid with respect to* $\operatorname{conv}(X)$*. We call the inequality* strengthening *if $P' = \{x \in P \mid \mathbf{a}'\mathbf{x} \geq b\}$ is a formulation for X such that $P' \subsetneq P$.*

Solving MILP problems

The most common way to solve MILP problems is to use a branch-and-bound (B&B) approach (see Land and Doig [103]) in conjunction with an LP solver. The resulting procedure is called LP-based B&B and is summarized in Algorithm 2.1.

We start by solving the LP relaxation. If there exists an integer variable that is fractional in the current relaxed solution, we create two new branches. In one branch we add a

Algorithm 2.1: LP-based branch-and-bound [172, p. 100]

```
problem list $L : \min\{\mathbf{c}'\mathbf{x} \mid \mathbf{x} \in S\}$
$\overline{z} = \infty$                                   // best primal bound
incumbent $\mathbf{x}^* = NULL$                        // best feasible solution
while $L \neq \emptyset$ do
    choose set $S_i$ and remove it from $L$
    obtain optimal LP solution $\mathbf{x}^{i,\mathrm{LP}}$ and its solution value $\underline{z}^i$ for $S_i$
    if $S_i = \emptyset$ then prune $S_i$ by infeasibility
    else if $\underline{z}^i \geq \overline{z}$ then prune $S_i$ by bound
    else if $\mathbf{x}^{i,\mathrm{LP}} \in S$ then                 // LP solution is integral
        if $\underline{z}^i \leq \overline{z}$ then
            update primal bound $\overline{z} = \underline{z}^i$
            update incumbent $\mathbf{x}^* = \mathbf{x}^{i,\mathrm{LP}}$
        end
        prune $S_i$ by optimality
    else
        choose an integer variable $x_j$ that is still fractional
        $S_{i,1} = \{\mathbf{x} \in S_i \mid x_j \leq \lfloor x_j^{i,\mathrm{LP}} \rfloor\}$
        $S_{i,2} = \{\mathbf{x} \in S_i \mid x_j \geq \lceil x_j^{i,\mathrm{LP}} \rceil\}$
        $L = L \cup \{S_{i,1}, S_{i,2}\}$
    end
end
```

constraint that forces the variable to be at most as large as the rounded-down fractional value while the other branch adds a constraint that forces the variable to be at least as large as the rounded-up fractional value. This ensures that the current solution is not encountered again and eventually guarantees termination. Observe that the dual bounds obtained within the search tree are only locally valid, i.e., in the respective subtree. The primal bounds, however, are globally valid. Hence, we store and update the best feasible solution encountered so far. This solution is referred to as *incumbent (solution)*. The algorithm considers three types of pruning that can close a branch. If a subproblem does not contain any feasible solutions, we prune it by *infeasibility*. Based on the (globally valid) primal bound $\overline{z}$ we prune all subproblems with larger dual bound by *optimality* as they cannot contain better solutions. Finally, if we encounter a feasible solution, we update the incumbent if necessary and prune by *optimality*. Subproblems that cannot be pruned have to be split further.

The described procedure leaves several questions open. Most importantly: in which order should the subproblems be considered and which variable should be branched on? Several general strategies exist, however, modern solvers typically do not rely on a single strategy but rather act dynamically depending on the structure of the B&B tree and the subproblems observed so far. In practice, it sometimes makes sense to use problem-specific techniques for selecting the variable to branch on. A common example are problems based on variables that model different structural aspects with cost-coefficients of different order of magnitude, see, e.g., Chapter 6.

2.2.3 Exponentially Sized Models

Up to now we assumed that formulations are small enough—i.e., of (pseudo-)polynomial size—to solve them directly with an appropriate algorithm. However, further modeling options become available when considering models with an exponential number of constraints and/or variables.

The idea is in both cases to start with a reduced problem that considers only part of the exponentially many constraints and/or variables and to dynamically add only those that are required. Such an approach often turns out to be highly efficient when only a small subset of variables and/or constraints is required to support an optimal solution. Constraints that are not restrictive can be omitted similar as variables assigned to zero.

Cutting Plane Methods/Branch-and-Cut

We start by explaining an approach to deal with exponentially many variables in LP problems. At the beginning we consider a relaxation of the original problem by removing some of the constraints. After solving the relaxation to optimality we check whether any of the disregarded constraints are violated in the obtained solution. If no violations are present, we know that this solution must be feasible for the original problem and thus optimal (cf. Theorem 2.1). Otherwise, we identify one or more violated constraints, add them to the relaxed model and start over. Approaches of this type date back to the work of Dantzig et al. [45].

The problem of finding violated constraints for a given relaxed solution is called *separation problem* and the added inequalities are referred to as *cutting planes*, originating from the fact that these inequalities are essentially hyperplanes that cut-off part of the polyhedron of the relaxed problem. An efficient separation algorithm is required to make the described approach successful. In particular, LP problems with a polynomial-time separation routine are still polynomially solvable, see [130].

Cutting planes can also be used for MILP problems. To this end we slightly modify the LP-based B&B. In addition to solving the LP relaxation at each node, we also separate cutting planes exhaustively. The resulting approach is referred to as branch-and-cut (B&C) algorithm. Under certain conditions a modified approach is used that delays (part of) the separation to solutions that satisfy the integrality restrictions. This can be beneficial if too many inequalities are separated for fractional solutions that are redundant for integral solutions.

We also want to point out that cutting planes are not only used to deal with exponentially-sized families of inequalities that are required to enforce feasibility. In practice also strengthening inequalities are often separated dynamically. They lead to tighter LP relaxations that speed up the B&B process. Moreover, sometimes even redundant inequalities are considered. Often they have a strong impact on the formulation (although not being sufficient to guarantee feasibility) and thereby help to improve convergence. Modern solvers typically provide several families of general purpose inequalities that are problem-independent and separated by default. Therefore, such solvers actually have to be considered B&C approaches, even if no user-defined separation is performed.

Column Generation/Branch-and-Price

Dynamically adding variables to LP problems is called *(delayed) column generation* and was first considered in Gilmore and Gomory [73, 74]. The name of this method comes from the fact that when viewing LP (or MILP) problems in block form, each column corresponds to a variable and each row to a constraint. For the cutting planes it was somehow obvious that we want to separate those constraints that are violated in the current relaxation. Deriving a similar scheme for the variables might not seem straightforward at first. One difference is that we require a sufficient subset of variables to guarantee the existence of a feasible solution in the initial iteration. Opposed to the cutting planes we are not dealing with a relaxation that provides a dual bound but rather with heuristic solutions that provide primal bounds. Therefore, suitable starting variables can often be identified by computing a potentially suboptimal solution by means of a simple constructive heuristic, see Section 2.3, and adding the variables necessary to encode that solution. A generic alternative would be, e.g., *Farkas pricing*, see [63]. The partial model for which we incrementally incorporate further variables is called *restricted master problem*.

In the iterative step we aim at identifying variables that have the potential to improve the current solution. Observe that this is the dual concept of cutting planes. The cutting planes approach considers an initially infeasible solution and attempts to converge towards feasibility. Column generation, on the other hand, starts with a potentially suboptimal solution and works towards optimality.

Definition 2.24 ([17, p. 84])**.** *The* reduced costs *of variable x_j are defined as*

$$\bar{c}_j = c_j - \mathbf{p}'\mathbf{A}_j$$

where $\mathbf{p}$ is the dual variable vector and $\mathbf{A}_j$ the j-th column of the coefficient matrix $\mathbf{A}$.

Informally speaking, the reduced costs indicate the cost change for a unit increase in the associated variable's value. Therefore, we seek those variables with negative reduced cost because they may reduce the solution value if included (with non-zero value) in the next iteration's solution. Note that the dual solution vectors are readily available in modern solver implementations. The problem of identifying variables with negative reduced cost is called *pricing subproblem* and typically seeks a variable with minimum reduced cost. If the minimum reduced cost is non-negative, then we cannot improve the objective any further and the current solution must be optimal. Depending on the problem at hand it might be useful to consider adding multiple variables per iteration if there are, e.g., variables related to different aspects of the problem that are priced independently.

If column generation is done within the B&B tree to solve MILP problems we refer to the associated approach as branch-and-price (B&P). Thereby, variables are priced exhaustively at each node to ensure that the solution values obtained from the respective LP relaxations represent valid dual bounds.

Finally, the concepts of column generation and cutting planes can be combined to deal with exponentially many variables and constraints. To this end one typically starts by pricing variables until none with negative reduced cost remain. Then, one iteration of separating cutting planes is performed. If no violated inequalities are identified, the approach terminates with an optimal solution. Otherwise, the violated inequalities are added to the restricted model and we repeat the process by pricing further variables. When done within the B&B tree for MILP problems, we refer to this approach as branch-price-and-cut (BP&C).

2.2.4 Benders Decomposition

We start by explaining the recently introduced LBBD by Hooker and Ottosson [91], which we use in Chapter 4. It includes the classical BD as a special case, details are provided below. The idea of BD is to solve large problems based on variables that can be partitioned into two subsets $(\mathbf{x}, \mathbf{y})$ such that the problem separates into one or more easier solvable subproblems on the x variables after fixing the y variables.

The general shape of these problems is the following, where $\mathbf{S}$ is the set of feasible solutions (typically formulated via a collection of constraints) and $\mathbf{D_x}, \mathbf{D_y}$ are the domains of $\mathbf{x}$ and $\mathbf{y}$, respectively:

$$\begin{aligned} \min \quad & f(\mathbf{x}, \mathbf{y}) && (2.13) \\ \text{subject to} \quad & (\mathbf{x}, \mathbf{y}) \in \mathbf{S}, && (2.14) \\ & \mathbf{x} \in \mathbf{D_x}, && (2.15) \\ & \mathbf{y} \in \mathbf{D_y}. && (2.16) \end{aligned}$$

The idea is to decompose the problem into a master problem only using the y variables and depending subproblems expressed on the x variables. The master problem is obtained by removing all elements containing x variables. Instead, their contributions are modeled through additional inequalities (2.18), called *Benders cuts*:

$$\begin{aligned} \min \quad & z && && (2.17) \\ \text{subject to} \quad & z \geq \beta_{\mathbf{y}^k}(\mathbf{y}) && \forall k \in K, && (2.18) \\ & \mathbf{y} \in \mathbf{D_y}. && && (2.19) \end{aligned}$$

In the above model the new variable z corresponds to the original objective function $f(\mathbf{x}, \mathbf{y})$ and is now determined by the Benders cuts. To solve this model, one starts with a reduced master problem containing no (or only a small initial set of) Benders cuts, yielding a solution $\bar{\mathbf{y}}$. When considering again the original problem (2.13–2.16) and fixing the y variables to $\bar{\mathbf{y}}$, we obtain the following subproblem, solely defined on the x variables and supposed to be in some sense much easier to solve than the original problem:

$$\begin{aligned} \min \quad & f(\mathbf{x}, \bar{\mathbf{y}}) && (2.20) \\ \text{subject to} \quad & (\mathbf{x}, \bar{\mathbf{y}}) \in \mathbf{S}, && (2.21) \\ & \mathbf{x} \in \mathbf{D}_{\mathbf{x}}. && (2.22) \end{aligned}$$

The *inference dual* of the subproblem is defined as follows:

$$\begin{aligned} \max \quad & \beta && (2.23) \\ \text{subject to} \quad & (\mathbf{x}, \bar{\mathbf{y}}) \xrightarrow{\mathbf{D}_{\mathbf{x}}} f(\mathbf{x}, \bar{\mathbf{y}}) \geq \beta. && (2.24) \end{aligned}$$

Considering the dual of the subproblem we want to find the best possible dual bound β^* on the optimal solution value, when $\mathbf{y}$ is fixed to $\bar{\mathbf{y}}$, that can be inferred from the constraints. The main challenge is to identify a bounding function $\beta_{\bar{\mathbf{y}}}(\mathbf{y})$ providing a valid dual bound on the optimal objective value of (2.13) given any fixed value $\mathbf{y}$.

If all subproblems are feasible, they yield a valid dual bound $\beta_{\bar{\mathbf{y}}}(\mathbf{y})$ with respect to the current assignment of the y variables. So-called Benders *optimality cuts* are derived from the dual solution and are added to incorporate this information into the master problem. If one or more subproblems turn out to be infeasible, this means that the considered solution on the y variables is not acceptable. This information is also communicated to the master problem by means of Benders cuts which are then called *feasibility cuts*. In particular, since infeasible subproblems have unbounded dual, we obtain $\beta_{\bar{\mathbf{y}}}(\bar{\mathbf{y}}) = \infty$. With this additional information we continue and search for a better assignment of the y variables by solving the augmented master problem and the corresponding subproblems again. This procedure is iterated until it is no longer possible to find an assignment of the y variables that improves the objective and no infeasible subproblems remain. Then, an optimal solution to the initial problem has been found. We denote by $\beta_{\bar{\mathbf{y}}^k}(\mathbf{y})$ the bounding function inferred from the k^{th} trial value $\bar{\mathbf{y}}^k$. According to Hooker and Ottosson [91] it is known that if in every iteration k of the Benders algorithm

- the Benders cut $z \geq \beta_{\bar{y}}(\mathbf{y})$ is valid, i.e., any feasible solution $(\mathbf{x}, \mathbf{y})$ to (2.13) satisfies $f(\mathbf{x}, \mathbf{y}) \geq \beta_{\bar{\mathbf{y}}}(\mathbf{y})$ and
- $\beta_{\bar{\mathbf{y}}^k}(\bar{\mathbf{y}}^k) = \beta$ where β is the optimal solution to the dual of (2.20),

then, given finite domain $\mathbf{D}_{\mathbf{y}}$ for the y variables and if the subproblem dual is solved to optimality, the generic Benders algorithm terminates with the correct result.

It is often the case that not all x variables are linked to each other after the y variables have been fixed. In these cases it is possible to split the subproblem into smaller problems that can be solved independently. A prominent example for this case are problems whose constraint matrix has dual block-angular structure, see Figure 2.1. This structure is the dual to the one arising and exploited in *Dantzig-Wolfe decomposition* [47]. Having independently solvable, smaller and/or easier subproblems is one of the advantages of BD as this often helps to deal with much larger problem instances than is possible when solving the problem as a whole.

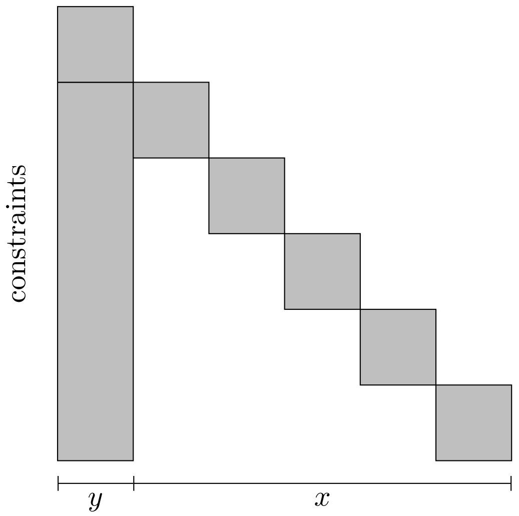


Figure 2.1: Dual block-angular structure of a constraint matrix: A set of y variables links x variables that could otherwise be considered independently. Gray areas denote non-zero coefficients.

In the original BD, introduced by Benders [14], the subproblems are restricted to be LP problems. This makes it possible to derive Benders cuts by means of classical duality theory. Let

$$\begin{aligned} \min \quad & \mathbf{c}'\mathbf{x} + f(\mathbf{y}) && (2.25)\\ \text{subject to} \quad & \mathbf{A}\mathbf{x} + \mathbf{g}(\mathbf{y}) \geq \mathbf{b}, && (2.26)\\ & \mathbf{x} \in \mathbb{R}^n_{\geq 0}, \mathbf{y} \in \mathbf{D}_{\mathbf{y}}, && (2.27) \end{aligned}$$

be the initial problem with $\mathbf{g}(\mathbf{y})$ a vector of functions $g_i(\mathbf{y})$. After fixing the y variables to some trial value $\bar{\mathbf{y}}$, we obtain the subproblem

$$\begin{aligned} \min \quad & \mathbf{c}'\mathbf{x} + f(\bar{\mathbf{y}}) && (2.28)\\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} - \mathbf{g}(\bar{\mathbf{y}}), && (2.29)\\ & \mathbf{x} \in \mathbb{R}^n_{\geq 0}, && (2.30) \end{aligned}$$

and its dual

$$\max \quad \mathbf{u}'(\mathbf{b} - \mathbf{g}(\bar{\mathbf{y}})) + f(\bar{\mathbf{y}}) \qquad (2.31)$$

$$\text{subject to} \quad \mathbf{u}'\mathbf{A} \leq \mathbf{c}', \tag{2.32}$$

$$\mathbf{u} \in \mathbb{R}^n_{\geq 0}. \tag{2.33}$$

If the subproblem is feasible with finite solution $\mathbf{u}$, we obtain an optimality cut of the form $z \geq \mathbf{u}'(\mathbf{b} - \mathbf{g}(\mathbf{y})) + f(\mathbf{y})$. Observe that this cut stays valid for any trial value of the master problem and is tight for the current one. For infeasible or unbounded duals we obtain a Benders feasibility cut of the form $\mathbf{v}'(\mathbf{b} - \mathbf{g}(\mathbf{y})) \leq 0$ where $\mathbf{v}$ is a ray that solves

$$\max \quad \mathbf{v}'(\mathbf{b} - \mathbf{g}(\bar{\mathbf{y}})) \tag{2.34}$$

$$\text{subject to} \quad \mathbf{v}'\mathbf{A} \leq \mathbf{0}', \tag{2.35}$$

$$\mathbf{v} \in \mathbb{R}^n_{\geq 0}. \tag{2.36}$$

Geoffrion showed in [72] how to extend this method to other convex optimization methods using nonlinear convex duality theory. This allows for a systematic generation of the bounding function by means of duality theory. Unfortunately, this also limits the applicability of the approach.

The advantage of LBBD is that it permits more general subproblems. However, in contrast to the traditional BD there exists no (single) systematic way to identify a strong bounding function for the Benders cuts. Instead, tailored cuts have to be identified with respect to the encountered subproblems.

Despite of being a relatively new approach LBBD has been applied effectively in several areas including planning and scheduling (Hooker [90], Hamdi and Loukil [85]), location problems (Fazel-Zarandi and Beck [59], Wheatley et al. [170]), survivable network design (Garg and Smith [68]), and vehicle routing (Cire and Hooker [38], Raidl et al. [145, 146]).

Branch-and-Check

The idea behind classical BD and LBBD is to solve the master problem to optimality in each iteration. However, this might not always be necessary. Suboptimal solutions can be sufficient to derive relevant cuts for the master problem. In particular, one can consider a single B&C tree and separate Benders cuts for all identified intermediate (integral) solutions. In terms of LBBD this idea was first introduced in Hooker [89] and further examined in Thorsteinsson [161] and is also closely related to the concept of *combinatorial Benders cuts* considered by Codato and Fischetti [40]. Thorsteinsson [161] referred to this strategy as Branch-and-Check (BaC). We will adopt this term in the following. The comparable strategy in the context of classical BD is often referred to as Branch-and-Benders-cut, see Rahmaniani et al. [144].

Using the terminology introduced for BD, BaC specifies a single problem defined only on the y variables together with their constraints. This problem is then solved. Whenever a feasible solution is encountered within the B&C tree the corresponding subproblems are derived and solved. Depending on their solutions Benders cuts are added, possibly cutting off the current solution. The main difference to LBBD is that the master problem is solved only once and that Benders cuts are typically generated with respect to suboptimal assignments of the y variables inside the B&C tree.

Technically, traditional B&C and BaC are very similar. The main difference is that B&C separates cuts for LP solutions whereas BaC separates cuts only for feasible (integral) solutions.

When dealing solely with feasibility cuts, i.e., the solution of the subproblems has no influence on the objective of the master problem, BaC has one advantage over LBBD that really stands out. Since BaC operates on the B&B tree and iteratively approaches the feasible area, it is usually capable of finding feasible solutions quite fast. LBBD, on the other hand, either terminates with an optimal solution or no feasible solution at all. Often it is possible to derive a feasible solution from the trial values of the intermediate Benders iterations, however, this requires additional computational effort which is not incurred when using BaC.

2.2.5 Constraint Programming

Similar to MILP, CP is also based on a set of decision variables and a set of associated constraints. However, it is not restricted to linear constraints. This somehow reflects the fact that CP is particularly designed for solving problems that are tightly constrained. Often also constraint satisfaction problems (CSPs) are considered, i.e., problems without an objective function for which the goal is to obtain any feasible solution.

The number of available constraint types is quite large and solver-dependent, so listing them exhaustively would not be meaningful. Therefore, we give a few examples of popular constraints instead and refer the interested reader to the *global constraint catalog*, see https://sofdem.github.io/gccat/gccat/sec5.html. One of the most prominent constraints is the `all_different` constraint, which states that for a given set of variables no two of them may take the same value. Another important constraints is the `element` constraint. Given variable i and a table of values T, `element(T,i)` provides the value at position i in the table, for brevity also often written as T_i. It might not seem powerful at first but is required rather frequently, in particular, as it can also be nested. Another basic modeling tool is the `domain` constraint. It specifies for a given variable a lower and an upper bound on the values it may take. Different from MILP, variable domains do not have to be contiguous but can be arbitrary sets, e.g., specified via the `in` constraint. Integral domains are typically preferred due to a higher number of available constraints and better efficiency of the employed solution method. Nevertheless, also (intervals of) fractional values are possible.

The solution method for CP is primarily based on (1) *constraint propagation* and (2) tree search. The basic idea of constraint propagation is to remove from each variable's domain all values that cannot be part of a feasible solution when considering the specified constraints. Different levels of consistency can be achieved via constraint propagation. The simplest one is *node consistency*, which is considered for constraints involving only a single variable. To achieve node consistency we remove all values incompatible with the unary constraint and afterwards the constraint itself. Hence, the complexity of achieving node consistency is linear in the number of variables and the cardinality of their domains. A more complex concept is *arc consistency.* For a binary constraint (i.e., involving two variables) arc consistency enforces that for each value of the first variable's domain there must exist a value in the second variable's domain satisfying the constraint and vice versa. Domain values that are not consistent with this requirement are removed. Different algorithms have been suggested to achieve arc consistency. They rely on the fact that binary CSPs can be represented by directed graphs. Thereby, each node of the graph represents a variable and each arc a constraint relating two variables. The corresponding algorithms now traverse this graph and successively reduce the variable domains, also called *filtering.* Over the years several algorithms with increasing efficiency have been developed. Among the most well-known algorithms is the AC-3 algorithm by Mackworth [119] that achieves arc consistency in time $O(ed^3)$ and space $O(e)$ where e is the number of edges in the graph and d the maximum cardinality among the variable domains. Improved time-efficient is possible, however at the expense of considerably higher space requirements, see, e.g., Mohr and Henderson [127] and van Hentenryck et al. [166]. Higher order consistencies exist but are rarely used in practice due to their increasing complexity.

Once constraint propagation for a certain consistency level finished, there are three possible outcomes. If there exists at least one variable with empty domain, then the CSP is infeasible. Should all variable domains be restricted to exactly one value, we identified a feasible solution. In most situations we will end up in the third case in which none of the variable domains is empty and some variable domains still contain more than one value. To obtain a feasible solution or prove infeasibility, we need to continue. This is done by a tree search that selects one of the variables to branch on. Infeasibility can only be proven by closing all branches but to show feasibility it suffices to find a feasible solution in any branch. If an optimization problem is considered, then a B&B scheme with respect to the encountered solution values has to be employed and all branches have to be closed to prove optimality. Similar as for MILP this involves several non trivial decision such as selecting the variable to branch on and deciding how the branching is performed. Efficient solvers are available, e.g., IBM ILOG CPLEX CP Optimizer[3] or Gecode[4].

For further details we refer to Rossi et al. [153].

2.3 Heuristic Methods

The heuristic methods presented in the following can, e.g., be found in Gendreau and Potvin [70] and Blum and Raidl [23]. Opposed to exact approaches heuristics provide no quality guarantees on the computed solutions, i.e., the difference to the optimal solution value might be arbitrarily large. Their main advantage is that by giving up optimality,

---

[3] https://www.ibm.com/analytics/cplex-cp-optimizer (accessed 09/2018)

[4] https://gecode.github.io (accessed 09/2018)

solutions can be computed much faster. This allows obtaining solutions also for very large problem instances that cannot be solved by exact approaches in a reasonable amount of time. Despite of providing no theoretical performance guarantees, more sophisticated types of heuristics such as metaheuristics often provide near optimal solutions. Therefore, heuristics can be valuable on their own but may also act as components of other algorithms. When considering a B&B approach to solve MILP problems, a high-quality starting solution can be helpful to enable stronger pruning in the tree.

2.3.1 Constructive Heuristics

Constructive heuristics (also construction heuristics) are in some sense the most basic variant of heuristics. As the name suggests, their idea is to construct a solution step-by-step. A well-known variant are so-called *greedy heuristics*. They start with an empty solution and in each step apply a locally optimal extension until a complete solution is obtained. The extension step is typically problem-dependent but in most situations rather easy to come up with. Consider, e.g., a simple tour planning problem such as the traveling salesman problem (TSP): A so-called *nearest neighbor* heuristic adds in each step a connection to the closest not yet visited city.

2.3.2 Metaheuristics

Metaheuristics are general problem-independent algorithms that typically provide solutions of much higher quality than simple constructive heuristics. Some of these algorithms belong to the class of so-called *improvement heuristics*, which require a starting solution. In the following we discuss two simple variants of metaheuristics that are used in this work. Many more established algorithms of this type exist. Well-known examples include simulated annealing, tabu search, variable neighborhoods search, genetic algorithms, and ant colony optimization. For a broad overview see, e.g., Gendreau and Potvin [70].

Local Search

Local search is an improvement heuristic and requires a starting solution, e.g., provided by a constructive heuristic. The idea is to compute a solution that is locally optimal according to a specific neighborhood. As shown in Algorithm 2.2 this is done by iteratively moving to better neighboring solution. For smaller neighborhoods this procedure can be iterated until none of the possible neighbors provides an improvement. Larger neighborhoods sometimes cannot be searched exhaustively. This makes it necessary to terminate according to some other criterion like a time limit, a specific number of iterations, or a number of iterations without improvement. If a dual bound is known, termination can also be based on the proximity to this bound.

For choosing a neighbor, there are typically several options available. Common strategies are *random neighbor*, *next improvement*, and *best improvement*, which choose a random, the first improving, or the neighbor with the best solution value within the neighborhood of the current solution, respectively.

Algorithm 2.2: Local search

```
Input: initial solution $x'$, cost function $c$, neighborhood function $N$
incumbent $x^* \leftarrow x'$                    // best feasible solution
repeat
    choose $x \in N(x^*)$
    if $c(x) \leq c(x^*)$ then
        $x^* \leftarrow x$
    end
until stopping criteria satisfied
```

Greedy Randomized Adaptive Search Procedure

The greedy randomized adaptive search procedure (GRASP) has been proposed by Feo and Resende [60, 61] and is based on the principle of randomizing an underlying construction heuristic to obtain diverse solutions that are optimized by local search, see Algorithm 2.3. The idea is that starting from different solutions allows the local search to reach different local optima. This decreases the chances of ending up in a particularly bad local optimum. The used neighborhoods are typically assumed small enough so that local optima can be computed for each starting solution.

Algorithm 2.3: Greedy randomized adaptive search procedure

```
Input: cost function $c$
Input: randomized greedy heurstic RG, local search algorithm LS
incumbent $\mathbf{x}^* \leftarrow NULL$                    // best feasible solution
repeat
    $\mathbf{x} \leftarrow \mathrm{RG}()$
    $\mathbf{x}^1 \leftarrow \mathrm{LS}(\mathbf{x})$
    if $c(\mathbf{x}^1) \leq c(\mathbf{x}^*)$ then
        $\mathbf{x}^* \leftarrow \mathbf{x}^1$
    end
until stopping criteria satisfied
```

A common way of randomizing the construction process are so-called *restricted candidate lists*. Thereby, we consider only a subset of the possible options to extend the current partial solution in each step. Among the selected candidates we choose randomly. The restricted candidate list can be constructed by selecting a specific number of the cheapest extensions or all extension within a specific range of the best extension. Several other options and general extensions to the basic GRASP such as biased selection from the restricted candidate list can be considered, see Resende and Ribeiro [147].

2.4 Hybrid Methods

The idea of hybrid methods is to combine different solution approaches to benefit from their strengths by covering their individual weaknesses. An example was already given above: The convergence of a B&B algorithm for MILP can possibly be accelerated by providing an initial solution computed by a heuristic. Going a step further one can apply heuristics at each node of the search tree by using the solutions to the LP relaxations as guidance. Modern solvers for MILP use general purpose heuristics to derive feasible solutions, e.g., by rounding fractional values in solutions to LP relaxations.

In general there exists a large variety of hybrid methods that can be classified according to the components that are chosen and the way in which the individual techniques are combined. For an extensive discussion we refer to Blum and Raidl [23].

In the following we want to focus on a specific type of hybrid methods: *matheuristics*. Matheuristics are essentially a combination of mathematical programming techniques (such as LP and MILP) and metaheuristic approaches. According to Maniezzo et al. [121] there exist two main variants: The first one aims to improve the metaheuristic by exploiting mathematical programming techniques. The second one aims to improve the mathematical programming technique with the robustness and time efficiency of the metaheuristic. In this work we are primarily interested in the latter that typically leads to an exact algorithm.

As a typical example consider the following, see also Puchinger et al. [139]. Sometimes separation problems or pricing subproblems are too difficult to (always) solve them exactly. In this situation one can use a strong heuristic instead. As long as the heuristic provides violated inequalities or columns with negative reduced cost, everything proceeds as usual. However, at some point the heuristic will stop finding further extensions to the model. To preserve optimality we have to verify with an exact algorithm that there are indeed no remaining constraints that are violated or columns with negative reduced cost. If we find further extensions in this way, the heuristic can be applied again afterwards until the next exact iteration becomes necessary. In case of column generation we may alternatively skip this step and settle for a heuristic solution. If we are dealing with cutting planes that are required to achieve feasibility, simply terminating could result in an infeasible solution. In some situations this can be resolved by a subsequent repair process. For cutting planes that are not required to enforce feasibility, exhaustive separation is not necessary, so we are still guaranteed to obtain an optimal solution.

A technique we use in this work are matheuristics that derive converging sequences of primal and dual bounds. Thereby, heuristic techniques are used to obtain primal bounds and relaxations of MILP models are used to obtain dual bounds. By successively improving the quality of the relaxation we strengthen the dual bound until it becomes tight. At the same time we use the refined relaxation as guidance for the heuristic component to obtain better incumbent solutions. Having a sequence of gradually improving primal solutions is particularly important for practical applications. If the algorithm is terminated due to a time limit before proving optimality, then it provides at least a good solution whose quality correlates with the time spent. Opposed to a pure (meta-)heuristic we also obtain a dual bound that makes it possible to assess how close we were to proving optimality, establishing confidence in the obtained solution's value.

CHAPTER 3

Exact Approaches for Network Design Problems with Relays

In this chapter we consider mixed integer linear programming (MILP) models for the network design problem with relays (NDPR). This problem considers the construction of communication networks subject to distance restrictions. If signals have to be transmitted beyond a certain limit, signal regeneration equipment (relays) has to be installed at intermediate network nodes. By means of a *communication graph* we model all paths that can be traversed without the use of relays, i.e., that are no longer than the distance limit. Valid solutions can be represented by collections of such paths with relays installed at all nodes where paths are joined. As the number of such links can become very large but only a small subset is expected to be relevant in practice, we resort to column generation for increased efficiency. Moreover, we consider a branch-price-and-cut (BP&C) approach that uses also an exponential umber of constraints to enforce connectivity requirements.

Opposed to previous work on the NDPR which mainly focused on heuristic approaches, our MILP formulations provide provably optimal solutions. In an extensive computational study, we analyze the performance of these approaches for instances that reflect different real-world settings. Finally, we also point out the relevance of the NDPR in the context of electric mobility.

This chapter has been accepted for publication in the *INFORMS journal on Computing*:

> M. Leitner, I. Ljubić, M. Riedler, and M. Ruthmair. Exact approaches for network design problems with relays. *INFORMS Journal on Computing*, To appear, 2018

3.1 Introduction

Cabral et al. [30] introduced the NDPR for modeling the design of telecommunication networks when the maximum distance a commodity (i.e., signal) can travel is bounded from above by some threshold and when distances exceeding this limit can be covered by locating special, commodity regenerating equipment (*relays*) at intermediate locations. Well-known applications in communication networks arise from signal deterioration and thus there is the need to regenerate them after some maximum distance by using rather expensive devices (e.g., repeaters), see, e.g., [30]. Without regeneration, the transmitted information might be falsified or the signal might be lost. As regeneration devices are usually expensive the goal is to use as few devices as possible, see [35]. As an alternative to placing relays the distance along certain connections may also be reduced by installing additional edges.

Problem Definition

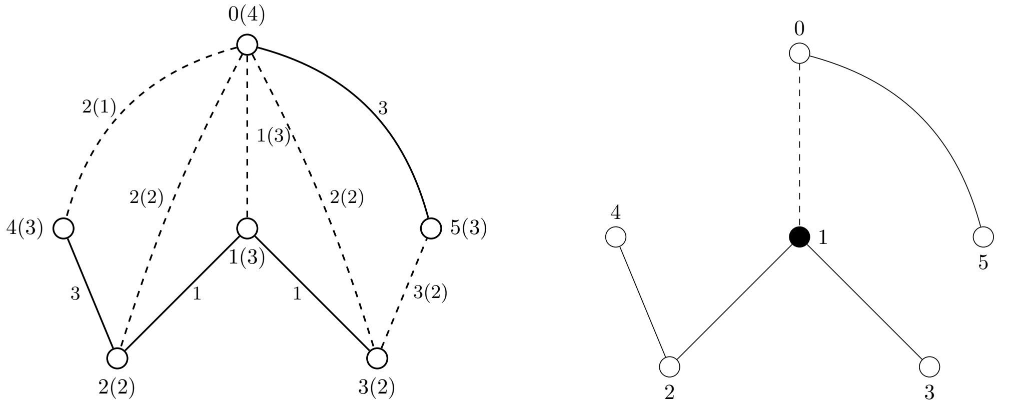
The NDPR is defined on an undirected graph $G = (V, E, c, w, d)$ with relay costs $c\colon V \to \mathbb{N}_{>0}$, edge costs $w\colon E \to \mathbb{N}_{\geq 0}$, and edge lengths $d\colon E \to \mathbb{N}_{\geq 0}$. The edge set E is the disjoint union of the set of free (e.g., existing) edges $E^0 = \{e \mid w(e) = 0\}$ and the set of augmenting edges $E^* = \{e \mid w(e) > 0\}$. Furthermore, $\mathcal{K} \subseteq V \times V$ is the set of commodities. Parameter $d_{\max} \in \mathbb{N}_{>0}$ defines the maximum distance a commodity can traverse without regeneration.

The NDPR consists of selecting augmenting edges $\hat{E} \subseteq E^*$ to install and nodes $\hat{V} \subseteq V$ where relays are to be placed minimizing the resulting costs $\sum_{i \in \hat{V}} c_i + \sum_{e \in \hat{E}} w_e$. A solution is feasible iff all commodity pairs from $\mathcal{K}$ can communicate using the edges in $\hat{E} \cup E^0$ and relays at nodes $\hat{V}$. Thereby, two distinct nodes $s, t \in V$ can communicate if there exists a walk $W = (s = v_0, v_1, v_2, \ldots, v_k = t)$, $v_i \in V$, $0 \leq i \leq k$, that does not contain a subwalk $W' = (v_l, v_{l+1}, \ldots, v_{l+m})$, $0 \leq l \leq k-1$, $m \geq 0$, $l + m \leq k$, whose length $d(W') = \sum_{j=l}^{l+m-1} d_{j,j+1}$ is greater than $d_{\max}$ and which does not contain a relay at an intermediate node, i.e., $v_j \notin \hat{V}$ for $l < j < l + m$. A walk satisfying these conditions is called a feasible walk and a feasible path is defined analogously.

An example of an NDPR instance together with an optimal solution is given in Figure 3.1.

Outline and Discussion of the Contributions

Available literature for the NDPR mainly deals with heuristic approaches (see, e.g., [30, 102, 100]). On the contrary, we provide a comprehensive computational study of MILP models and underlying exact algorithms for the NDPR. A common characteristic of the presented MILP models is that they require an exponential number of variables representing paths in a so-called *communication graph*. The communication graph contains the same nodes as the original graph G and an edge between every two nodes that can be connected via a feasible path without installing relays.



(a) Instance (b) Optimal solution

Figure 3.1: An NDPR instance and its optimal solution for $d_{\max} = 4$ and $\mathcal{K} = \{(0,3),(0,4),(2,5),(3,4)\}$ in which a relay is installed at node 1 and the augmenting edge $\{0,1\}$ is selected. Nodes are labeled by node index and relay installation costs in parentheses. Edges are labeled by their lengths and installation costs in parentheses (for augmenting edges). Solid lines indicate free edges and dashed lines augmenting edges. Black nodes mark the selected relays in the solution.

For deriving computationally viable optimization tools for the proposed MILP models, we implemented two BP&C algorithms. Their performance is assessed on a large set of benchmark instances—some of them taken from the available literature, and some newly generated ones.

The chapter is organized as follows. In the remainder of this section we summarize our notation and provide an overview of the related literature. In Section 3.2 we prove some structural properties of feasible/optimal solutions that will help us to create tighter formulations. In Section 3.3 we discuss transformations of the input graph and associated MILP formulations. Section 3.4 provides the details of our algorithmic framework followed by the presentation of our computational results on a diverse family of benchmark instances in Section 3.5. Finally, we discuss challenges and open questions for further research in Section 3.6 where we also summarize our main conclusions.

3.1.1 Notation and Assumptions

To ease notation in many of the results and formulations introduced in the following, we will consider the node pairs (i.e., commodities) to be directed pairs $(u,v) \in \mathcal{K}$. This assumption is without loss of generality, since there are no costs nor capacities associated with the routing decisions in the NDPR. Furthermore, we define the set of *sources* $S = \{u \mid \exists v \in V \text{ such that } (u,v) \in \mathcal{K}\}$ and the set of targets $T^u = \{v \mid (u,v) \in \mathcal{K}\}$ that have to be reached by source $u \in S$. Similarly, the set of all *targets* is defined as $T = \{v \mid \exists u \in V \text{ such that } (u,v) \in \mathcal{K}\} = \bigcup_{u \in S} T^u$.

Let $\delta(W) = \{\{i,j\} \in E \mid i \in W, j \in V \setminus W\}$ denote the set of edges incident to $W \subseteq V$ in the undirected graph $G = (V, E)$. For a directed graph $G' = (V', A')$ and node subset $W \subseteq V'$, we define $\delta^+(W) = \{(i,j) \in A' \mid i \in W, j \in V \setminus W\}$ and $\delta^-(W) = \{(i,j) \in A' \mid i \in V \setminus W, j \in W\}$ as the set of outgoing and incoming arcs, respectively.

Finally, observe that edges $e \in E$ such that $d_e > d_{\max}$ as well as commodities $(u,v) \in \mathcal{K}$ for which G contains a feasible path between the endpoints, using free edges from E^0 only and no relays, can be removed in a preprocessing step. Thus, we assume without loss of generality that neither of them exists in the given input.

3.1.2 Related Work and Applications in Transportation

The NDPR was introduced by Cabral et al. [30] who showed that the problem is $\mathcal{NP}$-hard and described heuristic approaches intended to solve large instances. Furthermore, an MILP formulation of the problem based on an exponential number of variables is described. Contrary to our path variables derived in the communication graph (cf. Section 3.3.1), Cabral et al. [30] use variables representing entire walks between the commodities in the original graph (including the placement of relays). Their formulation is used to derive lower bounds by means of column generation. At the same time, the subset of generated columns is used to compute heuristic solutions in a subsequent branch-and-bound (B&B) phase. Computational results are discussed for instances with up to 62 nodes, 103 edges and 10 commodities.

A hybrid metaheuristic combining a genetic algorithm with local search has been proposed by Kulturel-Konak and Konak [102] whereas an improved genetic algorithm is given in Konak [100]. In the latter article, Konak also introduces a variant of the MILP formulation by Cabral et al. [30] by using separate variables to represent walks and relay placements. However, no computational studies concerning this model were conducted. Instead, some observations regarding the set covering constraints introduced in this formulation are used in the design of the proposed genetic algorithm. Computational results are given on instances with up to 160 nodes and 3624 edges and 10 commodities. Lin et al. [113] proposed a tabu search approach for the NDPR. Their solution method computes solutions of almost as good quality as the genetic algorithm from Konak [100] but requires less computation time. Very recently, Xiao and Konak [173] presented a variable neighborhood search for the NDPR that is combined with an exact algorithm for the relay placement. Independently from our work an alternative branch-and-price (B&P) approach was developed in Yıldız et al. [177]. The authors propose MILP formulations on a so-called virtual network with an exponential number of edges. Their computational study mainly focuses on a tree formulation specifically designed for the single source case, i.e., $|S| = 1$.

A related version of the NDPR, defined on a directed graph and called the directed network design problem with relays (DNDPR) (see also Chapter 6), has been introduced in Li et al. [110] where a compact MILP model and a B&P algorithm have been proposed.

The problem definition is however slightly different from the NDPR: in the DNDPR only simple paths are allowed for connecting commodity pairs, whereas the NDPR also allows using walks. In fact, allowing multiple node visits in general makes NDPR solutions significantly cheaper than DNDPR solutions, see Section 3.2 for further details. The compact formulation given in [110] exploits the fact that, along a path connecting a commodity pair, each node can be visited at most once. More precisely, it uses a single variable per node to record the distance from the source at which it is reached. Consequently, this formulation cannot be used for solving the NDPR. Their B&P algorithm, which is very similar to the one considered by Cabral et al. [30], is in general capable of allowing cycles but in the computational experiments the authors only consider the acyclic case. Finally, [96] study an extension of the NDPR which considers survivability, edge capacities, and allows for k-splitting of the routes of a commodity.

Application in Transportation

Yıldız and Karaşan [174] provide a detailed survey on applications of relay placement problems in transportation. In some of these applications dealing with hub location issues, hubs are interpreted as relays, i.e., the location of hubs naturally corresponds to the placement of relays. The underlying problems are concerned with identifying physical locations of hubs which may serve as places for the exchange of drivers, trucks and trailers, or as stations where drivers can rest (cf. [164, 31]). Moreover, hubs can be used for switching transportation means or simply for storing the consignment to be picked up by other drivers (see [165]). Since certain road sections might be more costly to use (e.g., if additional tolls need to be paid) or certain possibilities to extend the road network might exist, the consideration of edge selection is relevant as well. The placement of refueling stations for alternative-fuel vehicles is another important area where the NDPR arises as a subproblem. The distance constraints considered in the NDPR are well motivated by the typical range restrictions of such vehicles, see e.g., Schneider et al. [156] and the survey by Pelletier et al. [136].

Additionally, we want to point out the relation to the so-called minimum cost path problem for plug-in hybrid electric vehicles (PHEVs) considered by Arslan et al. [1]. In this problem a PHEV needs to travel from an origin to a destination node using gasoline refueling, and electric charging stations while minimizing refueling, charging and traveling costs. The authors solve the problem with a mixed integer quadratically constrained formulation. The considered problem is very similar to the NDPR. The charging stations can be viewed as some kind of relays and the vehicle corresponds to a single commodity. The primary difference lies in the fact that refueling and charging stations need to be modeled as two different types of relays which introduces additional complexity. Nevertheless, our solution techniques introduced for the NDPR (in particular, the modeling of path segments between two consecutive charging stations and the generation of the communication graph) might also be relevant for solving the PHEV problem. The placement of refueling stations has also been considered in Capar et al. [32] and Yıldız et al. [176] where the goal is to select locations for the refueling stations such that the total volume of the refueled demand is maximized for a given set of origin-destination pairs. Yıldız et al. [176] solve the problem with a B&P approach that uses an exponential number of path variables to model route feasibility.

Relation to Regenerator Location/Placement Problems

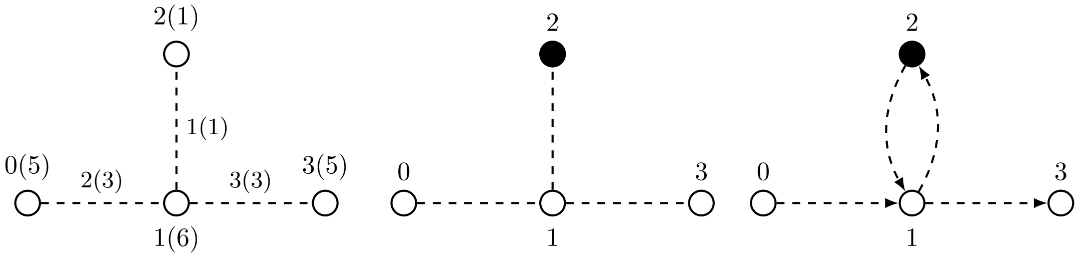
The regenerator location problem (RLP) introduced by Chen et al. [35] is closely related to the NDPR but focuses on the placement of regenerators (relays) and no additional edges can be purchased/installed in the network. More precisely, the RLP is a special case of the NDPR for $E^* = \emptyset$ and $\mathcal{K} = \{(u,v) \mid u,v \in V\, u < v\}$, i.e., it is assumed that all edges have zero cost and all node pairs need to communicate. As the RLP is equivalent to the maximum leaf spanning tree problem (MLSTP) and the minimum connected dominating set problem (MCDSP), see, e.g., [71, 117], the NDPR generalizes these problems as well. Chen et al. [35] provide several MILP formulations for the RLP which unfortunately cannot be directly used to solve the NDPR since they are not capable of selecting augmenting edges. We will, however, use the concept of a communication graph provided in Chen et al. [35] for solving the NDPR, cf. Section 3.3.1 for a detailed description. Further exact approaches for the RLP have been developed by Rahman et al. [143] who propose compact formulations and branch-and-cut (B&C) algorithms. In a recent contribution by Yıldız and Karaşan [174], survivability requirements are added to the RLP.

Chen et al. [36] introduced the so-called generalized regenerator location problem (GRLP) which extends the RLP by considering node sets $S \subseteq V$ of potential relay locations and $T \subseteq V$ of terminal nodes which need to be able to communicate with each other, i.e., $\mathcal{K} = \{(i,j) \mid i,j \in T, i < j\}$. Again, the proposed models for the GRLP cannot be applied to the NDPR. On the other hand, the NDPR is able to model the GRLP by assigning infinite costs to the nodes in $V \setminus S$ or by adding constraints that prohibit that the nodes in $V \setminus S$ are chosen as relays. The latter can be easily done in all formulations that will be introduced in the following.

Another recent contribution by Yıldız and Karaşan [175] considers several practical extensions like routing, bandwidth allocation, and modulation selection. The proposed flow model uses an exponential number of variables and is solved by a B&P approach. Finally, in a broader sense, the NDPR is also related to optimization problems dealing with wavelength division multiplexing (WDM), but, in contrast to the NDPR, the routing of an optical signal has to be optimized at two layers, the logical and the physical one, see, e.g., [142, 160] for further details.

3.2 Solution Properties

In this section we introduce and prove certain structural properties of optimal NDPR solutions that will be used to derive MILP formulations in the next section. Recall that a solution consists of a subset $\hat{E} \subseteq E^*$ of augmenting edges (together with the free edges E^0), and a subset $\hat{V} \subseteq V$ of nodes where relays have to be installed.



(a) Example graph (b) Undirected solution (c) Directed walk

Figure 3.2: Example instance with a cyclic solution for $d_{\max} = 4$ and $\mathcal{K} = \{(0,3)\}$. Nodes are labeled by node index and relay installation costs in parentheses. Augmenting edges are labeled by their lengths and installations costs in parentheses. Dashed lines indicate augmenting edges. Black nodes mark the selected relays in the solution.

As mentioned above, commodity pairs $(u, v) \in \mathcal{K}$ are assumed to be ordered, so that routing a signal from u to v is determined by a feasible (directed) u, v-walk embedded in the subgraph induced by $\hat{E} \cup E^0$, and using a subset of relays from the set $\hat{V}$. Hence, even though the solution corresponds to an undirected graph (along with the placement of relays), routing decisions are given by directed walks that are incorporated in the solution graph. In our terminology, a feasible walk corresponds to a directed subgraph embedded in the undirected solution graph, so that each edge can be traversed in both directions. The number of visits of a node in a feasible walk is equal to the in-degree of this node in the associated directed graph.

To better illustrate this interplay between the undirected solution graph and the directed subgraph associated with the routing decisions, let us consider the instance given in Figure 3.2a and assume $d_{\max} = 4$ and $\mathcal{K} = \{(0,3)\}$. The unique optimal solution which is visualized in Figure 3.2b selects all edges and places a relay at node 2. The routing of a signal from node 0 to node 3 is given by a directed walk $(0,1,2,1,3)$ embedded in this solution. This walk traverses the edge $\{1,2\}$ twice and visits node 1 twice forming a cycle, cf. Figure 3.2c. Notice that the definition of the NDPR does not forbid such cycles. As a matter of fact, enforcing that each commodity is connected by a (simple) path might significantly increase the cost of a solution. In the considered example a relay would need to be placed at node 1 instead of at 2 and edge $\{1,2\}$ would not be traversed. Consequently, the solution cost would increase from 8 to 12.

In the following, we focus on structural properties dealing with routing decisions in an optimal NDPR solution. We first prove two properties that are concerned with the maximum number of node visits for connecting a single commodity before we show that analogous results also hold when simultaneously considering all commodities with a common source.

Property 3.1. *In every optimal solution there exists for every pair $(u, v) \in \mathcal{K}$ a feasible walk from u to v visiting each relay at most once.*

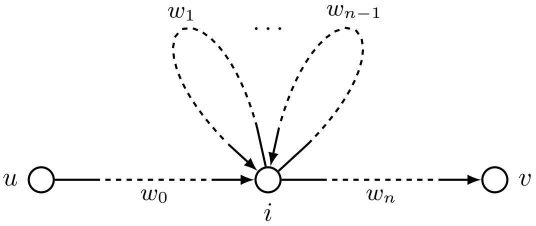


Figure 3.3: Path from u to v visiting node i n times.

Proof. Let $W = (u, w_0, i, w_1, i, \dots, w_{n-1}, i, w_n, v)$ be a feasible u, v-walk in an optimal solution that consists of subwalks $\{w_0, w_1, \dots, w_n\}$ with $i \in V$, $i \neq u, v$, cf. Figure 3.3. If i is a relay node, then walk $W' = (u, w_0, i, w_n, v)$ visiting node i only once clearly is a feasible u, v-walk as well. By repeating this argument, we will end up with a feasible u, v-walk in which each relay node appears at most once. ☐

Property 3.2. *In every optimal solution there exists for every pair $(u, v) \in \mathcal{K}$ a feasible walk from u to v visiting each non-relay node at most twice.*

Proof. Let $i \in V$ be the non-relay node closest to u which is visited $n > 2$ times in a feasible u, v-walk. Let the feasible u, v-walk be represented as $W = (u, w_0, i, w_1, i, \dots, w_{n-1}, i, w_n, v)$ with subwalks $\{w_0, w_1, \dots, w_n\}$ that do not visit i, cf. Figure 3.3. Clearly, each subwalk w_ℓ $(1 \leq \ell \leq n-1)$ that contains no relays can be deleted from W without violating feasibility. Hence, let us assume that at least one relay is traversed in w_ℓ, for all $1 \leq \ell < n$, and let $r \in V$ be the relay node with minimum distance from/to i in any of the subwalks $w_1, \dots w_n$ (considered undirected). Let $(i, v_0, v_1, \dots, v_l = r)$, $v_k \neq i$, $0 \leq k \leq l$, be the corresponding path from i to r with minimum length. Then, $W' = (u, w_0, i, v_0, v_1, \dots, v_l = r, v_{l-1}, \dots, v_0, i, w_n, v)$ is a feasible u, v-walk visiting i exactly two times. By repeating this procedure for each non-relay node which is visited more than twice, we can construct a feasible walk with the desired property. ☐

Definition 3.1. *A feasible walk w connecting a commodity pair $(u, v) \in \mathcal{K}$ is called* non-redundant *if it does not contain a feasible u, v-subwalk w', $w' \neq w$.*

Remark 3.1. *Every non-redundant walk satisfies Properties 3.1 and 3.2. If a non-relay node is visited twice in a non-redundant walk, then the second visit occurs in terms of a cycle that visits a relay. Each feasible walk can be converted into a non-redundant one using the reduction techniques from the proofs of Properties 3.1 and 3.2.*

Theorem 3.1. *Given a source $u \in S$, in every optimal solution one can embed a digraph rooted at u containing a feasible walk from u to every target $v \in T^u$. In this digraph, each relay has in-degree at most one and each non-relay node has in-degree at most two.*

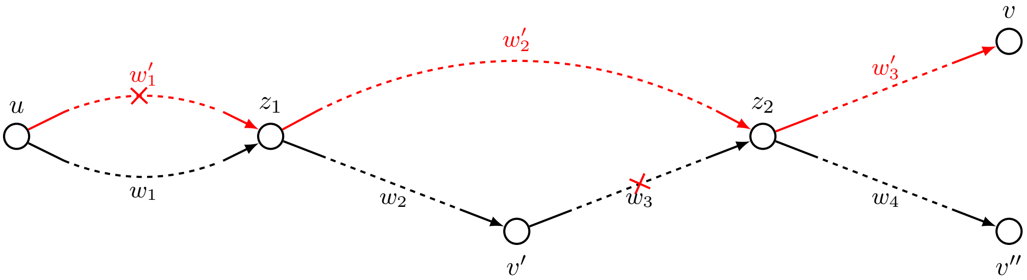


Figure 3.4: A directed graph rooted at u that needs to reach targets $T^u = \{v, v', v''\}$. We illustrate the pruning procedure after adding the red walk to the current digraph $\tilde{G}$ shown in black. The pruning procedure applied to nodes z_2 and z_1 removes subwalks w_3 and w_1', respectively.

Proof. We show the existence of such a digraph $\tilde{G}$ by construction. According to Properties 3.1 and 3.2, for each target $v \in T^u$ there exists a feasible walk from u to v with the desired properties. We choose one of the targets $v \in T^u$ and initialize the graph $\tilde{G}$ with arcs determining its feasible walk (satisfying Properties 3.1 and 3.2). We then continue to iteratively insert non-redundant walks for the remaining targets. Whenever a new walk (associated with a new target) is considered, there are three possibilities: (a) the graph already contains the walk's target, (b) the graph and the walk are node-disjoint (except for the source u), or (c) the graph and the walk share more than one node.

If (a) occurs, we do not modify $\tilde{G}$ and continue with the next target node. Case (b) allows to add the walk to $\tilde{G}$ without violating the desired properties since the in-degrees of nodes already present in $\tilde{G}$ do not change. The third case (c) is more difficult to handle since simply adding the walk might increase the in-degree for some nodes and thus can destroy the properties we need.

Initially we add the walk to $\tilde{G}$. Let $Z = \{z_1, \ldots, z_n\}$ be the set of nodes on this walk with in-degree greater than one. Each of these nodes might be in conflict with the required properties. To resolve this issue we apply a pruning procedure to each node $z \in Z$.

First, we identify a non-redundant walk in $\tilde{G}$ that reaches z at minimum distance (using as few edges as possible) from the preceding relay or node u. This walk might visit z at most twice using a cycle to a relay to reduce the covered distance. Then, we delete all incoming arcs of z not contained in this walk. Moreover, we iteratively remove preceding arcs of the already removed ones (not contained in the walk) until we arrive at source node u, a node that is a target, or a node that has an outgoing walk to a target. For an example see Figure 3.4.

After processing all u, v-walks in this way we can reach all targets of u in $\tilde{G}$. Moreover, walks are added such that all nodes in $\tilde{G}$ are guaranteed to meet the required degree restrictions. □

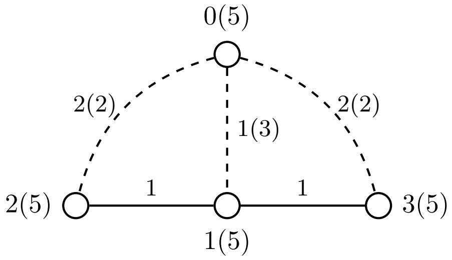


Figure 3.5: Shortest path connections may be suboptimal due to edge reuse. Paths $(0,1,2)$ and $(0,1,3)$ are dominated by $(0,2)$ and $(0,3)$, respectively. Nevertheless, the optimal solution is $\{\{0,1\},\{1,2\},\{1,3\}\}$ ($\mathcal{K}=\{(0,2),(0,3)\}$ and $d_{\max}=3$). Nodes are labeled by node index and relay installation costs in parentheses. Edges are labeled by their lengths and installation costs in parentheses (for augmenting edges). Solid lines indicate free edges and dashed lines augmenting edges.

3.3 Mixed Integer Linear Programming Formulations

The MILP formulations that will be introduced in the following are based on a *communication graph* whose construction from the original graph will be explained next. Afterwards, we will provide two MILP formulations and discuss their properties.

3.3.1 Communication Graph

Communication graph $G_{\mathrm{C}}=(V,C)$ is defined on the original node set V and its edge set C consists of all node pairs $\{i,j\}\subseteq V,\ i\neq j$, for which at least one path P with length $d(P)\leq d_{\max}$ exists in G, i.e., i and j can communicate using edges from E^* or E^0 without installing relays. Recall that (after preprocessing) every feasible path without relays connecting commodity $(u,v)\in\mathcal{K}$ in G contains at least one edge $e\in E^*$ with positive costs. Since multiple paths can be used for connecting a node pair, it is not clear which one of the potentially exponentially many paths will be used in an optimal solution and thus all of them need to be considered. Our definition of the communication graph extends the one introduced in [35] and [36] for solving the (G)RLP. In contrast to their definition, our communication graph considers all edges $e\in E$ and not only those with zero costs.

Figure 3.5 demonstrates how the existence of common subpaths for several commodities produces a better solution than the one obtained by combining cheapest subpaths of the individual commodities. Assume $\mathcal{K}=\{(0,2),(0,3)\}$ and observe that the cheapest $0,2$-path is the edge $\{0,2\}$ and the cheapest $0,3$-path is edge $\{0,3\}$, both with a cost of 2 yielding a solution with total cost 4. The union of paths $(0,1,2)$ and $(0,1,3)$ results, however, in a cheaper solution with total cost 3.

In the following for each pair of distinct nodes $b=\{i,j\}$, we define $P_b=\{p\mid p \text{ is an } i,j\text{-path in } G,\ \text{such that } d(p)\leq d_{\max}\}$ as the set of all feasible i,j-paths in

G. Furthermore, let $P_b^0 = \{p \mid p \text{ is an } i,j\text{-path in } G^0, \text{ such that } d(p) \leq d_{\max}\}$ be the set of all such paths in $G^0 = (V, E^0)$, i.e., using only free edges. Then, sets $C^0 = \{b = \{i,j\} \mid P_b^0 \neq \emptyset\}$ and $C^* = \{b = \{i,j\} \mid P_b^0 = \emptyset \text{ and } P_b \neq \emptyset\}$ define the node pairs that can be connected using only free edges and using at least one augmenting edge, respectively. Thus, the edge set C of communication graph $G_{\mathrm{C}} = (V, C)$ corresponding to $G = (V, E)$ is defined as $C = C^* \cup C^0$, i.e., the set of node pairs that can be connected using edges in $E^* \cup E^0$.

Figure 3.6 illustrates the stepwise generation of a communication graph for the input graph provided in Figure 3.6a, $d_{\max} = 4$. After initializing C with the set of free edges E^0, all remaining edges $C^0 \setminus E^0$ corresponding to connections that can be established by using only free edges and no relays are added, cf. the edges $\{1,4\}$ and $\{2,3\}$ corresponding to paths $(1,2,4)$ and $(2,1,3)$, respectively, in Figure 3.6b. Next, all connections possible through the use of augmenting edges (i.e., either augmenting edges or paths containing at least one augmenting edge) are considered, see the dashed edges in Figure 3.6c. The corresponding node pairs are connected by dotted lines in Figure 3.6c. Communication graph G_{C} is finally obtained by removing potentially existing multi-edges, cf. Figure 3.6d where connections in C^0 and C^* are displayed by solid and dotted lines, respectively.

The following property of the communication graphs is crucial for developing the MILP models shown below.

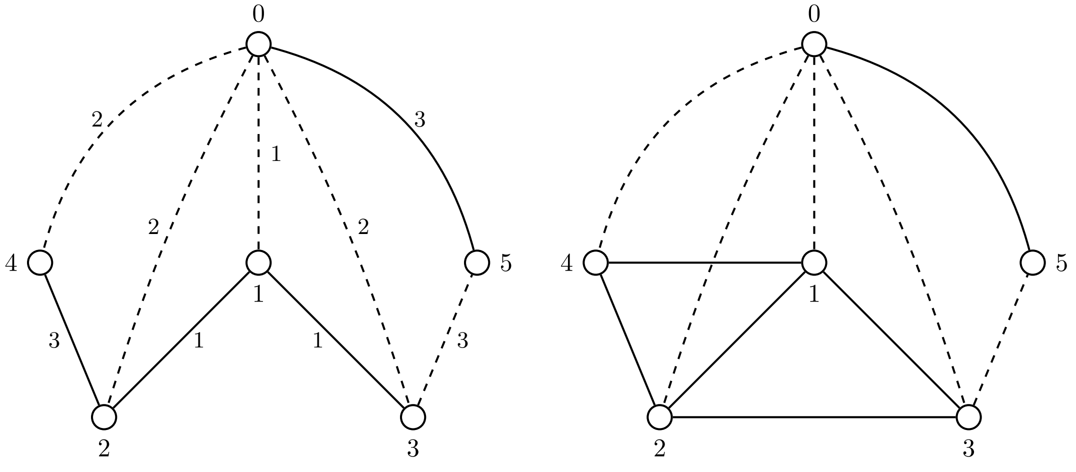
Property 3.3. *For every commodity pair $(u, v) \in \mathcal{K}$, a non-redundant u, v-walk in the original graph can be mapped to a simple path in the communication graph with relays placed at all intermediate nodes (if any).*

Proof. According to Remark 3.1 there exists a non-redundant (feasible) walk in the original graph visiting every relay at most once. Observe that we can partition the walk in the original graph into maximal feasible subpaths such that none of their intermediate nodes are relays. Communication graph G_{C} contains an edge for every feasible path that is not required to visit any relays. Therefore, the mentioned subpaths can be translated to edges of the communication graph leading to the desired simple path. Conversely, each simple path in the communication graph can be translated to a non-redundant walk in the original graph. $\square$

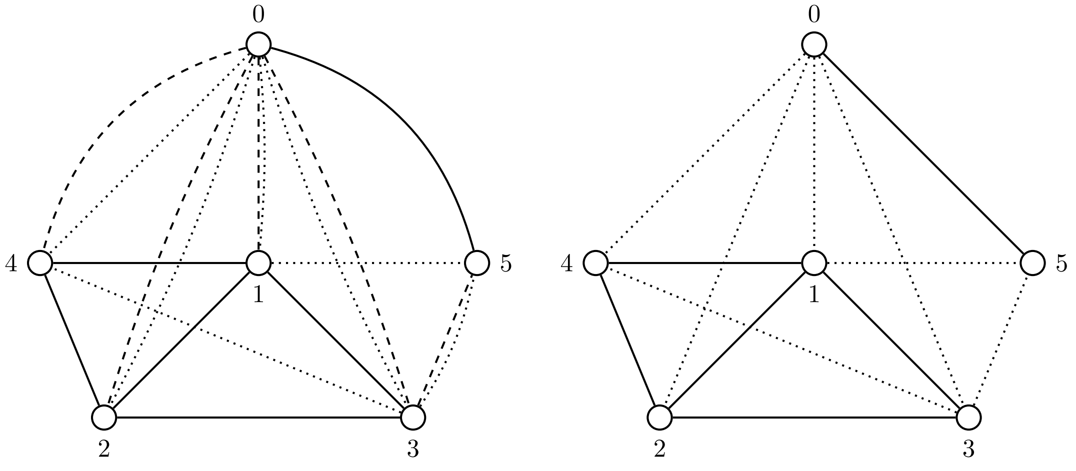
Corollary 3.1. *For a feasible solution we can identify for each source $u \in S$ and all its targets a tree in the communication graph with relays placed at all intermediate nodes where each leaf is a target of u such that each (unique) path to a target corresponds to a feasible walk in the original graph.*

Proof. We use the graph obtained according to the proof of Theorem 3.1. Then, we partition and translate it in the same way as in the proof of Property 3.3. $\square$

The MILP formulations introduced in the following subsections make use of flows or cut-sets to model feasible paths in the communication graph. In addition, the relation between edges in G_C and (an exponential number of) paths in G is established.



(a) Original graph (b) Adding all connections in C^0



(c) Adding all connections in C^* (d) Final communication graph

Figure 3.6: Generation of the communication graph for $d_{\max} = 4$. Nodes are labeled by node index. Edges are labeled by their lengths. Solid lines indicate free edges, dashed lines augmenting edges, and dotted lines indicate connections from C^*.

3.3.2 Multi-Commodity Flow Formulation

We first present a multi-commodity flow formulation on the communication graph (MCF) which uses one set of flow variables for each commodity in $\mathcal{K}$. It utilizes the following design variables defined on G:

$$x_e = \begin{cases} 1, & \text{if } e \text{ is installed in the network} \\ 0, & \text{otherwise} \end{cases} \qquad \forall e \in E^*,$$

$$y_i = \begin{cases} 1, & \text{if a relay is installed at } i \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \in V.$$

Using $A(C) = \{(i,j) \mid \{i,j\} \in C\}$ we define flow variables f_{ij}^{uv} for all commodity pairs $(u,v) \in \mathcal{K}$ and each direction of edge $\{i,j\} \in C$, i.e.,

$$f_{ij}^{uv} = \begin{cases} 1, & \text{if the } u,v\text{-path in } G_{\mathrm{C}} \text{ traverses edge } \{i,j\} \text{ in direction from } i \text{ to } j \\ 0, & \text{otherwise.} \end{cases}$$

Finally, we use variables λ_b^p that correspond to the paths $p \in P_b$ that have been identified as possible realizations for the connections $b \in C^*$. More precisely:

$$\lambda_b^p = \begin{cases} 1, & \text{if connection } b \text{ is realized by path } p \in P_b \\ 0, & \text{otherwise} \end{cases} \qquad \forall b \in C^*.$$

Since no augmenting edges need to be purchased when sending flow through edges from C^0, we do not need to consider path variables for $b \in C^0$. The MILP formulation reads as follows:

$$\text{(MCF)} \quad \min \sum_{i \in V} c_i y_i + \sum_{e \in E^*} w_e x_e \tag{3.1}$$

$$\sum_{(i,j) \in A(C)} f_{ij}^{uv} - \sum_{(j,i) \in A(C)} f_{ji}^{uv} = 1 \qquad \forall (u,v) \in \mathcal{K}, \forall i \in V, i = u, \tag{3.2}$$

$$\sum_{(i,j) \in A(C)} f_{ij}^{uv} - \sum_{(j,i) \in A(C)} f_{ji}^{uv} = -1 \qquad \forall (u,v) \in \mathcal{K}, \forall i \in V, i = v, \tag{3.3}$$

$$\sum_{(i,j) \in A(C)} f_{ij}^{uv} - \sum_{(j,i) \in A(C)} f_{ji}^{uv} = 0 \qquad \forall (u,v) \in \mathcal{K}, \forall i \in V, i \neq u, i \neq v, \tag{3.4}$$

$$-y_i + \sum_{(i,j) \in A(C)} f_{ij}^{uv} \leq 0 \qquad \forall (u,v) \in \mathcal{K}, \forall i \in V \setminus \{u,v\}, \tag{3.5}$$

$$-(f_{ij}^{uv} + f_{ji}^{uv}) + \sum_{p \in P_b} \lambda_b^p \geq 0 \qquad \forall (u,v) \in \mathcal{K}, \forall b = \{i,j\} \in C^*, \quad (\mu_b^{uv}) \tag{3.6}$$

$$x_e - \sum_{p \in P_b : e \in p} \lambda_b^p \geq 0 \qquad \forall e \in E^*, \forall b \in C^*, \quad (\alpha_b^e) \tag{3.7}$$

$$\lambda_b^p \geq 0 \qquad \forall b \in C^*, p \in P_b, \tag{3.8}$$

$$0 \leq f_{ij}^{uv} \leq 1 \qquad \forall (u,v) \in \mathcal{K}, \forall (i,j) \in A_{\mathrm{C}}, \tag{3.9}$$

$$\mathbf{y} \in \{0,1\}^{|V|}, \mathbf{x} \in \{0,1\}^{|E^*|}. \tag{3.10}$$

Constraints (3.2)–(3.4) ensure that for each commodity $(u,v) \in \mathcal{K}$ one unit of flow is sent from u to v. Every node with outgoing flow that is not the source of the corresponding flow must be a relay node, cf. Property 3.3. This relation is enforced by inequalities (3.5). Constraints (3.6) ensure that flow along a connection $b \in C^*$ is only permitted if at least one of the available path realizations has been selected. Due to Property 3.3, the solution for each commodity pair $(u,v) \in \mathcal{K}$ will be a path in G_{C}. Hence, only one arc (i,j) or (j,i) per edge $b = \{i,j\}$ will be selected in each variable set. The last set of inequalities guarantees that for all selected path realizations, the corresponding augmenting edges will be part of the solution. Note that each variable set only considers a single pair $(u, v) \in \mathcal{K}$. Arcs targeting u or leaving v are irrelevant with respect to the flow variables, hence the respective flow variables can be omitted from the formulation.

Pricing subproblem. Since the number of path variables in formulation (MCF) may be exponentially large, we will use column generation for solving its linear programming (LP) relaxation. To formulate the pricing subproblem, let $\mu_b^{uv} \geq 0$ and $\alpha_b^e \geq 0$ be the dual variables associated to constraints (3.6) and (3.7), respectively. Then, for each $b \in C^*$, the pricing subproblem (i.e., find a path $p \in P_b$ with negative reduced costs) is defined as

$$\arg\min_{p \in P_b} \left(\sum_{e \in E^* \cap p} \alpha_b^e - \sum_{(u,v) \in \mathcal{K}} \mu_b^{uv} \right).$$

Since the second summation is a constant for a fixed $b \in C^*$, it further reduces to finding

$$\arg\min_{p \in P_b} \sum_{e \in E^* \cap p} \alpha_b^e \qquad \forall b \in C^*.$$

which is a weight constrained shortest path problem (WCSPP) defined on the graph $G = (V, E)$ with edge weights $\omega\colon E \to \mathbb{R}_{\geq 0}$, defined as $\omega_e = d_e$, for all $e \in E$ and edge costs $\gamma\colon E \to \mathbb{R}_{\geq 0}$ defined as

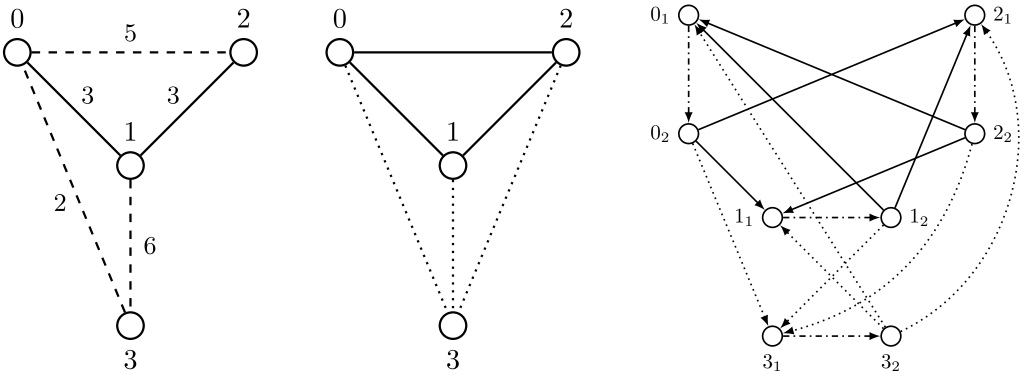
$$\gamma_e = \begin{cases} 0, & e \in E^0 \\ \alpha_b^e, & e \in E^* \end{cases} \qquad \forall e \in E.$$

Further details regarding pricing will be provided in Section 3.4.2.

3.3.3 Cut Formulation

We propose an alternative MILP formulation that is based on a single, *directed communication graph* in which relays are identified by splitting each node i into two copies i_1, i_2. Besides *relay arcs* (i_1, i_2) for each node $i \in V$, we add arcs (j_2, i_1) and (i_2, j_1) for each edge $\{i, j\} \in C$, i.e., all incoming arcs target i_1 and all outgoing arcs emanate from i_2. We obtain graph $G'_{\mathrm{C}} = (V'_{\mathrm{C}}, A'_{\mathrm{C}})$ with node set $V'_{\mathrm{C}} = \{i_1, i_2 \mid i \in V\}$ and arc set $A'_{\mathrm{C}} = \{(i_2, j_1), (j_2, i_1) \mid \{i, j\} \in C\} \cup A^{\mathrm{r}}_{\mathrm{C}}$ where arcs in $A^{\mathrm{r}}_{\mathrm{C}} = \{(i_1, i_2) \mid i \in V\}$ are used to identify relays. An example of a directed communication graph is shown in Figure 3.7 where solid arcs represent free connections, dotted arcs connections with augmenting edges, and dash-dotted arcs correspond to relays.

The formulation introduced next represents each feasible u, v-walk of a given commodity pair $(u, v) \in \mathcal{K}$ as a directed path from u to v in G'_{C} with relays placed at all intermediate nodes (cf. Property 3.3). To this end we utilize binary variables $\mathbf{x}$ for the augmenting edges and path variables $\boldsymbol{\lambda}$, introduced above. In addition, we associate binary variables X_{ij} to the arcs $(i, j) \in A'_{\mathrm{C}}$ of the directed communication graph. Due to the one-to-one



(a) Original graph $G = (V, E^0 \cup E^*)$

(b) Communication graph $G_C = (V, C)$

(c) Directed communication graph $G'_C = (V'_C, A'_C)$

Figure 3.7: Generation of the directed communication graph $G'_C = (V'_C, A'_C)$ for $d_{\max} = 7$. Nodes are labeled by node index. Edges are labeled by their lengths. Solid lines indicate free edges, dashed lines augmenting edges, and dotted lines indicate connections from C^*. Dash-dotted arcs correspond to relays.

correspondence between the arcs in A^{r}_C and the relays we can use the arc variables directly to identify the relays. The model then reads as follows:

$$\text{(CUT)} \quad \min \sum_{i \in V} c_i X_{(i_1,i_2)} + \sum_{e \in E^*} w_e x_e \tag{3.11}$$

$$\sum_{a \in \delta^-(W)} X_a \geq 1 \qquad \forall (u,v) \in \mathcal{K}, \forall W \subset V'_C, \; v_1 \in W, u_2 \notin W, \tag{3.12}$$

$$-X_{i_2 j_1} + \sum_{p \in P_b} \lambda_b^p \geq 0 \qquad \forall b = \{i,j\} \in C^*, \quad (\mu_b^1) \tag{3.13}$$

$$-X_{j_2 i_1} + \sum_{p \in P_b} \lambda_b^p \geq 0 \qquad \forall b = \{i,j\} \in C^*, \quad (\mu_b^2) \tag{3.14}$$

$$x_e - \sum_{p \in P_b : e \in p} \lambda_b^p \geq 0 \qquad \forall e \in E^*, \forall b \in C^*, \quad (\alpha_b^e) \tag{3.15}$$

$$\lambda_b^p \geq 0 \qquad \forall b \in C^*, p \in P_b, \tag{3.16}$$

$$\mathbf{X} \in \{0,1\}^{|A'_C|}, \mathbf{x} \in \{0,1\}^{|E^*|}. \tag{3.17}$$

We will refer to this model as the *cut formulation on a directed communication graph* (CUT). Cut-set inequalities (3.12) ensure the existence of a directed path in G'_C from u_2 (the source copy of u) to v_1 (the target copy of v) for each commodity pair $(u,v) \in \mathcal{K}$. By construction, every second arc along this path corresponds to a relay node. Costs for these arcs are included in the objective function. The remaining arcs of type (i_2, j_1) or (j_2, i_1) correspond to paths between i and j, where $b = \{i,j\} \in C^*$. Constraints (3.13) and (3.14) ensure that whenever an arc (i_2, j_1) or (j_2, i_1) is used, then at least one corresponding path from $P_{\{i,j\}}$ is selected. Due to the presence of multiple sources, both arcs (i_2, j_1) or (j_2, i_1) can be used in a feasible solution. Finally, constraints (3.15) ensure that all augmenting edges of the selected paths are included in the solution. Since the number of these constraints is in general exponential, we will separate them dynamically only when violated, see Section 3.4.3.

Notice that a feasible solution may contain arcs (j_2, i_1) and (i_2, h_1) (for some distinct nodes $i, j, h \in V$), but not necessarily the arc (i_1, i_2). This happens, for example, when node i is both a target and a source, but it is not contained in any connection passing through it (and thus there is no need to install a relay at i). As a consequence, the arcs that correspond to node splitting identify the nodes from V where relays have to be placed, whereas the arcs linking the node copies of i and j map to the paths $p \in P_b$, $b = \{i, j\}$ as in the (MCF) model.

Pricing subproblem. To formally state the pricing subproblem for variables λ_b^p we associate dual variables $\mu_b^1 \geq 0$ and $\mu_b^2 \geq 0$ to constraints (3.13) and (3.14), respectively, and dual variables $\alpha_b^e \geq 0$ to constraints (3.15). Similar to the previous two cases, for each $b \in C^*$ we need to identify a feasible path with minimum reduced costs

$$\min_{p \in P_b} \left(\sum_{e \in E^* \cap p} \alpha_b^e - \mu_b^1 - \mu_b^2 \right).$$

Thus, we can solve the pricing subproblem by solving for each $b \in C^*$ a WCSPPs with edge weights set to α_b^e for $e \in E^*$, and to zero otherwise, see Section 3.4.2 for further details.

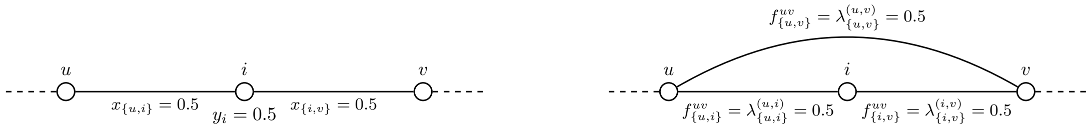
Since formulation (CUT) contains an exponential number of variables ($\boldsymbol{\lambda}$) *and* an exponential number of cut-set constraints (3.12), a column-and-row generation approach is employed to solve its LP relaxation (and a BP&C algorithm to find an optimal solution), see, e.g., [11, 52]. Fortunately, the λ variables are not involved in the connectivity constraints (3.12). Thus, we can separate these parts so that column generation can be done independently of cut generation, i.e., the added cuts do not influence the structure of the pricing subproblem.

3.3.4 Valid Inequalities

We now describe several types of valid inequalities that are redundant for the set of feasible solutions but can strengthen the models' LP relaxations.

Connectivity Cuts in the Original Graph

The example given in Figure 3.8 shows that connectivity constraints (3.18) can be violated in LP solutions of (MCF) and (CUT), respectively. These constraints ensure that the value of each (undirected) cut separating the source and target of a commodity is at least


u
i
v
$x_{\{u,i\}} = 0.5$
$y_i = 0.5$
$x_{\{i,v\}} = 0.5$
$f^{uv}_{\{u,v\}} = \lambda^{(u,v)}_{\{u,v\}} = 0.5$
u
i
v
$f^{uv}_{\{u,i\}} = \lambda^{(u,i)}_{\{u,i\}} = 0.5$
$f^{uv}_{\{i,v\}} = \lambda^{(i,v)}_{\{i,v\}} = 0.5$

(a) Original graph

(b) Communication graph

Figure 3.8: A partial solution to the LP relaxation of (MCF) that violates connectivity constraints (3.18) for commodity $(u, v) \in \mathcal{K}$. The total flow in the communication graph from source u to its target v equals 1. However, the value of the minimum u-v cut in the original graph is only 0.5.

one. Since cut inequalities including free edges are trivially satisfied, we only consider subsets W inducing cuts without free edges in (3.18):

$$\sum_{e\in\delta(W)} x_e \geq 1 \qquad \forall W \subset V : \delta(W) \cap E^0 = \emptyset, \exists (u,v) \in \mathcal{K} : u \notin W, v \in W. \qquad (3.18)$$

One can further strengthen the quality of the LP relaxation by replacing undirected cut-set inequalities (3.18) by their directed counterparts. To this end, we introduce for each root $u \in S$ variables $z^u_{ij} \geq 0$, $\forall \{(i,j) \mid \{i,j\} \in E^*\}$. Variable z^u_{ij} is set to one if one can embed in the original graph a directed path from u to some $v \in T^u$ using arc (i,j). Then, constraints (3.18) can be enhanced by:

$$\sum_{a\in\delta^-(W)} z^u_a \geq 1 \qquad \forall W \subset V : \delta(W) \cap E^0 = \emptyset, \exists (u,v) \in \mathcal{K} : u \notin W, v \in W, \qquad (3.19)$$

$$z^u_{ij} + z^u_{ji} \leq x_{\{i,j\}} \qquad \forall u \in S, \{i,j\} \in E^*, \qquad (3.20)$$

$$z^u_{ij}, z^u_{ji} \geq 0 \qquad \forall u \in S, \{i,j\} \in E^*. \qquad (3.21)$$

It can easily be seen that the directed connectivity constraints (3.19) are at least as strong as the undirected ones introduced above. From Figure 3.9 we conclude that they can be strictly stronger if there exists at least one commodity source with more than one target (in the presence of linking constraints (3.20)). Note that similar to the undirected variant, we do not add cut-set inequalities for node subsets with incident free arcs.

Since both classes of connectivity constraints are of exponential size, we will dynamically separate them, see Section 3.4.3 for details.

Relay Constraints

For the (CUT) model we additionally exploit the fact that a relay has to be placed at some node iff it is an intermediate node along a path within the communication graph. This results in the following constraints that are added to the (CUT) model:

$$\sum_{a\in\delta^-(i_1)} X_a \leq \min(|S|, |\delta(i)| - 1) \cdot X_{(i_1,i_2)} \qquad \forall i \notin S \cup T, \qquad (3.22)$$

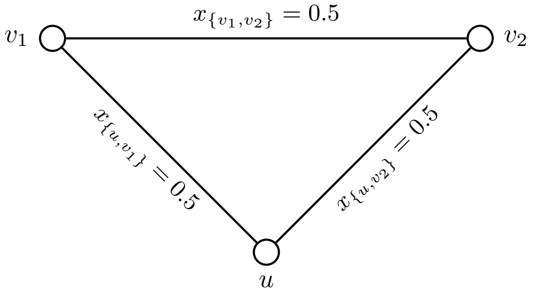


Figure 3.9: A solution to the LP relaxation for $\mathcal{K} = \{(u, v_1), (u, v_2)\}$ and $w_{\{u,v_1\}} = w_{\{u,v_2\}} = 1$, $w_{\{v_1,v_2\}} = 3$. The solution is optimal with respect to the undirected cut-set inequalities but not with respect to the directed ones which will cut off this LP solution.

$$\sum_{a \in \delta^+(i_2)} X_a \leq \min(|T|, |\delta(i)| - 1) \cdot X_{(i_1,i_2)} \qquad \forall i \notin S. \tag{3.23}$$

The first set of constraints makes sure that a relay is installed at each node i which does not belong to any commodity pair, whenever there is an arc entering i_1. Similarly, whenever there is an arc leaving i_2, and node i is not a source, there has to be a relay installed at i.

These constraints benefit from the fact that the (CUT) model uses only one set of variables to ensure connectivity in the communication graph. They are particularly effective if the big-M constants are small, e.g., if only a single source node exists. These constraints turned out to be beneficial not only for strengthening the LP relaxation of the (CUT) model, but also for improving the convergence concerning the dynamically separated inequalities.

3.4 Algorithmic Framework

We developed BP&C algorithms for the MILP formulations described in the previous section, see Table 3.1 for a summary. In the following, after providing some remarks on preprocessing that aim to reduce the size of the problem instances, we present additional details (including separation and pricing procedures) of these algorithms.

3.4.1 Preprocessing

Recall that at the beginning, we remove all edges $e \in E$ such that $d_e > d_{\max}$. If graph G separates into several connected components, and there exists a pair $(u, v) \in \mathcal{K}$ such that u and v belong to different components, then the instance is clearly infeasible. Note that if the problem admits a feasible solution and it contains more than one connected component, every component describes a separate problem instance that can be solved independently. Next, we identify and remove all pairs in $\mathcal{K}$ that can be connected using

Model	Type	B	Graph
(MCF)	BPC	$\sum_{(u,v)\in\mathcal{K}} \mu_b^{uv}$	Communication graph G_{C}
(CUT)	BPC	$\mu_b^1 + \mu_b^2$	Directed communication graph G'_{C}

Table 3.1: Algorithm overview. Name (Model), considered decomposition algorithm (Type), model specific pricing subproblem threshold for WCSPP (B), and considered communication graph (Graph).

solely free edges from E^0 and no relays (this can be easily done by applying shortest path algorithms on $G^0 = (V, E^0)$).

For the separation procedures explained in Section 3.4.3, few commodities with many targets per source are preferable. To this end, we use the fact that commodity pairs can be reordered because in an undirected graph the existence of a feasible u, v-walk implies the existence of a feasible v, u-walk. We heuristically reorder the commodities as follows. First, we compute for each node the number of times it appears in a commodity pair. Then, we iteratively choose the node i with the highest count (breaking ties by node index), reorder the commodity pairs involving i by setting i to be the source, and decrease the counts of all nodes by the number of times i is involved in an associated commodity. The procedure is repeated until the count of every node becomes zero.

3.4.2 Column Generation

We use column generation to deal with the exponential number of path variables in the considered models. The paths we are looking for correspond to the edges of the communication graph. Due to Property 3.3, these edges represent loop-free paths between node pairs $b = \{i, j\}$. More precisely, for each $b \in C^*$, the reduced costs, denoted by R_b, of its associated path variable are calculated as

$$R_b = \min_{p\in P(b)} \sum_{e\in E^*\cap p} \alpha_b^e - B$$

where the value of the constant B depends on the considered formulation and is given in Table 3.1.

As already mentioned, for each $b \in C^*$, this pricing subproblem is a WCSPP defined on the graph $G = (V, E)$ with non-negative edge weights $\omega_e = d_e$, for all $e \in E$ and non-negative edge costs $\gamma_e = \alpha_b^e$ if $e \in E^*$ and $\gamma_e = 0$, otherwise. The goal is to find a path in G connecting a node pair $b = \{i, j\}$ that minimizes the sum of edge costs and whose weight does not exceed $d_{\max}$.

In our implementation we add one variable corresponding to a least cost path for each $b \in C^*$ in each pricing iteration if it has negative reduced costs. Our decision to add at most $|C^*|$ variables in each iteration is based on preliminary experiments indicating that this strategy outperforms other options such as adding only a single variable in each iteration.

Initial set of columns. We initially add a set of variables ensuring that there exists a feasible solution to the LP relaxation. To this end, we add a variable corresponding to a connection with minimal length for each $b \in C^*$. Such a connection can easily be found with Dijkstra's algorithm (see [54]) using the edge lengths as costs.

Solving the pricing subproblems. The WCSPP on a graph with nonnegative edge costs is a weakly $\mathcal{NP}$-hard problem for which fast pseudo-polynomial exact algorithms are available. For the implementation in our models we use the generic resource-constrained shortest path algorithm from the Boost Graph Library (BGL) in version 1.63.0, see [18]. To speed up performance we prevent path expansions leading to costs larger than or equal to B since such paths can never result in negative reduced costs.

3.4.3 Separation

Depending on the formulation, up to three different families of exponentially-sized constraints can be considered. We separate the three classes in the following order: (1) undirected cut-set inequalities (3.18) on the original graph, (2) directed cut-set inequalities (3.19) on the original graph, and (3) cut-set inequalities (3.12) on the communication graph (for the (CUT) model). Only if no violated inequalities of previous classes can be found, we continue with the next class. Violated inequalities of all three classes are identified by maximum flow computations according to the commodity pairs using the algorithm by Cherkassy and Goldberg [37]. Thereby, the edge (or arc) capacities are set to the current LP solution values plus a small value in order to prefer sparse cuts, i.e., those that contain the fewest edges or arcs, respectively. In case of ties, we always choose a cut that is closest to the target node. To avoid adding too many cuts we only consider inequalities that are violated by a value of at least 0.5.

Connectivity in the Original Graph.

As noted in Section 3.3.4, directed cuts on the original graph are stronger than their undirected counterpart for commodity sources that need to be connected to more than one target. Therefore, if $|T^u| = 1$ for some $(u, v) \in \mathcal{K}$, we only add undirected cut-set inequalities (3.18) for this commodity pair. Otherwise, we add variables z^u and consider the directed constraints (3.19). That way, we always use the strongest variant of the connectivity inequalities while avoiding unnecessary overhead whenever possible. We note that such a separation strategy also benefits from the aforementioned reordering of the commodity pairs.

For connectivity cuts based on the original graph, we also consider so-called nested cuts (see, e.g., Ljubić et al. [115]): We set the arc capacities of just added cuts to one and repeat the flow computation to possibly find other violated inequalities. The procedure is continued until no further violations can be detected. Observe that the capacity updates influence the subsequent separation steps. To avoid an unwanted bias we consider the commodity pairs in a random order based on a fixed seed.

Connectivity in the Communication Graph.

Since connectivity constraints (3.12) on the communication graph are not redundant, their separation is not optional, i.e., they need to be applied at least to all integer solutions encountered during the B&B procedure. In our implementation, we additionally use these cuts to cut off fractional solutions, applying the maximum-flow procedures described above.

3.4.4 Initial Pool of Inequalities

We now shortly summarize the set of valid inequalities that are used to initialize our models.

Cuts in the Original Graph.

As mentioned above, both types of the original graph connectivity cuts are dynamically separated. To speed up convergence we add a subset of these inequalities a priori to the model:

$$\sum_{a \in \delta^-(v)} z_a^u = 1 \qquad \forall (u,v) \in \mathcal{K} : |T^u| > 1, \delta(v) \cap E^0 = \emptyset, \tag{3.24}$$

$$\sum_{a \in \delta^-(i)} z_a^u \leq \sum_{a \in \delta^+(i)} z_a^u \qquad \forall u \in S : |T^u| > 1, \forall i \in V \setminus (S \cup T), \delta(i) \cap E^0 = \emptyset. \tag{3.25}$$

If undirected cuts are separated for at least one commodity pair (i.e., $\exists u \in S : |T^u| = 1$), we also add the following inequalities since each commodity source and target node has at least one incident edge:

$$\sum_{e \in \delta(i)} x_e \geq 1 \qquad \forall i \in V : i \in S \cup T, \delta(i) \cap E^0 = \emptyset. \tag{3.26}$$

Similarly, we know that relays are never isolated. Thus, we add the following type of inequalities to (MCF):

$$\sum_{e \in \delta(i)} x_e \geq y_i \qquad \forall i \in V : i \notin S \cup T, \delta(i) \cap E^0 = \emptyset. \tag{3.27}$$

Equivalent constraints are considered for the (CUT) model by replacing y_i by $X_{(i_1,i_2)}$.

Cuts in the Communication Graph.

We add all constraints from Section 3.3.4 a priori to model (CUT), extended by the following inequalities that ensure that each target has at least one incoming arc and each source has at least one outgoing arc:

$$\sum_{a \in \delta^-(v_1)} X_a \geq 1 \qquad \forall v \in T,$$

$$\sum_{a \in \delta^+(u_2)} X_a \geq 1 \qquad \forall u \in S.$$

3.4.5 Heuristic

Feasible NDPR solutions and initial upper bounds for our algorithms are obtained by using heuristic (CH1) originally introduced in [30]. Its basic idea is to iteratively compute a solution by solving the problem for the individual commodities. In each iteration all previously added augmenting edges and relays are assigned zero costs. In our implementation we perform ten runs of (CH1) in which we vary the order in which the commodities are considered (fixed seed random order) and finally adopt the best solution found. Columns required to represent the respective solution are added to the initial formulation.

For each commodity (i.e., in each iteration) we need to solve the minimum cost path problem with relays (MCPPR). Our implementation uses a variant of the pseudo-polynomial dynamic programming (DP) algorithm introduced by [105]. Their algorithm for the MCPPR solves the problem on a directed graph. In the undirected variant we need to make sure that, once an edge has been traversed in one direction, using it in the other direction incurs no additional costs. The simplest way of handling this is to augment the DP states by a set of already used edges, see Algorithm 3.1 for the adjusted pseudocode. Each state is a tuple of the form $x = (\pi_x^{\mathrm{c}}, \pi_x^{\mathrm{d}}, \xi_x, v_x, E_x^*)$ where π_x^{c} denotes the cost of the current walk, π_x^{d} the distance from the last relay or the starting node along the walk, ξ_x a reference to the preceding state, v_x the final node of the walk, and E_x^* the set of already traversed edges. As suggested in [105], the list of states L is ordered according to non-decreasing cost to allow for early termination once a state containing the target node as final node is reached.

After the ten runs of (CH1) we perform a final run for which we set the costs of all relays and edges selected by the best solution to zero. The idea behind this run is to remove possible redundancies with respect to the selected relays and edges. Thereby, it is important to break ties regarding the ordering of L by prioritizing states with smaller π_x^{d}. We denote the modified algorithm by (CH1+).

3.4.6 Solver Configuration

Our algorithms are implemented in C++ using SCIP 3.2.1 (see [64]) as BP&C framework and CPLEX 12.6.3 as LP solver. The dual simplex method has been used for solving the LP relaxations as it outperformed other options (primal simplex, barrier) in preliminary experiments. All experiments have been performed in single thread mode with presolving, probing, and the solvers general purpose heuristics turned on. General purpose cutting planes have been deactivated.

Algorithm 3.1: DP algorithm for the MCPPR in an undirected graph

```
Input: graph $G = (V, E, c, w, d), E = E^0 \cup E^*$, pair $(s, t) \in \mathcal{K}$
Data: cheapest path $M_{di}$ to $i$ at distance $d$
Data: set $L$ of unexpanded states
$L \leftarrow \{(0, 0, NULL, s, \emptyset)\}$
forall $d \in \{0, \ldots, d_{\max}\}, i \in V$ do $M_{di} \leftarrow NULL$
while $L \neq \emptyset$ do
    select first $x \in L$ and remove it from $L$
    forall $\{v_x, j\} \in \delta(v_x)$ do
        $\hat{d} \leftarrow \pi^{\mathrm{d}}_x + d_{\{v_x,j\}}$                // arrival distance at $j$
        if $\hat{d} \leq d_{\max}$ then
            $\hat{c} \leftarrow \pi^{\mathrm{c}}_x$                              // cost at $j$
            if $\{v_x, j\} \notin E^0 \cup E^*_x$ then $\hat{c} \leftarrow \hat{c} + w_{\{v_x,j\}}$
            $\hat{E}^* \leftarrow E^*_x \cup (\{v_x, j\} \cap E^*)$     // traversed augmenting edges at $j$
            if $j \neq t \wedge (M_{0j} = NULL \vee \hat{c} + c_j < \pi^{\mathrm{c}}_{M_{0j}})$ then     // expansion with
             relay at $j$
                $M_{0j} \leftarrow (\hat{c} + c_j, 0, x, j, \hat{E}^*)$
                $L \leftarrow L \cup \{M_{0j}\}$
            end
            if $M_{\hat{d}j} = NULL \vee \hat{c} < \pi^{\mathrm{c}}_{M_{\hat{d}j}}$ then  // expansion without relay at $j$
                $M_{\hat{d}j} \leftarrow (\hat{c}, \hat{d}, x, j, \hat{E}^*)$
                $L \leftarrow L \cup \{M_{\hat{d}j}\}$
            end
        end
    end
end
return $\arg\min_{x \in M_{dt}: 0 \leq d \leq d_{\max}} \pi^{\mathrm{c}}_x$
```

3.5 Computational Study

In this section we first give details on benchmark instances which are then used to compare the performance of the developed BP&C algorithms and to demonstrate their advantages and drawbacks.

3.5.1 Benchmark Instances

We consider three groups of benchmark instances: (1) instances from [30], (2) instances introduced by [100], and (3) an entirely new set of instances (ARLP), generated to reflect some of the real-world properties not covered by the previous two families.

Cabral instances. These instances have been introduced by [30], see Table 3.2 for an overview. They are extremely sparse 4-grid graphs in which each node is connected only to its direct vertical and horizontal neighbors. All edges have costs greater than zero (i.e., $E^0 = \emptyset$) and the maximum distance is equal to 70 for all instances. There are 180 instances in this family: for a fixed input graph and the given number of commodities, 10 instances are generated by sampling the set of commodities. All commodities share one node, i.e., we can reorder them such that $|S| = 1$. The number of nodes varies between 20 and 60. Although $|\mathcal{K}| \in \{5, 10\}$, we point out that for some instances the "effective" number of commodities is smaller than specified by the instance. This is due to two reasons. First, some instances contain the same commodity more than once. Second, some instances contain commodities for which the source and target are identical. Since commodities of this type are trivially connected by the empty path, we simply ignore them. Column $|\mathcal{K}|$ in Table 3.2 reports the average number of effective commodity pairs.

Konak instances. These instances, which were generated by randomly placing and connecting nodes on a grid, were originally introduced by Konak [100]. The number of nodes varies between 40 and 160. The length of each edge $\{i, j\}$ is set to the Euclidean distance between i and j while its cost is either set equal to the edge length (type I) or to $d_{\max} - d_{\{i,j\}}$ (type II). The basic instance properties ($|V|$, $|E^0|$, $|E^*|$, $|\mathcal{K}|$, and $d_{\max}$) are shown in Tables 3.3 and 3.4. Instances with an identical number of nodes and the same $d_{\max}$ are based on the same graph and only the number of commodities differs. There are 40 instances in total, 20 of each type. Notice that also for this family of instances, the number of commodities is extremely low ($|\mathcal{K}| \in \{5, 10\}$). In some instances free edges are present but their number is always rather small.

ARLP instances. This newly generated set of benchmark instances is intended to complement the previous two sets available from the literature. Both, Cabral and Konak instances assume $|\mathcal{K}| \in \{5, 10\}$. On the contrary, ARLP instances aim to simulate applications where many node pairs need to communicate. We refer to this set as augmented RLP (ARLP) instances since we require all nodes to communicate with each other, as it is the case for the RLP (cf. Section 3.1.2). In contrast to the RLP, the set of augmenting edges E^* is not empty, and in contrast to the Konak and Cabral instances, a significant number of zero cost edges exists.

The instances have been generated as follows. Nodes are placed randomly on a 100×100 grid and edges with length equal to the Euclidean distance (rounded up) between two nodes are added whenever this distance does not exceed 30. Each edge is chosen to be a free edge with probability 20, 50, or 80 % in instance subsets `20F`, `50F`, and `80F`, respectively. The costs w_{ij} of augmenting edges $\{i, j\}$ are chosen randomly according to a normal distribution with parameters $\mu = d_{ij}, \sigma = 5$ (rounded up). Relay costs are chosen randomly according to the normal distribution $\mu = 10 \cdot \bar{w}, \sigma = 20$ (rounded up) where $\bar{w}$ denotes the average cost of augmenting edges. Finally, $d_{\max} = 50$ for all instances and $\mathcal{K}$ contains all pairs that cannot be connected using solely free edges, i.e., $\mathcal{K} = \{(u, v) \mid (u, v) \in V \times V, u < v\} \setminus C^0$ (see Section 3.4.1).

In addition, a second set of instances (denoted as ARLP-p25) with a smaller number of commodities has been created. Each such instance is generated from an ARLP instance by adopting each commodity with a probability of 25 %. The main characteristics of sets

ARLP and ARLP-p25 are summarized in Tables 3.5 and 3.6, respectively. The instances are already preprocessed, in the sense that our preprocessing procedures do not apply. Especially, we only consider instances that are connected, i.e., they consist of a single connected component. Furthermore, we define the set $\mathcal{K}$ so that commodity pairs that can be connected only using free edges and without relays are not included.

The ARLP and the ARLP-p25 instance sets are available at https://www.ac.tuwien.ac.at/research/problem-instances/#Network_Design_Problem_with_Relays.

3.5.2 Computational Results

Test results reported in this section have been obtained on an Intel Xeon E5540 machine with 2.53 GHz. The computation time limit has been set to 7200 seconds and the memory limit to 8 GB RAM. As discussed in Section 3.4.3 we use a violation threshold of 0.5 when separating strengthening inequalities. This threshold is only considered when solving the problem to integer optimality. When reporting LP bounds in this section, we add all violated inequalities. This means that independent experiments are conducted for the two cases. This leads to situations in which the integer run finds an optimal solution but the LP run terminates due to the time or the memory limit as a result of excessive separation of cutting planes. Conversely, it is also possible that the LP gap is tighter than the final optimality gap of the integer run if the latter cannot progress fast enough.

Tables 3.2–3.6 summarize the results. Both models (MCF) and (CUT) are compared with respect to the LP relaxation gap (LP gap [%]), the final optimality gap (opt. gap [%]), the used computation time (t [s]), and the number of priced columns (columns). LP and optimality gaps are computed as $100 \cdot (UB^* - LB)/UB^*$ where UB^* is the best known upper bound and LB is the lower bound obtained by the respective algorithm. The upper bounds shown in the tables in column UB^* are printed bold iff the given value is shown to be the optimal objective value by any of the considered algorithms. Entries marked with “ML” indicate that an experiment has been terminated due to the memory limit. Furthermore, for some of the most challenging instances it was impossible to obtain a lower bound within the imposed time limit. In these cases the LP or optimality gap cannot be computed and respective fields are marked with “TL”. Finally, for the heuristic (CH1+) we report the percentage increase with respect to the best known upper bound given by $100 \cdot UB^H/UB^*$ where UB^H is the objective value obtained by (CH1+).

Cabral Instances

Mean values of the computational results obtained for instances from set Cabral are provided in Table 3.2. Each row corresponds to ten instances for the given instance graph and the number of commodities.

We first notice that both algorithms provide very small LP gaps, with the gaps from (CUT) being consistently smaller than those from (MCF). This can be explained by the fact that the big-M coefficients in inequalities (3.22) are equal to one since all commodities have the same source in this instance set. This advantage concerning the quality of LP

					LP		IP					
					Gap [%]		Opt. gap [%]		t [s]		Columns	
Instance	$\|V\|$	$\|E^*\|$	$\|\mathcal{K}\|$	(CH1+)	(MCF)	(CUT)	(MCF)	(CUT)	(MCF)	(CUT)	(MCF)	(CUT)
4A5B70L5K	20	31	4.5	100.2	**0.7**	**0.7**	**0.0**	**0.0**	**< 1**	**< 1**	**21**	24
4A5B70L10K	20	31	7.9	103.8	1.4	**1.2**	**0.0**	**0.0**	**1**	**1**	**34**	51
5A5B70L5K	25	40	4.2	105.0	1.6	**1.4**	**0.0**	**0.0**	1	**< 1**	**34**	39
5A5B70L10K	25	40	8.5	105.7	1.6	**1.5**	**0.0**	**0.0**	**1**	**1**	**49**	70
6A5B70L5K	30	49	4.7	101.3	0.6	**0.2**	**0.0**	**0.0**	**1**	**1**	**48**	50
6A5B70L10K	30	49	8.8	102.1	2.0	**1.5**	**0.0**	**0.0**	4	**3**	**78**	102
7A5B70L5K	35	58	4.5	101.6	1.4	**1.2**	**0.0**	**0.0**	**1**	2	57	**50**
7A5B70L10K	35	58	8.4	102.3	2.1	**1.7**	**0.0**	**0.0**	6	**5**	**98**	115
8A5B70L5K	40	67	4.7	103.3	1.6	**1.2**	**0.0**	**0.0**	4	**3**	90	**73**
8A5B70L10K	40	67	8.9	106.0	2.9	**2.6**	**0.0**	**0.0**	11	**7**	**127**	142
9A5B70L5K	45	76	4.8	105.3	**1.3**	**1.3**	**0.0**	**0.0**	4	**2**	88	**66**
9A5B70L10K	45	76	9.0	105.3	2.4	**1.9**	**0.0**	**0.0**	19	**8**	176	**158**
10A5B70L5K	50	85	5.0	103.5	2.2	**1.6**	**0.0**	**0.0**	8	**7**	127	**111**
10A5B70L10K	50	85	9.4	104.3	2.4	**2.3**	**0.0**	**0.0**	32	**14**	210	**184**
11A5B70L5K	55	94	4.6	100.8	1.9	**1.8**	**0.0**	**0.0**	5	**4**	103	**86**
11A5B70L10K	55	94	9.1	105.7	1.3	**0.4**	**0.0**	**0.0**	45	**11**	242	**170**
12A5B70L5K	60	103	4.7	104.7	1.2	**0.8**	**0.0**	**0.0**	9	**8**	142	**114**
12A5B70L10K	60	103	9.0	103.0	3.9	**2.6**	**0.0**	**0.0**	179	**26**	357	**252**

Table 3.2: Results on the Cabral instances. Column $|\mathcal{K}|$ reports the average number of effective commodity pairs. (CH1+) gives the ratio between the objective value obtained by the heuristic and the best known upper bound. We report the LP gap, the optimality gap, the total computation time in seconds (t [s]), and the number of priced columns. Each row reports a mean value over a set of ten instances. Best values are marked bold.

bounds carries over to the integral runs leading to significantly smaller computation times for the (CUT) model. Both models require a comparable number of columns to solve the instances to optimality. In general, the number of priced columns is rather low, which can be explained by the sparsity of the considered input graphs.

Our results constitute a clear improvement compared to the results reported by Cabral et al. [30] where these instances have been introduced. Whereas Cabral et al. [30] provide only heuristic solutions with relatively large optimality gaps (with up to 20 % with respect to their best-performing arc-path based formulation), we are able to solve all instances to provable optimality—in most cases within a few seconds only. Moreover, our (CUT) formulation features very small LP gaps that range between 0.2 % and 2.6 % on these instances.

Konak Instances

In contrast to the Cabral instances, for this data-set, the number of targets per given source does not exceed two (and is usually only one). Hence, the structure of optimal solutions on the communication graph is less arborescence-like, it is rather an intersection of multiple source-target walks. In Tables 3.3 and 3.4 we compare the performance of the proposed exact approaches for instances of type I and type II, respectively. We report the LP gaps, the final optimality gaps, the overall computation times and the numbers of priced columns. Among the instances of type I and II, 11 out of 20 and 18 out of 20 are solved to optimality, respectively. Interestingly, it turns out that instances with edge lengths equivalent to the costs (type I) are significantly harder to solve for our algorithms than those where edge costs and edge lengths are inversely correlated (type II).

Konak type I. We first compare the LP gaps reported in Table 3.3. Considering only those cases when both algorithms are able to finish the computation of the LP bound, we observe that the gaps of both formulations are the same. This is not surprising since the benefits of inequalities (3.22) diminish due to multiple source and target nodes in each instance. In general, these instances are much harder to solve for our algorithms than those from the Cabral set. The main reason is that the graphs are much denser which leads to higher separation and pricing efforts.

The performance for the complete runs (until finding an optimal integer solution or reaching the time limit) is consistent with the LP bound results. While the majority of instances with at most 60 nodes could be solved by both algorithms, the final optimality gaps are quite large for instances with 80 and more nodes. Both algorithms feature similar computation times with a slight advantage for the (MCF) model. The main advantage of the (CUT) formulation is that it can provide bounds for all instances. The (MCF), on the other hand, cannot provide bounds for the two most difficult instances within the time limit of two hours. We observe that our algorithms improve the initial upper bounds received from the heuristic (reported in the column (CH1+)) for all but two instances with 80 nodes and the instances with 160 nodes.

Konak type II. The results shown in Table 3.4, compared with those obtained for instances of type I, clearly indicate that for our algorithms type II instances are easier to solve than the type I instances. The above discussed relation between the two approaches remains roughly the same, both regarding the quality of the LP gaps and the overall performance. In total 18 out of 20 instances of this group could be solved to optimality by the (MCF), three more than by the (CUT) model. The LP gaps are much smaller than for the type I instances and for two instances the LP gap is even zero. The upper bounds obtained from (CH1+) have been shown to be optimal by our algorithms in two cases and have been improved by them for all remaining ones except the largest one (`160N_10K_35L`).

ARLP Instances

Recall that the Cabral and Konak instances contain very few commodities and almost no free edges. The influence of the ratio of free edges to augmenting edges on the proposed approaches as well as the influence of a larger number of commodities is therefore investigated on the set of ARLP instances. Results obtained for the ARLP instances

								LP		IP					
								Gap [%]		Opt. gap [%]		t [s]		Columns	
Instance	$\lvert V\rvert$	$\lvert E^0\rvert$	$\lvert E^*\rvert$	$\lvert\mathcal{K}\rvert$	d_{max}	UB^*	(CH1+)	(MCF)	(CUT)	(MCF)	(CUT)	(MCF)	(CUT)	(MCF)	(CUT)
40N_5K_30L	40	0	198	5	30	**473.80**	102.7	**11.9**	**11.9**	**0.0**	**0.0**	**16**	27	299	**227**
40N_5K_35L	40	0	272	5	35	**352.08**	102.7	**21.0**	**21.0**	**0.0**	**0.0**	1173	**533**	1012	**767**
40N_10K_30L	40	0	198	10	30	**518.98**	108.2	**15.9**	**15.9**	**0.0**	**0.0**	180	**152**	**401**	421
40N_10K_35L	40	0	272	10	35	399.36	112.3	**23.1**	**23.1**	**6.6**	8.0	7200	7200	1043	**1027**
50N_5K_30L	50	0	279	5	30	**283.79**	120.3	**0.0**	**0.0**	**0.0**	**0.0**	**1**	5	232	**186**
50N_5K_35L	50	0	372	5	35	**260.24**	100.0	**1.2**	**1.2**	**0.0**	**0.0**	**6**	27	565	**499**
50N_10K_30L	50	0	279	10	30	**540.39**	103.8	**13.0**	**13.0**	**0.0**	**0.0**	**158**	411	**540**	557
50N_10K_35L	50	0	372	10	35	**404.32**	111.9	**11.8**	**11.8**	**0.0**	**0.0**	**204**	261	**1177**	1307
60N_5K_30L	60	0	305	5	30	**509.12**	103.2	**21.5**	**21.5**	**0.0**	**0.0**	**39**	70	**509**	541
60N_5K_35L	60	0	412	5	35	**377.02**	105.8	**10.9**	**10.9**	**0.0**	**0.0**	**48**	94	**939**	1146
60N_10K_30L	60	0	305	10	30	**678.84**	107.0	**24.4**	**24.4**	**0.0**	**0.0**	**886**	1063	**703**	730
60N_10K_35L	60	0	412	10	35	**499.64**	112.9	**19.0**	**19.0**	**0.0**	**0.0**	**891**	1400	**1498**	1520
80N_5K_30L	80	0	641	5	30	353.86	105.2	**15.7**	**15.7**	**9.1**	10.0	7200	7200	3887	**3784**
80N_5K_35L	80	0	853	5	35	334.21	103.2	**19.9**	TL	**16.1**	16.7	7200	7200	9110	**8394**
80N_10K_30L	80	0	641	10	30	513.02	100.0	**36.7**	TL	**32.0**	35.2	7200	7200	**3613**	3815
80N_10K_35L	80	0	853	10	35	516.91	100.0	**43.0**	TL	44.4	**42.2**	7200	7200	**6952**	8592
160N_5K_30L	160	3	2770	5	30	298.31	100.0	**29.7**	TL	29.6	**28.2**	7200	7200	**15941**	23489
160N_5K_35L	160	3	3621	5	35	314.52	100.0	TL	**37.0**	58.1	**37.2**	7200	7200	**20303**	30877
160N_10K_30L	160	3	2770	10	30	470.54	100.0	TL	TL	TL	**44.1**	7200	7200	**11229**	23027
160N_10K_35L	160	3	3621	10	35	484.97	100.0	TL	TL	TL	**50.2**	7200	7200	**11872**	36042

Table 3.3: Results on the Konak instances (type I). Column UB^* provides the best known upper bounds, optimal bounds are marked bold. (CH1+) gives the ratio between the objective value obtained by the heuristic and the best known upper bound. We report the LP gap, the optimality gap, the total computation time in seconds (t [s]), and the number of priced columns. Best values are marked bold. If an algorithm failed to compute a lower bound due to the time limit, the respective gap entry is marked with "TL".

and all considered percentages of free edges (20 %, 50 %, or 80 % of all available edges) are summarized in Table 3.5. It is not surprising that the most difficult instances are those with only 20 % free edges and that the instances become significantly easier to solve as this percentage increases. Notice that, due to the huge number of commodities, the model (MCF), which performed quite well on the other two data sets, can now only deal with the smallest instances with 40 nodes and some instances with 50 nodes. For larger instances (MCF) always hits the memory limit due to its excessive size. Clearly, using the (CUT) model greatly helps to overcome this issue.

As before, (MCF) provides the same LP gaps as (CUT) whenever both models manage to terminate within the time limit. The ARLP instances are more challenging for the (CUT) model as well. This can be explained by the excessive number of cuts that need to be generated, especially for the calculation of LP bounds, where no violation threshold is used. Moreover, the effects of the connectivity cuts in the original graph diminish on this data set due to the presence of a substantial number of free edges. Overall, (CUT) is clearly the best-performing model for this data-set. It is able to find optimal solutions for 22 out of 24 instances. The (MCF) model, on the other hand, manages to solve only six instances to optimality. As for the other instance sets, the upper bounds obtained

								LP		IP					
								Gap [%]		Opt. gap [%]		t [s]		Columns	
Instance	$\lvert V\rvert$	$\lvert E^0\rvert$	$\lvert E^*\rvert$	$\lvert\mathcal{K}\rvert$	d_{max}	UB^*	(CH1+)	(MCF)	(CUT)	(MCF)	(CUT)	(MCF)	(CUT)	(MCF)	(CUT)
40N_5K_30L	40	1	197	5	30	**247.27**	100.1	**0.0**	**0.0**	**0.0**	**0.0**	**< 1**	1	111	**80**
40N_5K_35L	40	0	272	5	35	**111.30**	101.7	**7.1**	**7.1**	**0.0**	**0.0**	**9**	17	459	**314**
40N_10K_30L	40	1	197	10	30	**292.62**	100.1	**4.4**	**4.4**	**0.0**	**0.0**	**13**	18	293	**267**
40N_10K_35L	40	0	272	10	35	**140.51**	101.0	**7.7**	**7.7**	**0.0**	**0.0**	61	**45**	690	**625**
50N_5K_30L	50	1	278	5	30	**119.80**	100.0	**0.0**	**0.0**	**0.0**	**0.0**	**< 1**	1	**98**	104
50N_5K_35L	50	0	372	5	35	**155.57**	105.2	**1.0**	**1.0**	**0.0**	**0.0**	**2**	33	**422**	547
50N_10K_30L	50	1	278	10	30	**279.70**	100.6	**2.0**	TL	**0.0**	**0.0**	**3**	28	**247**	432
50N_10K_35L	50	0	372	10	35	**206.22**	102.1	**1.6**	TL	**0.0**	**0.0**	**32**	127	**828**	1303
60N_5K_30L	60	3	302	5	30	**317.32**	104.0	**14.2**	**14.2**	**0.0**	**0.0**	**7**	26	**337**	444
60N_5K_35L	60	0	412	5	35	**166.35**	100.0	**0.0**	**0.0**	**0.0**	**0.0**	**< 1**	3	**315**	446
60N_10K_30L	60	3	302	10	30	**414.32**	120.4	**12.2**	**12.2**	**0.0**	**0.0**	**52**	88	**444**	583
60N_10K_35L	60	0	412	10	35	**242.32**	100.1	**5.2**	**5.2**	**0.0**	**0.0**	**21**	53	**748**	1016
80N_5K_30L	80	2	639	5	30	**134.73**	108.6	**4.9**	**4.9**	**0.0**	**0.0**	**32**	167	**1455**	1705
80N_5K_35L	80	1	852	5	35	**104.04**	100.4	**2.0**	**2.0**	**0.0**	**0.0**	**46**	306	**2307**	2899
80N_10K_30L	80	2	639	10	30	**187.17**	105.0	**8.1**	**8.1**	**0.0**	**0.0**	695	**625**	2756	**2501**
80N_10K_35L	80	1	852	10	35	**168.62**	102.2	**11.2**	TL	**0.0**	12.9	**4382**	7200	**7947**	9114
160N_5K_30L	160	9	2764	5	30	**78.61**	106.2	**6.6**	**6.6**	**0.0**	4.2	**1491**	7200	**12812**	23985
160N_5K_35L	160	9	3615	5	35	**68.15**	101.4	**9.0**	**9.0**	**0.0**	6.7	**5273**	7200	**29676**	30003
160N_10K_30L	160	9	2764	10	30	112.06	106.1	**6.5**	TL	**5.7**	11.6	7200	7200	**16850**	26344
160N_10K_35L	160	9	3615	10	35	117.19	100.0	TL	TL	**10.3**	33.4	7200	7200	**22324**	34909

Table 3.4: Results on the Konak instances (type II). Column UB^* provides the best known upper bounds, optimal bounds are marked bold. (CH1+) gives the ratio between the objective value obtained by the heuristic and the best known upper bound. We report the LP gap, the optimality gap, the total computation time in seconds (t [s]), and the number of priced columns. Best values are marked bold. If an algorithm failed to compute a lower bound due to the time limit, the respective gap entry is marked with "TL".

from (CH1+) are improved for almost all instances by our algorithms which only fail to improve them for one of the most challenging problems (`80N50L20F_A`) and prove optimality of one solution obtained by (CH1+).

ARLP-p25 instances. The purpose of evaluating our algorithms on this family of instances was to study the influence of the number of commodities on the algorithmic performance. Recall that for the ARLP instances, each node pair is a commodity, i.e., $|\mathcal{K}|$ is in $O(|V|^2)$. For the ARLP-p25 instances, the number of commodities is reduced to a quarter. In general, the results indicate that as long as the number of commodities remains $O(|V|^2)$, the NDPR is much more difficult to solve than when the number of commodities is fixed to a small constant value (as for the Cabral and the Konak instances). Moreover, the remaining commodities still enforce solutions that guarantee full connectivity since the optimal objective values do not change for the corresponding instances. The detailed results are provided in Table 3.6.

Again, the LP gaps of both models are the same whenever both of them terminated within the time limit. However, this time more LP bounds have been obtained due to the smaller number of commodities. The quality of the bounds is roughly comparable to

							LP		IP					
							Gap [%]		Opt. gap [%]		t [s]		Columns	
Instance	$\lvert V\rvert$	$\lvert E^0\rvert$	$\lvert E^*\rvert$	$\lvert\mathcal{K}\rvert$	UB^*	(CH1+)	(MCF)	(CUT)	(MCF)	(CUT)	(MCF)	(CUT)	(MCF)	(CUT)
40N50L20F_A	40	26	124	724	**874**	122.5	TL	**10.7**	TL	**0.0**	7200	**142**	**132**	1340
40N50L20F_B	40	35	123	688	**874**	113.3	TL	**13.4**	TL	**0.0**	7200	**139**	**147**	1196
40N50L50F_A	40	89	78	513	**837**	103.9	**15.2**	**15.2**	**0.0**	**0.0**	2504	**14**	**59**	83
40N50L50F_B	40	71	72	586	**876**	108.3	**6.4**	**6.4**	**0.0**	**0.0**	2683	**7**	76	**74**
40N50L80F_A	40	146	32	443	**516**	100.0	**0.0**	**0.0**	**0.0**	**0.0**	10	**2**	1	**0**
40N50L80F_B	40	154	35	423	**777**	100.9	**6.6**	**6.6**	**0.0**	**0.0**	155	**5**	**18**	23
50N50L20F_A	50	44	212	1111	**815**	106.5	TL	**5.9**	ML	**0.0**	ML	**416**	ML	**3973**
50N50L20F_B	50	59	235	1022	**656**	131.2	TL	**2.8**	ML	**0.0**	ML	**72**	ML	**3032**
50N50L50F_A	50	132	157	719	**543**	104.4	**9.6**	TL	ML	**0.0**	ML	**11**	ML	**114**
50N50L50F_B	50	117	132	873	**775**	100.8	TL	**5.8**	ML	**0.0**	ML	**30**	ML	**211**
50N50L80F_A	50	175	51	788	**630**	143.8	**11.1**	TL	**0.0**	**0.0**	211	**10**	25	**7**
50N50L80F_B	50	212	58	682	**572**	119.2	**0.0**	**0.0**	**0.0**	**0.0**	29	**5**	1	**0**
60N50L20F_A	60	72	269	1549	**775**	111.1	ML	**5.5**	ML	**0.0**	ML	**257**	ML	**3685**
60N50L20F_B	60	63	268	1588	**976**	129.7	ML	**17.8**	ML	**0.0**	ML	**1397**	ML	**3725**
60N50L50F_A	60	216	204	1036	**628**	115.8	TL	ML	ML	**0.0**	ML	**44**	ML	**201**
60N50L50F_B	60	197	200	1103	**743**	103.0	TL	ML	ML	**0.0**	ML	**58**	ML	**255**
60N50L80F_A	60	311	85	854	**503**	119.7	**2.1**	TL	ML	**0.0**	ML	**16**	ML	**16**
60N50L80F_B	60	283	74	1041	**624**	110.1	**9.9**	ML	ML	**0.0**	ML	**18**	ML	**4**
80N50L20F_A	80	123	525	2729	1084	100.0	ML	**35.1**	ML	**34.0**	ML	7200	ML	**19111**
80N50L20F_B	80	124	545	2659	790	125.8	ML	TL	ML	**9.6**	ML	7200	ML	**21190**
80N50L50F_A	80	335	342	1916	**498**	120.9	ML	ML	ML	**0.0**	ML	**65**	ML	**71**
80N50L50F_B	80	366	375	1902	**541**	135.5	ML	TL	ML	**0.0**	ML	**159**	ML	**424**
80N50L80F_A	80	548	148	1834	**577**	101.4	ML	ML	ML	**0.0**	ML	**67**	ML	**5**
80N50L80F_B	80	597	158	1532	**549**	159.2	ML	ML	ML	**0.0**	ML	**85**	ML	**4**

Table 3.5: Results on the ARLP instances. Column UB^* provides the best known upper bounds, optimal bounds are marked bold. (CH1+) gives the ratio between the objective value obtained by the heuristic and the best known upper bound. We report the LP gap, the optimality gap, the total computation time in seconds (t [s]), and the number of priced columns. Best values are marked bold. Entries marked with "ML" indicate that an experiment has been terminated due to the memory limit. If an algorithm failed to compute a lower bound due to the time limit, the respective gap entry is marked with "TL".

those of the ARLP instances. The number of optimal solutions found does not change when reducing the number of commodities, i.e., optimal solutions have been found for 22 out of 24 instances. For the (MCF) the number of instances solved to optimality greatly increases from 6 to 14, mainly due to the smaller number of commodities reducing the size of the model. Surprisingly, the impact on the algorithmic performance of (CUT) is much smaller. In fact, in several cases the reduced instances are even harder to solve. As mentioned above the optimal solutions remain the same as for the original instances. Therefore, we "loose" constraints that might help to prove optimality earlier. This is particularly relevant to the (CUT) model where the number of variables is independent of the number of the commodity pairs. As before, one solution obtained from (CH1+) is shown to be optimal and all but one of the remaining upper bounds from (CH1+) are improved by our approaches.

							LP		IP					
							Gap [%]		Opt. gap [%]		t [s]		Columns	
Instance	$\lvert V\rvert$	$\lvert E^0\rvert$	$\lvert E^*\rvert$	$\lvert\mathcal{K}\rvert$	UB^*	(CH1+)	(MCF)	(CUT)	(MCF)	(CUT)	(MCF)	(CUT)	(MCF)	(CUT)
40N50L20F_A_p25	40	26	124	181	**874**	119.2	**11.4**	**11.4**	11.7	**0.0**	7200	**106**	**499**	1429
40N50L20F_B_p25	40	35	123	172	**874**	102.7	**14.6**	**14.6**	11.2	**0.0**	7200	**109**	**529**	1238
40N50L50F_A_p25	40	89	78	129	**837**	102.2	**15.2**	**15.2**	**0.0**	**0.0**	152	**10**	**58**	81
40N50L50F_B_p25	40	71	72	147	**876**	104.8	**6.4**	**6.4**	**0.0**	**0.0**	112	**3**	**80**	96
40N50L80F_A_p25	40	146	32	111	**516**	100.0	**0.0**	**0.0**	**0.0**	**0.0**	2	**1**	2	**0**
40N50L80F_B_p25	40	154	35	106	**777**	100.9	**6.6**	**6.6**	**0.0**	**0.0**	20	**2**	**16**	22
50N50L20F_A_p25	50	44	212	278	**815**	106.0	TL	**6.1**	TL	**0.0**	7200	**611**	**572**	4396
50N50L20F_B_p25	50	59	235	256	**656**	122.0	TL	**2.8**	TL	**0.0**	7200	**209**	**547**	4256
50N50L50F_A_p25	50	132	157	180	**543**	112.0	**9.6**	**9.6**	**0.0**	**0.0**	179	**5**	**142**	153
50N50L50F_B_p25	50	117	132	219	**775**	100.0	**5.8**	**5.8**	**0.0**	**0.0**	4431	**11**	273	**184**
50N50L80F_A_p25	50	175	51	197	**630**	124.0	**11.1**	**11.1**	**0.0**	**0.0**	26	**2**	17	**3**
50N50L80F_B_p25	50	212	58	171	**572**	119.2	**0.0**	**0.0**	**0.0**	**0.0**	6	**1**	**1**	3
60N50L20F_A_p25	60	72	269	388	**775**	116.3	TL	**5.5**	TL	**0.0**	7200	**2174**	**366**	4904
60N50L20F_B_p25	60	63	268	397	**976**	126.0	TL	**18.4**	TL	**0.0**	7200	**4412**	**343**	4037
60N50L50F_A_p25	60	216	204	259	**628**	115.8	**8.0**	TL	**0.0**	**0.0**	6777	**46**	**220**	237
60N50L50F_B_p25	60	197	200	276	**743**	122.3	**8.5**	**8.5**	4.7	**0.0**	7200	**45**	**242**	360
60N50L80F_A_p25	60	311	85	214	**503**	148.7	**2.1**	**2.1**	**0.0**	**0.0**	125	**7**	25	**24**
60N50L80F_B_p25	60	283	74	261	**624**	124.5	**9.9**	**9.9**	**0.0**	**0.0**	81	**16**	15	**14**
80N50L20F_A_p25	80	123	525	683	1095	100.0	TL	**35.8**	ML	**33.9**	ML	7200	ML	**19296**
80N50L20F_B_p25	80	124	545	665	1024	100.0	TL	TL	ML	**35.4**	ML	7200	ML	**21959**
80N50L50F_A_p25	80	335	342	479	**498**	120.9	**1.1**	TL	**0.0**	**0.0**	1890	**28**	**155**	193
80N50L50F_B_p25	80	366	375	476	**541**	141.6	TL	TL	TL	**0.0**	7200	**181**	**224**	538
80N50L80F_A_p25	80	548	148	459	**577**	100.2	**2.6**	TL	**0.0**	**0.0**	826	**40**	17	**14**
80N50L80F_B_p25	80	597	158	383	**549**	132.1	**9.7**	TL	**0.0**	**0.0**	1580	**55**	13	**12**

Table 3.6: Results on the ARLP-p25 instances. Column UB^* provides the best known upper bounds, optimal bounds are marked bold. (CH1+) gives the ratio between the objective value obtained by the heuristic and the best known upper bound. We report the LP gap, the optimality gap, the total computation time in seconds (t [s]), and the number of priced columns. Best values are marked bold. Entries marked with “ML” indicate that an experiment has been terminated due to the memory limit. If an algorithm failed to compute a lower bound due to the time limit, the respective gap entry is marked with “TL”.

Sensitivity analysis. The number of feasible walks realizing a connection of some commodity pair depends on the distance limit in relation to the edge lengths. More specifically, it depends on the average number of edges that can be part of a feasible subpath that uses no relays. In the following, we want to investigate the effect of this characteristic on our algorithms. To this end, we consider the Cabral instances and vary the distance limit $d_{\max}$.

The unmodified Cabral instances feature a distance limit of 70 and the edge lengths are chosen uniformly at random from the interval $[10, 30]$. Therefore, a feasible path of maximal length consists on average of three to four edges. Increasing $d_{\max}$ by a value of 30 means that a feasible path without relays may contain at least one additional edge. We consider distance limits between 40 and 220 in steps of 30 for our experiment. The results are visualized in terms of box plots in Figure 3.10 for the (CUT) model.

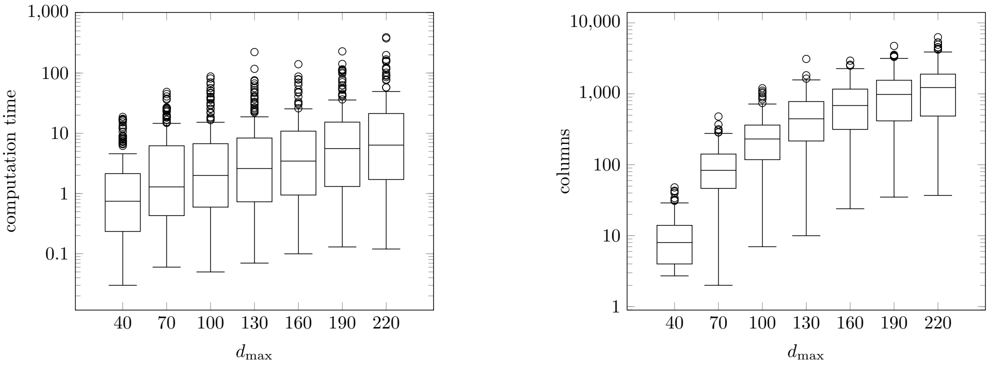


Figure 3.10: Sensitivity analysis regarding the distance maximum $d_{\max}$ on the Cabral instances for the (CUT) model. Each box considers 180 instances. Both box plots use a logarithmic scale.

Due to the increasing number of possible options for connecting the commodity pairs it can be expected that the instances become more challenging as the distance limit increases. This can be verified in terms of the box plots. However, we can also see that the computation times increase only moderately. Per step the median computation time rises by roughly one second. Similarly, the number of columns required to solve an instance to optimality increases. The respective results obtained with the (MCF) model are quite similar except that the baseline lies a bit higher. In total, we can conclude that the performance of our algorithms remains quite robust against changes to the distance limit.

Solution shape and characteristics. In this section, we analyze the structure of optimal solutions for two small examples corresponding to instances `40N30C50L20F_A` and `40N30C50L80F_A` from the ARLP set with 20 % and 80 % of free edges, respectively. In Figures 3.11a and 3.12a we visualize the two corresponding optimal solutions. We notice that for `40N30C50L20F_A`, despite the fact that there are 724 commodities, only three relays need to be installed (with a reasonable complement of augmentation edges) to enable these communications. In Figure 3.11, we compare the optimal solution of `40N30C50L20F_A` with the one obtained for the same input graph, but with a much smaller number of commodities: the set of commodities $\mathcal{K}'$ is obtained by removing all commodities except those containing the node that is involved in the fewest commodities. The structure of the obtained optimal solution is similar to the original one, with three installed relays and a comparable number of augmentation edges. Along with the results reported for the ARLP instances, this figure indicates that the estimated investment costs do not increase linearly with the number of commodities. In general there are higher investment costs for setting up the infrastructure, and once it is established, marginal increase in the number of commodities will not be reflected in the increase of the set-up costs (cf. the solution values in Tables 3.5 and 3.6).

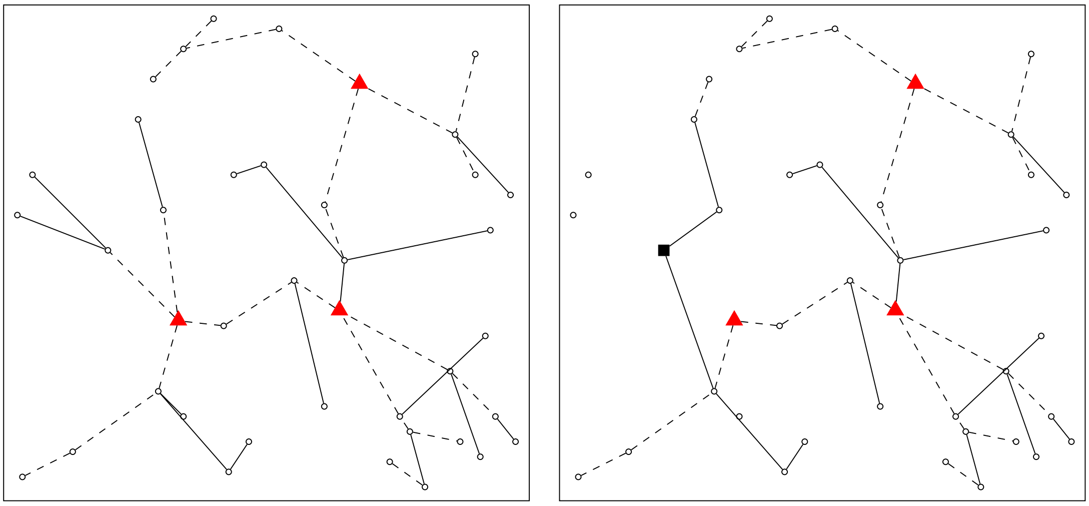
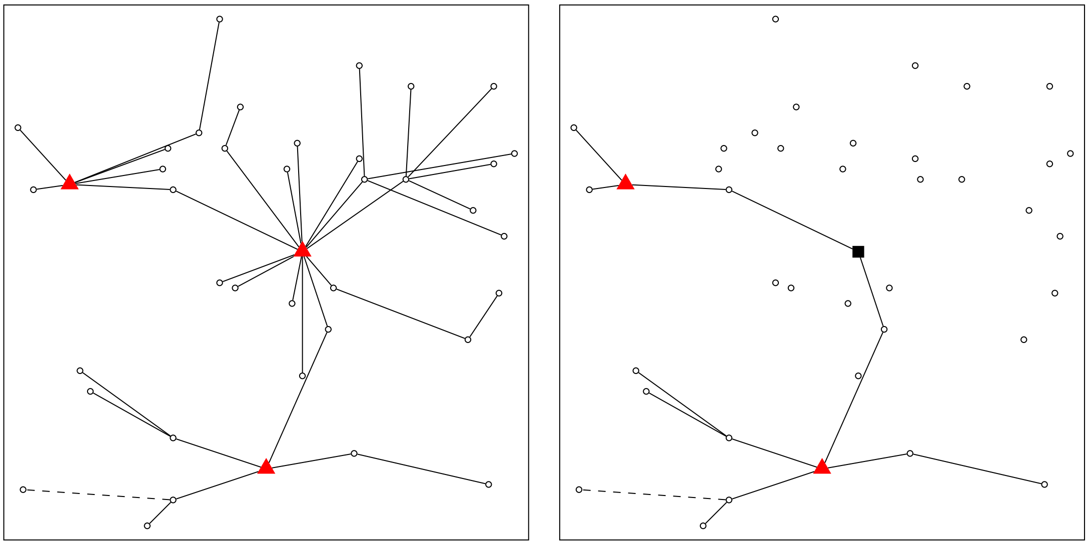

(a) 40N30C50L20F_A (original)
(b) 40N30C50L20F_A ($\mathcal{K}'$)

Figure 3.11: Solutions to ARLP instance 40N30C50L20F_A with different numbers of commodities. Solid lines indicate free edges and dashed lines augmenting edges. Selected relays are marked with triangles. On the right the single source is marked with a square.

Figure 3.12 visualizes the optimal solutions for `40N30C50L80F_A` and its "sparser" variant (with the set of commodities $\mathcal{K}'$ constructed as above), respectively. We observe that when 80 % of available edges are free, the need to install augmentation edges almost vanishes but a certain number of relays still needs to be installed to enable communications. Comparing the optimal solution for the sparser problem with the original one, we notice that the number of relays can be reduced, if the source node is centrally located, as in the shown example.

3.6 Conclusion

In this work we introduced new MILP formulations for solving the NDPR that utilize an exponential number of variables (and constraints). We proved several conditions and properties of optimal solutions and revised the concept of communication graphs for the NDPR to state our MILP formulations. Two BP&C algorithms have been developed. The first one is based on a multi-commodity flow formulation on an undirected communication graph whereas the second is based on a cut-set formulation on a directed communication graph. The computational study on instances from the literature and a newly created set of instances shows that the cut-set formulation on the directed communication graph has the overall best performance. The multi-commodity flow formulation on the undirected communication graph performs reasonably well, but only up to a limited number of nodes and/or commodities. Due to its excessive number of variables, the latter formulation exhibits serious memory issues that makes it less appealing for practical applications involving larger graphs or a higher number of commodities.



(a) 40N30C50L80F_A (original)

(b) 40N30C50L80F_A ($\mathcal{K}'$)

Figure 3.12: Solutions to ARLP instance 40N30C50L80F_A with different numbers of commodities. Solid lines indicate free edges and dashed lines augmenting edges. Selected relays are marked with triangles. On the right the single source is marked with a square.

We conducted a sensitivity analysis on the Cabral instances with different $d_{\max}$ restrictions. The results showed that our algorithms are quite robust against changes to this parameter featuring only a comparatively small increase in computation times and priced columns.

Compared to the previous attempt from the literature to develop an exact model for the NDPR by [30], the main advantage of our modeling approach is that we do not use variables corresponding to entire walks between commodities (including the decisions for placing relays). Instead, we consider only simple paths between two consecutive relays and model the decision where to place the relays through the communication graph. That way, our pricing draws a computational advantage from the fact that augmenting edges can be simultaneously shared by multiple commodities. The positive effect of our modeling approach is striking for instances with many commodities, where the need for the simultaneous reuse of augmenting edges is even more amplified.

3.6.1 Future Work

Besides telecommunication network design, the NDPR can be used to answer strategic questions in the context of electric mobility. A company running its operations based on a fleet of electric vehicles (EVs), be it a logistics company or an e-car provider, faces a difficult decision problem of planning the underlying charging infrastructure. Due to expensive EV batteries and their limited range, a stable and robust charging infrastructure is crucial for running the business suitably. Since building and/or renting charging stations is expensive, logistic companies or e-car providers are interested in minimizing the number of charging stations whilst enabling travel between specific locations (cf. commodities). Furthermore, in some metropolises (including Stockholm, Gothenburg, and Singapore) *congestion taxing* or *congestion charging* mechanisms are implemented. This means that shorter distances can be traversed (e.g., via shortcuts through the inner city), but in that case a certain road toll is to be paid(see, e.g., [58, 104]). Similarly, urban freeways passing through a downtown area can be subject to compulsory electronic toll (like the case in Santiago de Chile, or some Norwegian cities). Consequently, if a company is interested in building charging stations for its fleet, it also has to gauge whether toll roads are to be used. When making strategic decisions, costs for toll roads are typically estimated over a longer planning period and considered as fixed link costs.

Two main strategic design questions arising in this complex decision process are addressed by the NDPR: Given a family of origin-destination pairs EVs need to travel, and given the existing links that can be traversed:

1. What are the optimal locations for placing the charging stations and how many of them are needed?
2. Could the available infrastructure be enhanced by including additional links (shortcuts), to reduce the travel distances?

The relevance of the NDPR for planning EVs' charging infrastructure has not been sufficiently acknowledged in the existing literature. This is maybe due to the additional aspects that need to be taken into account when dealing with e-mobility. These include restricting the distance arising from detours necessary for vehicle recharging and the maximum number of (time-consuming) recharging stops an EV requires before reaching its final destination. Although the NDPR does not consider these additional aspects, there is no doubt that the problem plays an important role for e-mobility applications for two reasons: (1) the NDPR may appear as a subproblem (i.e., in some decomposition schemes), and (2) the proposed algorithms can be used to derive heuristic solutions in multi-phase approaches, where the complex decision process is approached step-by-step. Hence, the NDPR provides important insights for the companies running their business with a fleet of EVs. It helps in estimating the initial set-up costs (induced by the installation of recharging stations and potential purchases of road-toll passes). Moreover, by using a correlation between the edge lengths and lengths of the trips, the routing decisions obtained through an NDPR solution implicitly help in estimating a lower bound on the number of required EVs. Similarly, assuming that all trips will be covered, an upper bound on the expected profit can be calculated.

Interesting and more difficult NDPR variants that are important directions for future work include the following aspects: (1) limiting the maximum number of recharging stops (relays) used by a single commodity, (2) limiting the maximum waiting times imposed by recharging, or (3) limiting the overall trip length per commodity.

CHAPTER 4

Solving a Selective Dial-a-Ride Problem with Logic-based Benders Decomposition

In the previous chapter we employed column generation as solution approach. While being used for decades and belonging to the most well-known decomposition techniques for mixed integer linear programming (MILP) it is still very successful and achieves state-of-the-art results for various problems. The approach considered in this chapter also originates from a well-established technique, Benders decomposition (BD), which has been first proposed in the 1960s. BD received only moderate attention for several years before enjoying a renaissance since the 2000s, see [144]. During this time also an interesting extension, logic-based Benders decomposition (LBBD), was developed that will be considered in the following.

We propose an LBBD approach for solving a selective dial-a-ride problem (DARP). This problem considers transportation requests of people from pick-up to drop-off locations. Users specify time windows with respect to these points. Requests are served by a given vehicle fleet with limited capacity and maximum tour duration per vehicle. Moreover, user inconvenience considerations are taken into account by limiting the travel time between origin and destination for each request. In contrast to previous work we do not focus on travel cost minimization under the assumption that all requests can be satisfied but rather consider maximization of the number of served requests with a given vehicle fleet. This appears to be particularly relevant for funded systems that have to cope with overallocation.

In our study we in particular investigate the impact of strengthening the Benders cuts. We compare plain Benders cuts to heuristically strengthened ones, as well as two variants of theoretically strongest cuts. Moreover, we consider heuristic boosting techniques as well as valid inequalities to speed up solving the Benders master problem. The models are implemented as LBBD and also as Branch-and-Check (BaC) algorithms and empirically compared on a diverse set of benchmark instances.

This chapter has been published in *Computers & Operations Research*:

> M. Riedler and G. R. Raidl. Solving a selective dial-a-ride problem with logic-based Benders decomposition. *Computers & Operations Research*, 96: 30–54, 2018

4.1 Introduction

The DARP considers the design of vehicle routes for a set of customers who specify transportation requests from origin (pick-up) to destination (drop-off) points. Users typically impose time windows with respect to these locations. To reduce user inconvenience the time required to go from the pick-up to the drop-off location (ride time) is limited. The available requests shall be served by a fleet of vehicles. Each vehicle has a limited capacity corresponding to the number of customers that can be transported and a maximum total travel time. The restriction on the tour duration is important in order to deal with regulations regarding driver shifts.

As done by Jaw et al. [95], Cordeau [41], and others we distinguish between outbound and inbound requests. An outbound request considers the case that a customer wants to go from some starting location to a destination. An inbound request corresponds to the opposite case, i.e., a customer who wants to return to his/her starting location. According to the survey presented in [134] customers have different priorities with respect to the adherence to time windows. For outbound requests it is critical to stay within the time window at the drop-off location and for inbound requests it is important to adhere to the time window at the pick-up location.

In the literature several variants of the DARP have been investigated, see [42, 44, 135]. The two main variants are the static and the dynamic case. In the former it is assumed that all requests are known in advance whereas in the latter requests become known gradually over time and routes need to be adjusted accordingly. There are also mixed variants for which some requests are known in advance and some are revealed dynamically. Moreover, there is a distinction between the single- and the multi-vehicle case. In the former variant the requests have to be served using a single vehicle and in the latter multiple vehicles are available. In the following we deal with a variant of the static multi-vehicle DARP.

4.1.1 Outline and Discussion of the Contributions

In many DARP applications it is assumed that all requests can be served and that the total travel expenses together with the user inconvenience have to be minimized. In contrast, we consider the scenario that in general not all customers can be handled with the given fixed-size vehicle fleet and aim at maximizing the number of served requests. This is intended to deal with situations in which dial-a-ride systems are overallocated. In these cases serving as many customers as possible appears to be more relevant than savings due to shorter tour lengths. Of course user inconvenience considerations still have to be taken into account to provide reasonable service conditions.

We consider solution algorithms based on LBBD (see Hooker and Ottosson [91]) and BaC (see Thorsteinsson [161]). The Benders master problem focuses on the selection of requests and their assignment to vehicles. It is modeled as integer linear programming (ILP) that we enhance through valid inequalities originating from subproblem relaxations. For solving the Benders subproblems, which correspond to the route planning tasks, we consider MILP as well as constraint programming (CP) approaches. In particular, we also present a hybrid approach that combines MILP and CP. Several strategies for constructing Benders cuts are studied. We consider cuts derived from greedily obtained minimal infeasible request subsets, the full set of all minimal infeasible request subsets, as well as the set of all minimum cardinality infeasible request subsets and compare them to the unrefined cuts that are directly obtained from the subproblem assignments. Moreover, we consider heuristic boosting techniques to possibly speed up the solution process. To this end we terminate the master problem prematurely according to a specific termination criterion and use the suboptimal solution to derive Benders cuts. As soon as no further cuts can be obtained this way, we fall back to solving the master problem to optimality and continue with regular Benders iterations. This is necessary to obtain a provably optimal solution. As termination criterion we consider a decreasing sequence of thresholds for the optimality gap and an increasing sequence of time limits. Employing an adaptive approach we start at the first element of the sequence and move to the next one whenever no further cuts can be found with the current termination condition. A more flexible approach allows traversing the sequence in both directions, depending on whether cuts could be obtained or not. The suggested algorithms are tested extensively on a novel set of benchmark instances as well as on instances from the literature.

The remainder of the chapter is organized as follows. We first provide an overview of previous work in the area. Then, we give a formal definition of the specific problem variant, including a complexity discussion. In terms of the formal specification we provide a compact reference model that is a straightforward extension of the MILP from [41] for the tour-length-minimization DARP. In the main part we present the details of our decomposition approaches; including important implementation details. Finally, we discuss computational results on various test instances and conclude with an outlook on future research directions.

4.1.2 Previous Work

The DARP has a rather long research history. Among the first was the work by Psaraftis [138] that deals with the static single-vehicle variant. Sexton and Bodin [158, 159] solve the problem by splitting it into a routing and a scheduling phase which they formally describe in the context of BD. The routing is done by an insertion heuristic. In [24] the same authors use this approach to tackle the multi-vehicle case by first forming clusters of requests and then solving the single-vehicle problem for each cluster. Since they construct the clusters (grouping close customers) as well as the routes heuristically, neither method can guarantee optimal solutions. Later on, this approach for the multi-vehicle problem has been refined by using so-called “mini-clusters”, see [53, 55]. The most recent contribution by Ioachim et al. [92] relying on this technique shows the positive influence of using mathematical optimization methods to globally define the set of “mini-clusters”. The authors argue that more sophisticated techniques provide a significant advantage over simpler heuristic approaches. However, all of these algorithms are still heuristics.

Only few contributions so far do not minimize traveling costs. Wolfler Calvo and Colorni [171] maximize the number of served customers and consider a penalty term regarding user inconvenience. This term considers the relative ratio between the direct and the actual travel time. The authors consider a fast heuristic construction approach based on an auxiliary graph.

Berbeglia et al. [15] and Häme and Hakula [86] focus on feasibility checking of DARP instances. Similarly, also the large neighborhood search by Jain and van Hentenryck [93] has been tested as feasibility checking algorithm. Although we consider an optimization problem here, we are still concerned with feasibility checking when it comes to the Benders subproblems.

For a broader overview on the DARP we refer to the surveys by Cordeau and Laporte [44, 42] and Parragh et al. [135].

An optimization problem closely related to the DARP is the pickup and delivery problem with time windows (PDPTW). The main difference between the two problems is that the PDPTW primarily deals with the transportation of goods rather than persons. As a consequence, it does not consider user inconvenience and related concerns. In this area branch-price-and-cut (BP&C) approaches have been shown to be able to provide state-of-the-art results in terms of exact solution approaches, see Ropke and Cordeau [151] and Baldacci et al. [8]. For further details consider the survey conducted in [135].

Recently, also revenue maximizing variants of the PDPTW have been considered. In Qiu and Feuerriegel [140] and Qiu et al. [141] each transportation request is assigned a profit. The goal is then to identify a subset of requests to be served with a given heterogeneous vehicle fleet that maximizes the revenue, i.e., sum of profits minus transportation cost. The problem is solved using a graph search algorithm as well as a set packing formulation for the case of a homogeneous vehicle fleet. A similar scenario is also considered in Gansterer et al. [65] and solved with different metaheuristic approaches.

Somewhat related are also certain variants of the team orienteering problem. A contribution in this respect is from Baklagis et al. [7] who solve a variant considering pick-up and delivery with a branch-and-price (B&P) approach.

Finally, we want to review contributions that are relevant to our work from the methodological point of view, i.e., works that apply (logic-based) BD in the context of vehicle routing problems. Cire and Hooker [38] consider the home health care problem in which medical services need to be provided to patients. Each service is represented as a job and requires a certain minimal qualification level. The services are provided by nurses that travel to the patients. The aim is to design routes and shift plans such that all required services can be provided while minimizing the costs for the nurses' working hours. The problem is solved using LBBD. In the master problem the jobs are assigned to the nurses and the subproblems determine the actual shift plan and route per nurse. After solving a subproblem a cut is introduced into the master problem reflecting the cost of the assignment or prohibiting an infeasible allocation. In case of an infeasible subproblem it is often possible to strengthen the obtained cut by identifying a subset of assigned jobs that is the cause of the infeasibility. Moreover, a local search procedure is employed that tries to repair infeasible solutions by reassigning jobs to other nurses. The authors solve the master problem only heuristically and therefore optimal solutions cannot be guaranteed. In the computational study the LBBD approach is compared to a CP model which it outperforms clearly.

The bi-level vehicle routing problem (VRP) considers the distribution of goods in two stages. The goods are first transported from the main depot to satellite depots. Starting at each satellite depot the goods are then brought to the customers. This kind of VRP arises for example in newspaper distribution. Raidl et al. [145, 146] consider a bi-level VRP with a global restriction on the time until which all customers need to receive their goods. The assignment of customers to the satellite depots is pre-specified. Deliveries are carried out with a homogeneous fleet of vehicles with restricted capacity. The goal is to perform all deliveries within the time limit at minimal routing cost. Due to the structure of the problem routing costs at the first level as well as for every satellite depot can be considered independently. However, the levels are still interlinked via the global time limit. These properties provide a promising basis for the application of LBBD. Raidl et al. [145, 146] consider a decomposition approach in which the master problem determines the route from the main depot to the satellite depots. With the now fixed starting times at the satellite depots the corresponding routes can be computed independently as subproblems. Infeasibilities (due to the global time limit) are prevented by computing a minimal starting time for each satellite depot that guarantees the existence of a feasible route. Hence, only Benders optimality cuts are required. These cuts turn out to be quite strong here since routing costs can only be reduced given a smaller starting time at the respective depot. Raidl et al. [145] consider an exact variant of this decomposition, as well as a hybrid approach with either the master or the subproblems solved via metaheuristics, and a completely heuristic approach. In Raidl et al. [146] the hybrid approach is further refined by verifying and, if needed, correcting the heuristically added Benders cuts in a second phase. With this approach the obtained solution is guaranteed to be provably optimal and the solution process is much faster than the purely exact one.

4.2 Formulations

The DARP is defined on a directed graph $G = (N, A)$. Given n requests, node set N consists of two copies of the depot $\{0, 2n+1\}$, the set of pick-up locations $P = \{1, \dots, n\}$, and the set of drop-off locations $D = \{n+1, \dots, 2n\}$. A request corresponds to a pair $(i, n+i)$ such that $i \in P$ and $(n+i) \in D$. In the following, we occasionally identify requests by their corresponding pick-up locations. The load (e.g., the number of persons to be transported) at each pick-up location $i \in P$ is given by $q_i \geq 0$ and the same amount is to be unloaded at the drop-off location, i.e., $q_{n+i} = -q_i$. The service duration at each node $i \in N$ is given by $d_i \geq 0$. For the depot $q_0 = q_{2n+1} = d_0 = d_{2n+1} = 0$ holds. In addition, each node i has an associated time window $[e_i, l_i]$, $e_i < l_i$.

The set of arcs is defined as $A = \{(i,j) \mid (i = 0 \land j \in P) \lor (i, j \in P \cup D \land i \neq j \land i \neq n+j) \lor (i \in D \land j = 2n+1)\}$. The non-negative travel time of arc (i,j) is t_{ij} and the maximum user ride time is denoted by $L > 0$. We are given a set of vehicles K and every vehicle $k \in K$ has a maximum capacity $Q^k > 0$ and a maximum route duration $T^k > 0$. Moreover, we assume that a time horizon limited by T is given, i.e., all requests have to be served in the time window $[0, T]$.

The goal is to serve as many requests as possible respecting all time windows, precedence constraints, capacity restrictions, maximum route durations, and the maximum ride times.

4.2.1 Complexity

The original DARP has been shown to be $\mathcal{NP}$-hard (see Baugh et al. [12]) and we will show that the problem still remains $\mathcal{NP}$-hard under the modified scenario. Our proof is based on the traveling salesman problem (TSP) and the decision problem variant of the selective DARP (S-DARP-D). Both are provided in the following. The TSP is well-known to be $\mathcal{NP}$-hard, see Garey and Johnson [66].

Definition 4.1. *TSP [66]*
INSTANCE: Set C of m cities, distance $c_{ij} \in \mathbb{Z}_{>0}$ for each pair of cities $i, j \in C$, positive integer B.
QUESTION: Is there a Hamiltonian tour of C having length B or less?

Definition 4.2. *S-DARP-D*
INSTANCE: Selective DARP instance, positive integer n'.
QUESTION: Is there a feasible solution to the selective DARP serving at least n' requests?

Theorem 4.1. *The selective DARP is $\mathcal{NP}$-hard.*

Proof. We show $\mathcal{NP}$-hardness of the selective DARP via a reduction from the TSP to S-DARP-D. First, we create a request for each city in C setting $d_i = q_i = 0$ for the pick-up and drop-off locations. Moreover, a single vehicle with $Q^1 = T^1 = \infty$ is considered. For the pick-up nodes we set the time windows to $[0, B]$ and for the drop-off nodes we set the time windows to $[B+1, \infty]$. The maximum user ride time is assumed to be unrestricted, i.e., $L = \infty$. For i and j, both pick-up nodes, we set $t_{ij} = c_{ij}$. The remaining travel times are set to zero. There exists a Hamiltonian tour of C with length B or less iff the constructed S-DARP-D instance allows serving at least $|C|$ requests. ☐

Corollary 4.1. *The selective DARP remains $\mathcal{NP}$-hard when the triangle inequality holds for the travel times t_{ij}; including the even more specific cases of the L_1 (rectilinear) metric and the L_2 (Euclidean) metric.*

Proof. Use the transformation stated above, but start from a TSP instance with the respective properties. The TSP is known to be also $\mathcal{NP}$-hard under these conditions, see Garey et al. [67]. ☐

4.2.2 Compact Model

The following MILP model is a slightly modified variant of the one introduced in [41]. We are going to refer to it as compact model (CM). The difference is that we are maximizing the number of requests served, instead of minimizing travel costs.

We use binary variables x_{ij}^k for each arc $(i,j) \in A$ per vehicle $k \in K$. Moreover, variables B_i^k and Q_i^k are used to track for each vehicle $k \in K$ the beginning-of-service time and the load at node $i \in N$ after serving i, respectively. Finally, we use variables L_i^k to model the ride time of each request identified by its pick-up location $i \in P$ on vehicle $k \in K$. The model reads as follows:

$$\max \quad \sum_{k \in K} \sum_{(i,j) \in A : j \in P} x_{ij}^k \tag{4.1}$$

$$\text{subject to} \quad \sum_{k \in K} \sum_{(i,j) \in A} x_{ij}^k \leq 1 \qquad \forall i \in P, \tag{4.2}$$

$$\sum_{(i,j) \in A} x_{ij}^k - \sum_{(n+i,j) \in A} x_{n+i,j}^k = 0 \qquad \forall i \in P, \forall k \in K, \tag{4.3}$$

$$\sum_{j \in P} x_{0j}^k = 1 \qquad \forall k \in K, \tag{4.4}$$

$$\sum_{(j,i) \in A} x_{ji}^k - \sum_{(i,j) \in A} x_{ij}^k = 0 \qquad \forall i \in P \cup D, \forall k \in K, \tag{4.5}$$

$$\sum_{i \in D} x_{i,2n+1}^k = 1 \qquad \forall k \in K, \tag{4.6}$$

$$(B_i^k + d_i + t_{ij}) x_{ij}^k \leq B_j^k \qquad \forall (i,j) \in A, \forall k \in K, \tag{4.7}$$

$$(Q_i^k + q_j) x_{ij}^k \leq Q_j^k \qquad \forall (i,j) \in A, \forall k \in K, \tag{4.8}$$

$$B_{n+i}^k - (B_i^k + d_i) = L_i^k \qquad \forall i \in P, \forall k \in K, \tag{4.9}$$

$$B_{2n+1}^k - B_0^k \leq T^k \qquad \forall k \in K, \tag{4.10}$$

$$e_i \leq B_i^k \leq l_i \qquad \forall i \in N, \forall k \in K, \tag{4.11}$$

$$t_{i,n+i} \leq L_i^k \leq L \qquad \forall i \in P, \forall k \in K, \tag{4.12}$$

$$\max\{0, q_i\} \leq Q_i^k \leq \min\{Q^k, Q^k + q_i\} \qquad \forall i \in N, \forall k \in K, \tag{4.13}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall (i,j) \in A, \forall k \in K. \tag{4.14}$$

The objective function (4.1) determines the number of served requests by counting the selected arcs leading to pick-up nodes. Constraints (4.2) ensure that each request is served by at most one vehicle. These are the only differences to the original model. Constraints (4.3) guarantee that pick-up and drop-off of each request are served by the same vehicle. Equalities (4.4) to (4.6) ensure that each vehicle leaves the depot as well as each node it visits and that it finally returns to the depot. Constraints (4.7) and (4.8) enforce that the B and Q variables are set correctly. Note that in addition to tracking the beginning-of-service times these constraints also serve as a variant of Miller-Tucker-Zemlin constraints to prevent subtours. Equalities (4.9) calculate the ride time for each request and inequalities (4.10) limit the route duration for each vehicle. The remaining inequalities ensure that the used variables stay within their respective domains.

The quadratic constraints (4.7) and (4.8) can be linearized as follows:

$$B_i^k + d_i + t_{ij} - M_{ij}^k(1 - x_{ij}^k) \leq B_j^k \qquad \forall (i,j) \in A, \forall k \in K, \tag{4.15}$$

$$Q_i^k + q_j - W_{ij}^k(1 - x_{ij}^k) \leq Q_j^k \qquad \forall (i,j) \in A, \forall k \in K, \tag{4.16}$$

with the big-M constants set to $M_{ij}^k = \max\{0, l_i + d_i + t_{ij} - e_j\}$ and $W_{ij}^k = \min\{Q^k, Q^k + q_i\}$, respectively.

4.2.3 Decomposition Approach

For the decomposition approach we split the problem into a master problem and several subproblems. The master problem is responsible for assigning the requests to the vehicles. When an assignment has been identified, we generate one subproblem per vehicle to check if a feasible tour exists. The model reads as follows:

$$\text{(master)} \quad \max \quad \sum_{k \in K} \sum_{i \in P} y_i^k \tag{4.17}$$

$$\text{subject to} \quad \sum_{k \in K} y_i^k \leq 1 \qquad \forall i \in P, \tag{4.18}$$

$$\text{Benders cuts} \qquad \forall k \in K, \tag{4.19}$$

$$y_i^k \in \{0,1\} \qquad \forall k \in K, \forall i \in P. \tag{4.20}$$

The master problem maximizes the number of served requests. Constraints (4.18) ensure that each request is assigned to at most one vehicle. The Benders cuts (4.19) will be provided by the subproblems. They are responsible for preventing infeasible assignments of requests. Furthermore, we will later augment this basic master problem by initially provided valid inequalities originating from a relaxation of the subproblem.

We formulate the subproblems $sub(k, I)$ based on a vehicle $k \in K$ and a subset $I \subseteq P$ of the requests. Dependent on a solution $\bar{\mathbf{y}}$ to the master problem we identify for each vehicle $k \in K$ the set $I^k = \{i \in P \mid \bar{y}_i^k = 1\}$ of assigned requests. Each of these sets results in an independently solvable subproblem $sub(k, I^k)$. The subproblems can be stated similarly to the compact formulation introduced in the previous section and essentially constitute feasibility-based single-vehicle DARPs.

For subproblem $sub(k, I)$ let $P^I = I$ and $D^I = \{n+i \mid i \in I\}$ be the pick-up and drop-off locations corresponding to set I, resulting in a restricted set of nodes $N^I = \{0, 2n+1\} \cup P^I \cup D^I$. According to N^I we define the reduced arc set $A^I = A \setminus \{(i,j) \mid i \notin N^I \vee j \notin N^I\}$. The subproblems can now be modeled as follows:

$$
\begin{aligned}
(sub(k,I)) \quad \min \quad & 0 && && (4.21)\\
\text{subject to} \quad & \sum_{(i,j)\in A^I} x_{ij} = 1 && \forall i \in P^I \cup D^I, && (4.22)\\
& \sum_{j \in P^I} x_{0j} = 1, && && (4.23)\\
& \sum_{(j,i)\in A^I} x_{ji} - \sum_{(i,j)\in A^I} x_{ij} = 0 && \forall i \in P^I \cup D^I, && (4.24)\\
& \sum_{i \in D^I} x_{i,2n+1} = 1, && && (4.25)\\
& B_i + d_i + t_{ij} - M_{ij}^k(1 - x_{ij}) \le B_j && \forall (i,j) \in A^I, && (4.26)\\
& Q_k + q_j - W_{ij}^k(1 - x_{ij}) \le Q_j && \forall (i,j) \in A^I, && (4.27)\\
& B_{n+i} - (B_i + d_i) = L_i && \forall i \in P^I, && (4.28)\\
& B_{2n+1} - B_0 \le T^k, && && (4.29)\\
& e_i \le B_i \le l_i && \forall i \in N^I, && (4.30)\\
& t_{i,n+i} \le L_i \le L && \forall i \in P^I, && (4.31)\\
& \max\{0, q_i\} \le Q_i \le \min\{Q^k, Q^k + q_i\} && \forall i \in N^I, && (4.32)\\
& x_{ij} \in \{0,1\} && \forall (i,j) \in A^I. && (4.33)
\end{aligned}
$$

The objective function (4.21) is constant since we are only interested whether there exists a feasible tour or not, i.e., this is actually a decision problem. In each subproblem all assigned requests I have to be served. We no longer need to enforce that pick-up and drop-off locations are visited by the same vehicle since we consider only one vehicle. It is sufficient to use constraints (4.22) for ensuring that the pick-up and drop-off locations of all assigned requests are visited. The remaining parts stay the same.

In addition to the MILP formulation we also provide a CP model similar to the one introduced in [15] but restricted to the single vehicle case and slightly adjusted. To formulate element constraints we define a bijective function $\pi\colon N^I \to \{0, \dots, 2 \cdot |I| + 1\}$ mapping the nodes required in the subproblem to a consecutive range as follows. Depot node 0 is mapped to itself and depot copy $2n+1$ is mapped to $2 \cdot |I| + 1$. Nodes in P^I are mapped to $\{1, \dots, |I|\}$ and those in D^I to $\{|I|+1, \dots, 2 \cdot |I|\}$ such that $\pi(i) = j$ iff $\pi(i+n) = j + |I|$, $\forall i \in P^I$. Accordingly, we define sets $\widetilde{P}^I = \{\pi(i) \mid i \in P^I\}$, $\widetilde{D}^I = \{\pi(i) \mid i \in D^I\}$, and $\widetilde{N}^I = \widetilde{P}^I \cup \widetilde{D}^I \cup \{0, 2 \cdot |I| + 1\}$. Additionally, we specify an appropriately reduced travel time matrix $\widetilde{t}_{ij} = t_{\pi^{-1}(i)\pi^{-1}(j)}$, $\forall (i,j) \in \widetilde{N}^I \times \widetilde{N}^I$ and load vector $\widetilde{q}_i = q_{\pi^{-1}(i)}$, $\forall i \in \widetilde{N}^I$. Observe that the remaining input (service duration and time windows) is not part of element constraints and therefore does not need transformed data structures.

To provide the model we use three sets of variables which are successor variables $s[i]$, $\forall i \in \widetilde{N}^I \setminus \{2 \cdot |I| + 1\}$, load variables $q[i]$, $\forall i \in \widetilde{N}^I$, and beginning-of-service time variables $b[i]$, $\forall i \in \widetilde{N}^I$. The model reads as follows:

$$(sub(k, I))$$

$$\text{allDifferent}(s), \tag{4.34}$$

$$b[i] + \widetilde{t}_{i,|I|+i} + d_{\pi^{-1}(i)} \leq b[|I| + i] \qquad \forall i \in \widetilde{P}^I, \tag{4.35}$$

$$b[i] + \widetilde{t}_{i,s[i]} + d_{\pi^{-1}(i)} \leq b[s[i]] \qquad \forall i \in \widetilde{N}^I \setminus \{2 \cdot |I| + 1\}, \tag{4.36}$$

$$b[i+n] - (b[i] + d_{\pi^{-1}(i)}) \leq L \qquad \forall i \in \widetilde{D}^I, \tag{4.37}$$

$$b[2 \cdot |I| + 1] - b[0] \leq T^k, \tag{4.38}$$

$$q[i] + \widetilde{q}_{s[i]} = q[s[i]] \qquad \forall i \in \widetilde{N}^I, \tag{4.39}$$

$$s[i] \in \{j \mid (i,j) \in A^I\} \qquad \forall i \in \widetilde{N}^I \setminus \{2 \cdot |I| + 1\}, \tag{4.40}$$

$$\text{domain}(b[i], e_{\pi^{-1}(i)}, l_{\pi^{-1}(i)}) \qquad \forall i \in \widetilde{N}^I, \tag{4.41}$$

$$\text{domain}(q[i], \widetilde{q}_i, Q^k) \qquad \forall i \in \widetilde{P}^I, \tag{4.42}$$

$$\text{domain}(q[i], 0, Q^k + \widetilde{q}_i) \qquad \forall i \in \widetilde{D}^I, \tag{4.43}$$

$$q[0] = 0, \tag{4.44}$$

$$q[2 \cdot |I| + 1] = 0. \tag{4.45}$$

Constraints (4.34) ensure that each node has a unique successor. Assuming that the triangle inequality holds, we know that the beginning-of-service times at the pick-up location and the corresponding drop-off location differ at least by the direct travel time (4.35). Constraints (4.36) model the time needed to travel from a node to its immediate successor. Inequalities (4.37) and (4.38) restrict the ride time and tour duration appropriately. The load restrictions are considered by constraints (4.39). The remaining constraints specify the variable domains.

Benders Cuts

If a subproblem turns out to be infeasible, we need to add a cut preventing that the requests that caused the infeasibility are again assigned to the same route in subsequent iterations. The easiest way to do this is to add a Benders cut preventing the exact same assignment and any superset of it.

In iteration j we denote by I_j^k the requests assigned to vehicle $k \in K$ and by $\overline{K}_j = \{k \in K \mid sub(k, I_j^k) \text{ is infeasible}\}$ the set of vehicles for which the subproblem turns out to be infeasible. The corresponding Benders cuts are:

$$\sum_{i \in I_j^k} y_i^k \leq |I_j^k| - 1 \quad \forall k \in \overline{K}_j. \tag{4.46}$$

This basic cut, however, frequently can be strengthened as it is likely that the infeasibility is caused by a proper subset of the assigned requests. Similar to the notation used in [40] we classify sets I_j^k as follows:

Definition 4.3. *We call a set of requests* I_j^k infeasible *iff subproblem* $sub(k, I_j^k)$ *is infeasible and* feasible *otherwise.*

Definition 4.4. *We call an infeasible set of requests* I_j^k irreducible infeasible set (IIS) *iff the removal of any request turns it into a feasible set. Otherwise, we call* I_j^k reducible infeasible set*.*

Reducible infeasible sets lead to unnecessarily weak Benders cuts. Therefore, we never want to add cuts that are based on reducible infeasible sets. In general, there exist several IISs of smaller cardinality for each reducible infeasible set I_j^k. All such sets are by definition minimal and, thus, lead to non-dominated cuts within this class of cuts. Note that the IISs with respect to a given base set can have different cardinality. For practical reasons it makes sense to prefer smaller sets when the number of IISs gets large. Moreover, each Benders cut prevents assignments that are supersets with respect to its underlying IIS. Hence, IISs of minimum cardinality are in general able to cut-off larger parts of the search space.

Unfortunately, there is neither an efficient way to compute all IISs nor those of minimum cardinality. However, by means of a greedy strategy (similar to what is done in [90]) we are at least able to reduce a given base set to an IIS efficiently, see Algorithm 4.1. The algorithm tries to remove requests one after another and checks each time if the resulting set is still infeasible. If this is the case, we keep the smaller set, otherwise we proceed with the next request.

Note that the order in which the requests are considered has in general a strong influence on the outcome of the algorithm. As mentioned before, smaller IISs are usually preferable as they cut off larger parts of the search space. The greedy strategy cannot guarantee to compute a set of minimum cardinality. Consequently, we should attempt to order the requests heuristically to increase the chances of ending up with a small set. Unfortunately, it is not trivial to find an appropriate ordering that can be computed quickly. One strategy would be to prioritize the removal of requests that are unlikely to be the cause of the infeasibility. However, identifying these requests is again difficult. Following

Algorithm 4.1: Greedy set reduction

```
Input: set $I$ of requests and vehicle $k \in K$ such that $sub(k, I)$ is infeasible.
foreach $i \in I$ do
    if $sub(k, I \setminus \{i\})$ is infeasible then
        $I = I \setminus \{i\}$
    end
end
return $I$                                  // $I$ is now an IIS
```

preliminary experiments, we finally decided in favor of low computation times by just keeping the natural order of the requests. To decrease the chances of ending up with bad results we apply the greedy reduction twice, the second time in reverse order and add both obtained cuts if they are distinct. The quality of heuristically computed sets is analyzed by comparing to variants in which we add cuts for all IISs as well as only for those of minimum cardinality.

Cuts obtained for one vehicle can also be added to the master problem for other vehicles with equally or more restrictive characteristics:

Definition 4.5. *We define a partial order on the vehicles denoted by* $\leq^{\mathrm{k}}$*:*

$$k_1 \leq^{\mathrm{k}} k_2 \Leftrightarrow (Q^{k_1} \leq Q^{k_2}) \wedge (T^{k_1} \leq T^{k_2}) \quad \forall k_1 \in K, k_2 \in K.$$

We can add Benders cuts for all vehicles with at most the capacity and the maximum tour length of the vehicle for which the infeasibility has been detected:

$$\sum_{i \in I_j^k} y_i^{k'} \leq |I_j^k| - 1 \quad \forall k \in \overline{K}_j, \forall k' \in K : k' \leq^{\mathrm{k}} k. \tag{4.47}$$

4.3 Algorithmic Framework

We start with some remarks on preprocessing that help to reduce the size of the problem instances for certain cases. Then, we present details for our algorithms and further techniques for speeding up the solving process.

4.3.1 Preprocessing

In this section we describe the used preprocessing techniques. They are based on the concepts introduced in [41]. We point out our modifications.

Time Window Tightening

In [41] several techniques for time window tightening are introduced. For outbound requests we can set the time window at the pick-up location to $e_i \leftarrow \max\{e_i, e_{n+i} - L - d_i\}$ and $l_i \leftarrow \min\{l_{n+i} - t_{i,n+i} - d_i, l_i\}$. Similarly, we set the time windows for drop-off nodes of inbound requests to $e_{n+i} \leftarrow \max\{e_{n+i}, e_i + d_i + t_{i,n+i}\}$ and to $l_{n+i} \leftarrow \min\{l_i + d_i + L, l_{n+i}\}$. The time windows on depot copies 0 and $(2n+1)$ are set to $e_0 = e_{2n+1} \leftarrow \min_{i \in P \cup D}\{e_i - t_{0i}\}$ and $l_0 = l_{2n+1} \leftarrow \max_{i \in P \cup D}\{l_i + d_i + t_{i,2n+1}\}$.

We consider a minor modification to avoid unwanted effects when requests are too close to the depot, i.e., $t_{0i} > e_i$ or $l_i + t_{i,2n+1} > T$. In these cases we might end up with increasing the time horizon $[0, T]$. To avoid this we additionally apply $e_i \leftarrow \max\{e_i, t_{0i}\}$ and $l_{n+i} \leftarrow \min\{l_{n+i}, T - t_{n+i,2n+1}\}$ for $i \in P$. Afterwards it is safe to tighten the time windows at the depot nodes as described. Alternatively, this can be taken into account during the following arc elimination.

Arc Elimination

As done in [41] we also eliminate arcs from A that cannot be part of a feasible solution. The following situations are considered:

- arcs $(0, n+i)$, $(i, 2n+1)$, and $(n+i, i)$ are infeasible for $i \in P$ (this is already considered by the definition of the arc set),
- arc (i, j) is infeasible if $e_i + d_i + t_{ij} > l_j$,
- arcs (i, j) and $(j, n+i)$ with $i \in P$, $j \in N$ are both infeasible if $t_{ij} + d_j + t_{j,n+i} > L$,
- arc $(i, n+j)$ with $i, j \in P$ is infeasible if path $(j, i, n+j, n+i)$ is infeasible as there is no other feasible path using that arc while serving i and j,
- symmetric to the previous condition arc $(n+i, j)$ with $i, j \in P$ is infeasible if path $(i, n+i, j, n+j)$ is infeasible,
- arc (i, j) with $i, j \in P$ is infeasible if paths $(i, j, n+i, n+j)$ and $(i, j, n+j, n+i)$ are both infeasible as the path can only be infeasible due to the arc itself or a time window violation when reaching either of the drop-off locations; visiting further nodes may only increase the degree of violation,
- symmetric to the previous condition arc $(n+i, n+j)$ with $i, j \in P$ is infeasible if paths $(i, j, n+i, n+j)$ and $(j, i, n+i, n+j)$ are both infeasible.

When checking the feasibility of paths, we also need to compute the *forward time slack*. In [43] the forward time slack F_i at node i in a path from i to q is computed as follows:

$$F_i = \min_{i \le j \le q} \left\{ \sum_{i < p \le j} W_p + \max\left\{0, \min\left\{l_j - B_j, L - P_j\right\}\right\} \right\}, \tag{4.48}$$

where W_i denotes the waiting time at node i and P_i represents the ride time for the request with destination node $i \in D$. For the remaining i we define $P_i = -\infty$. The second term of the inner minimum-function, i.e., $L - P_j$, is required to prevent any requests from exceeding the maximum user ride time.

If time windows of the requests do not prevent vehicles from returning too late to the depot, i.e., $l_i + t_{i,2n+1} > T$ for $i \in P$, we augment the paths considered above by including the depot $(2n+1)$ as final node. Similarly, it makes sense to add depot 0 as first node if $t_{0i} > e_i$ can be the case for some pick-up locations $i \in P$. If this is not done, we might miss detecting some infeasibilities. This can happen due to a too early beginning-of-service time in the former case and due to a too high forward time slack in the latter case.

Infeasible Request Pairs

As stated in [41] two requests $(i, n+i)$ and $(j, n+j)$ cannot be served by the same vehicle if all possible paths serving the two requests turn out to be infeasible. According to the precedence constraints the following paths have to be considered:

$$\begin{gathered}
(i, j, i+n, j+n), \\
(i, j, j+n, i+n), \\
(i, i+n, j, j+n), \\
(j, i, i+n, j+n), \\
(j, i, j+n, i+n), \\
(j, j+n, i, i+n).
\end{gathered}$$

Observe that a request is only feasible (assuming that the triangle inequality holds) if the direct connection between pick-up and drop-off is feasible. Therefore, we assume that both $(i, i+n)$ and $(j, j+n)$ are feasible since it makes no sense to consider per se infeasible requests. Thus, it is sufficient to check if at least one of the following options is available:

$$\begin{array}{lllll}
(i, j) & \wedge & (j, i+n) & \wedge & (i+n, j+n), \\
(i, j) & \wedge & (j+n, i+n), & & \\
(i+n, j), & & & & \\
(j, i) & \wedge & (i+n, j+n), & & \\
(j, i) & \wedge & (i, j+n) & \wedge & (j+n, i+n), \\
(j+n, i). & & & &
\end{array}$$

If none of them is possible, these two requests cannot be served by the same vehicle. As a consequence, this allows the removal of all arcs between the nodes associated with the pick-up and drop-off locations of requests i and j.

Let C be the set of all incompatible request pairs identified by their pick-up locations, i.e., $C \subseteq \{(i,j) \mid (i,j) \in P \times P, i < j\}$. Then, we can add the following constraints to the master problem:

$$y_i^k + y_j^k \leq 1 \quad \forall k \in K, \forall (i,j) \in C. \tag{4.49}$$

These constraints are essentially instances of Benders cuts for which the set of infeasible requests has cardinality two. Therefore, these are the smallest non-dominated cuts of type (4.46). Such cuts are particularly valuable due to their strong expected impact on the model. They are enumerated exhaustively and added to the initial formulation. We also add comparable constraints to the compact model using the sum of outgoing arcs for nodes i and j instead of the assignment variables.

In [41] the incompatible request pairs are used to fix certain requests to vehicles. This cannot be done here since it is unknown which requests will be served and which will be rejected.

4.3.2 Subproblem Relaxations

In this section we describe the used subproblem relaxations which are incorporated into the master problem in terms of valid inequalities. The purpose of these constraints is to integrate subproblem knowledge into the master problem to avoid poor assignments in earlier iterations where only few Benders cuts are present.

Capacities

We consider pairs of requests that are guaranteed to be together on the vehicle if served within the same tour. Based on these "overlapping requests" we construct a conflict graph to derive clique inequalities.

Definition 4.6. *Request* $(i, n+i)$ overlaps *with request* $(j, n+j)$ *for* $i, j \in P$ *if there exists a feasible path serving the two requests but paths* $(i, i+n, j, j+n)$ *and* $(j, j+n, i, i+n)$ *are both infeasible.*

Informally this means that two requests overlap if they can be served by a single vehicle but not in strict succession.

We then define graph $G^{\mathrm{C}} = (V^{\mathrm{C}}, E^{\mathrm{C}})$ with $V^{\mathrm{C}} = P$ and $E^{\mathrm{C}} = \{\{i,j\} \mid \text{request } (i, i+j) \text{ overlaps with request } (j, n+j), i \in P, j \in P, i \neq j\}$. In this graph we identify all maximal cliques. This can be done by the Bron-Kerbosch algorithm, see [28]. The cliques in G^{C} define sets of requests that have to be on board together when served by the same vehicle. We now need to determine whether all of them fit in the vehicle simultaneously. For each maximal clique and each vehicle $k \in K$ we sum up the loads of the corresponding requests, starting with the smallest one until we exceed the vehicle capacity. Then, we know the maximum number of requests in the clique that can be served by this vehicle.

Let $\mathcal{C}$ be the set of all maximal cliques in G^{C}. For each $C \in \mathcal{C}$ let k_C^k be the maximum number of requests in C that fit into vehicle $k \in K$. Then, we can add the following inequalities to the master problem:

$$\sum_{i \in C} y_i^k \leq k_C^k \quad \forall C \in \mathcal{C} : k_C^k < |C|. \tag{4.50}$$

Observe that these cuts are similar to the Benders cuts (4.46) introduced before. However, the difference between k_C^k and $|C|$ can be larger than one and thus these cuts are in general distinct.

Note that if there are several vehicles with the same capacity, these constraints need only be computed once per capacity variant. As the graph G^{C} is typically sparse, it is reasonable to search for all maximal cliques. Since the number of these cuts is usually not that large, we add all of them to the initial formulation.

Again, we also add this type of constraints to the compact model using the sum of outgoing arcs instead of the assignment variables.

Computing a Lower Bound on the Tour Duration

We compute for each node $i \in N$ the minimal time required to reach the next node, i.e., $t_i^{\min} = \min_{(i,j) \in A} t_{ij}$. If we consider a subset $I \subseteq P$ of the requests (given by the respective pick-up nodes), we can compute a lower bound on the time required to serve all requests as follows:

$$t_R^{\min} = t_0^{\min} + \sum_{i \in I} (t_i^{\min} + d_i + t_{n+i}^{\min} + d_{n+i}). \tag{4.51}$$

This relaxation gives us a (frequently weak) lower bound on the time required to serve the requests in I. We use this value to state the following constraints in the Benders master problem:

$$t_0^{\min} + \sum_{i \in P} y_i^k (t_i^{\min} + d_i + t_{n+i}^{\min} + d_{n+i}) \leq T^k \quad \forall k \in K. \tag{4.52}$$

This bound can be improved in certain cases. If $t_i^{\min}$ and $t_{n+i}^{\min}$ refer to the same target node v' (i.e., $t_i^{\min} = t_{iv'}$ and $t_{n+i}^{\min} = t_{n+i,v'}$), we consider the closest successors for i and $(n+i)$ excluding v'. We then choose the successor nodes resulting in the combined shorter distance $t_i^{\min} + t_{n+i}^{\min}$ and update the $t^{\min}$ values accordingly. If neither i nor $(n+i)$ has an outgoing arc to a node different from v', then the request is infeasible. This type of constraints is not considered for the compact model since tour duration restrictions are already explicit there.

4.3.3 Implementation of the Decomposition Approaches

The decomposition approach introduced in Section 4.2.3 can be implemented using LBBD or BaC. Algorithm 4.2 shows the basic functionality of the LBBD algorithm (ignoring Lines 12 to 15 for the moment). Remember that our decomposition approach uses only feasibility cuts, i.e., the subproblems do not directly contribute to the master problem's objective function. This means that LBBD either terminates with an optimal solution or no solution at all. BaC, on the other hand, relies on regular branch-and-cut (B&C) which means that it computes lower and upper bounds and tries to close the gap between them. Therefore, it usually provides a feasible solution prior to proving optimality. To make this also possible for LBBD we employ a repair heuristic (Line 13) to derive feasible solutions from intermediate infeasible master assignments, possibly even closing the optimality gap allowing premature termination. Details on the used repair heuristic will be given below.

Algorithm 4.2: Logic-based Benders algorithm

```
1  $j \leftarrow 0$                                         // iteration counter
2  repeat
3      $j \leftarrow j + 1$
4      feasible ← true
5      solve master problem
6      foreach $k \in K$ do
7          if $sub(k, I_j^k)$ is infeasible then
8              add Benders cuts to the master problem
9              feasible ← false
10         end
11     end
12     if feasible = false then
13         repair()      // construct feasible solution heuristically
           // check whether optimality gap could be closed
14         if obj(master problem) = obj(repair) then  feasible ← true
15     end
16 until feasible = true ∨ time limit reached
   // if feasible=true then optimal solution found
   // else potentially suboptimal, repaired solution
```

In Figure 4.1 we illustrate a simple iteration of the Benders algorithm. We consider three requests and one vehicle. The instance properties are shown in Figure 4.1a. To keep the example simple without terminating immediately we do not consider valid inequalities for the master problem here. Time window tightening and arc elimination have been applied to obtain a smaller graph, see Figure 4.1b. Initially the master problem assigns all requests to the single vehicle. This turns out to be infeasible. Once we try to reduce the identified infeasible assignment $\{1, 2, 3\}$, we find out that subsets $\{2, 3\}$ (Figure 4.1c) and $\{1, 3\}$ (Figure 4.1d) are IISs of minimum cardinality. However, subset $\{1, 2\}$ (Figure 4.1e) is feasible. Therefore, we add Benders cuts that prevent requests 2 and 3 as well as 1 and 3 to be in the same tour, respectively. In the second master iteration requests 1 and 2 are assigned to the vehicle. Now we are able to identify a feasible tour for the subproblem (Figure 4.1f) and the algorithm terminates with an optimal solution serving requests 1 and 2 but rejecting request 3.

Benders Cuts

In our experiments in Section 4.4 we will consider four strategies for constructing Benders infeasibility cuts: variant *simple* uses the unmodified master assignment, *aIIS* uses all IISs that can be obtained from the initial assignment, *mIIS* uses all IISs of minimal cardinality, and *gIIS* uses heuristically computed IISs. The IISs for variants aIIS and mIIS are computed using bottom-up enumeration by extending an initially empty set with the assigned requests until it becomes infeasible (including appropriate pruning for the minimum cardinality variant). Variant gIIS applies Algorithm 4.1 once in ascending and once in descending natural order of the request indices to obtain two IISs. If both turn out to be equivalent, the second set is discarded. As there is no connection between the order of request indices and their properties, this means that there is no strategic decision involved.

Repair Heuristic

Similarly as done in [38] we use a repair heuristic to construct feasible solutions based on infeasible assignments obtained from the Benders master problem. To this end we consider the input sets I^k, $k \in K$, with some of them possibly being infeasible. We construct a solution for each vehicle by assigning requests to it one at a time. If a request can be served by the vehicle, it is assigned to that vehicle, otherwise skipped. We first try to insert the requests selected by the Benders master problem. This step simplifies if the related subproblem turned out to be feasible because we can directly add all requests in this case. Afterwards we consider the unassigned requests. Requests that could not be served are added to the pool of unassigned requests and might be used by the remaining vehicles. Algorithm 4.3 provides details.

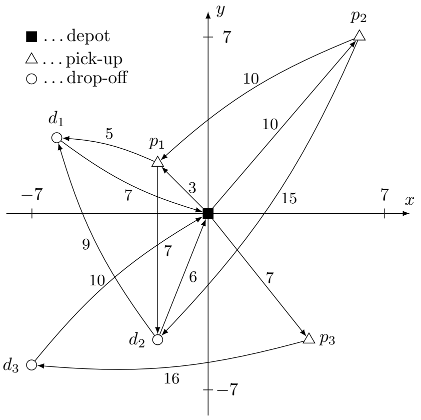
Note that the order in which the requests are considered has a significant impact on the outcome of the algorithm. However, the Benders master problem already makes a selection which provides (especially in later iterations) a reasonable starting assignment from which typically only few requests need to be removed. Hence, we avoid sorting the requests to save computation time since repair operations need to be performed rather frequently and thus execution speed is critical. For the same reasons we avoid a second pass over the vehicles that might be profitable due to freed-up requests.

Subproblem

In Section 4.2.3 we introduced the MILP and CP formulations for the Benders subproblems. The former is a compact model and can be implemented in a straightforward way. For

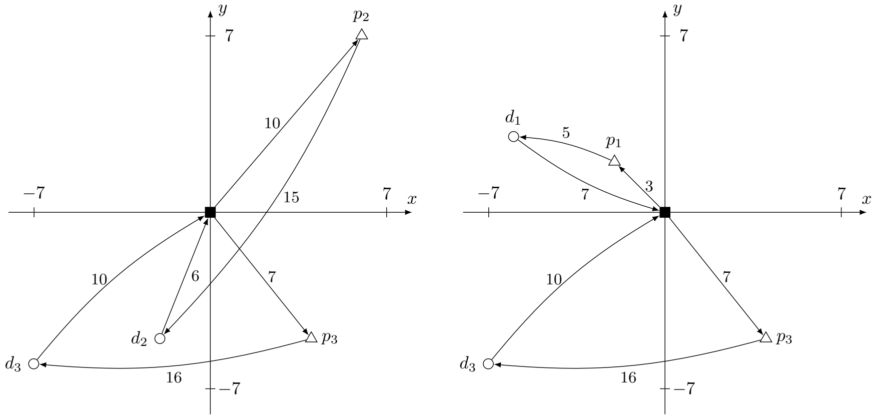
	Coordinates		Time window			
Node	x	y	e	l	q	d
depot	0	0	0	60	0	0
p_1	-2	2	17	30	1	3
d_1	-6	3	40	55	-1	3
p_2	6	7	12	25	1	3
d_2	-2	-5	35	50	-1	3
p_3	4	-5	32	41	1	3
d_3	-7	-6	45	50	-1	3

$|K| = 1$, $T^1 = 60$, $Q^1 = 3$, $L = 20$



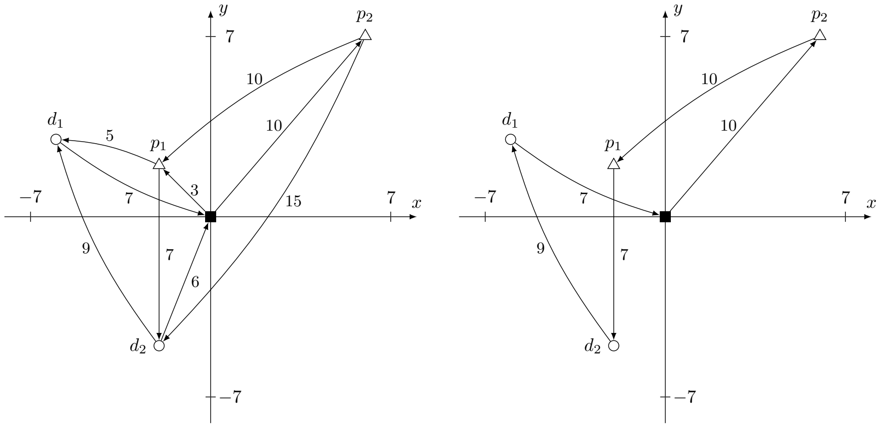
(a) Input data

(b) Preprocessed graph



(c) Considering requests 2 and 3

(d) Considering requests 1 and 3



(e) Considering requests 1 and 2

(f) Optimal solution

Figure 4.1: A simple Benders iteration without valid inequalities for the master problem. There exists no feasible tour visiting all three requests. The combination of request 3 with either of the remaining two turns out to be infeasible. Requests 1 and 2 can be served together which also constitutes the unique optimal solution.

Algorithm 4.3: Repair heuristic

```
Input: set $P$ of requests, identified by the pick-up locations
Input: sets $I^k$, $k \in K$ of potentially infeasible assignments
Output: pairwise disjoint feasible sets $I^k$, $k \in K$ of requests assigned to the
        vehicles
$F \leftarrow P \setminus \bigcup_{k \in K} I^k$                // set of unassigned requests
foreach $k \in K$ do
    if $sub(k, I^k)$ is feasible then
        $\tilde{I}^k \leftarrow I^k$
        $I^k \leftarrow \emptyset$
    else
        $\tilde{I}^k \leftarrow \emptyset$
        foreach $i \in I^k$ do
            if $sub(k, \tilde{I}^k \cup \{i\})$ is feasible then
                $I^k \leftarrow I^k \setminus \{i\}$
                $\tilde{I}^k \leftarrow \tilde{I}^k \cup \{i\}$
            end
        end
    end
    foreach $i \in F$ do
        if $sub(k, \tilde{I}^k \cup \{i\})$ is feasible then
            $F \leftarrow F \setminus \{i\}$
            $\tilde{I}^k \leftarrow \tilde{I}^k \cup \{i\}$
        end
    end
    $F \leftarrow F \cup I^k$        // unused requests might be assigned to the
     other vehicles
    $I^k \leftarrow \tilde{I}^k$
end
```

the CP model we additionally incorporated the custom branching heuristic presented by Berbeglia et al. [15]. Their approach branches on the successor variables $s[i]$, prioritizing variables with minimum cardinality domains. Ties are broken by counting the number of appearances of each value within all minimum cardinality domains, choosing the variable for which the sum of appearance counts of the values of its domain is maximal. We use no custom value selection heuristic and always pick the minimum value of the domain of the variable on which is branched.

4.3.4 Heuristic Boosting

Empirical tests have shown that the master problem of the LBBD approach frequently finds good or even optimal solutions fast. Afterwards, a significant amount of time is typically spent to close the relative gap between lower (LB) and upper (UB) bound, i.e., the optimality gap $(\mathrm{UB} - \mathrm{LB})/\mathrm{LB}$. However, closing the gap might not be required to obtain an intermediate solution yielding high-quality Benders cuts. Note the similarity to BaC which also derives Benders cuts from potentially suboptimal solutions encountered during the B&C search. The following sections describe our approaches exploiting this observation.

Gap Boosting

To reduce the time spent on closing the optimality gap of the Benders master problem we terminate the solving process when the optimality gap falls below a certain threshold. This is done until no further Benders cuts can be found with this strategy. Then, we proceed with regular Benders iterations without premature termination, i.e., using a threshold of zero, until no additional Benders cuts can be identified. Thus, we still obtain an optimal solution but might save time that is “wasted” on closing the optimality gap.

The difficulty is to choose a suitable threshold for premature termination, especially in earlier iterations. Using a large threshold has higher potential for speedup but can also lead to significantly worse intermediate solutions. Correspondingly we might obtain weaker Benders cuts, implying a larger number of master iterations.

To overcome the limitations of using a single threshold, we consider a more sophisticated adaptive approach based on a decreasing sequence of thresholds. Initially we start with the largest threshold and then switch to the subsequent smaller one every time no further cuts can be identified. Apart from this iterative variant we consider an *up-and-down* approach that allows the gap threshold to adjust in both directions. As before, we switch to the next smaller threshold whenever no further cuts can be identified. In addition, we now also switch back to the previous larger threshold if cuts could be added, for details see Algorithm 4.4. To preserve optimality the smallest threshold needs to be zero (or a sufficiently small numerical constant). For the used threshold values see Section 4.4.2. Preliminary experiments have shown that using a small sequence of specific values is superior to a more fine-grained approach, e.g., based on a geometric/arithmetic series.

Algorithm 4.4: Adaptive gap boosting (up-and-down)

```
Input: decreasing sequence of gap thresholds $g = (g_m, g_{m-1}, \dots, g_1)$ with
       $g_i > g_{i-1}$ for $1 < i \leq m$
$i \leftarrow m$
while termination criteria not met do
    solve Benders master problem until a relative optimality gap $\leq g_i$ is reached
    derive and add Benders cuts
    if no Benders cuts found $\wedge\, i > 1$ then  $i \leftarrow i - 1$
    else if no Benders cuts found $\wedge\, i = 1$ then  terminate // optimal
    else if Benders cuts found $\wedge\, i < m$ then  $i \leftarrow i + 1$
end
```

A similar—yet different—approach is considered in Tran and Beck [163]. There the authors also terminate the master problem prematurely according to a threshold on the optimality gap. However, their motivation is to complete at least a single master iteration from which a heuristic solution is derived. In contrast, our approach is designed in an optimality preserving way while speeding up the overall computation.

Observe that the considered objective function is integral. Thus, it is also possible to specify a threshold for the absolute optimality gap ($\mathrm{UB} - \mathrm{LB}$) instead of the relative one. Preliminary experiments have shown that the behavior is roughly the same when choosing comparable thresholds.

Time Limit Boosting

Early termination based on the optimality gap helps to reduce time spent in the Benders master problem. However, the amount of time that is used still might vary substantially. As an alternative we may directly limit the time allowed to be spent on finding a solution to each master problem instance. However, a fixed time limit might not accommodate for the increasing size of the master problem due to the Benders cuts. To deal with this we again consider a more flexible adaptive approach. In the beginning we use the smallest value of an increasing sequence of time limits and switch to the next larger one whenever no additional cuts can be found. Again, we consider a variant in which the time limit is adjusted in both directions. Optimality is preserved by using the total remaining time as final value of the sequence. For the used time limits see Section 4.4.2.

Solving the Subproblems Heuristically

We further tried to improve the solving of the subproblems by first using heuristics as done in [145]. To this end we employed a simple iterative algorithm that attempts to find a feasible route for the requests assigned to a vehicle during the Benders iterations. The algorithm constructs a route by inserting nodes sequentially, prioritizing those with the smallest amount of time left in their service window or the least remaining ride time.

Due to the heuristic nature of the algorithm we can accept the result if a feasible route has been found. However, if no valid route can be computed, we still have to check with an exact approach whether this result is correct. Preliminary tests have shown that the employed heuristic required such exact reevaluations quite frequently, outweighing the provided speedup from the positive cases.

4.4 Computational Study

In this section we are going to present the computational results for the considered algorithms with their variants. We start by giving details on the used test instances and the motivation for their selection. Then, we provide details on the actually used configuration. Finally, we present the obtained results.

4.4.1 Test Instances

The most commonly used benchmark instances for the classical DARP are those by Cordeau and Laporte [43] and Cordeau [41]. The first set is mainly interesting for testing with heuristics due to the large number of requests considered. Therefore, we decided to use the latter. The mentioned work considers instances of up to 48 requests and 4 vehicles. In Ropke et al. [152] this set got extended with instances of up to 96 requests and 8 vehicles. The properties of this instance set are shown in Table 4.1.

Berbeglia et al. [15] consider variants of these instances with modified maximum user ride times of $L = 30$ and $L = 22$, respectively, and a variant with 75% of the originally available vehicles. We consider the original instances and two of the modified variants excluding the one with $L = 22$ because it turned out that this modification makes some requests infeasible, i.e., not even a tour containing no other requests is feasible[1]. The unmodified instances are guaranteed to be feasible, i.e., it is known in advance that all requests can be served. Therefore, they are only of minor relevance for testing our algorithms. As shown in Berbeglia et al. [15] and Häme and Hakula [86] also most of the modified instances are feasible.

Under these premises we decided to generate further, for our scenario more relevant, instances. We aim at two aspects: First, we require instances that are more challenging from the “packing” perspective. This means that it is not guaranteed that all requests can be served. We accomplish this by choosing the number of requests large in relation to the length of the time horizon. Second, the existing instance set is too diverse to precisely measure the impact of certain instance properties. In particular, we are interested in instances with different degrees of utilization, i.e., the number of requests compared to the number of available vehicles. To this end, we consider scenarios with four and five vehicles and a (small) fixed time horizon while only varying the number of available requests.

---

[1] For an example see request 21 in instance a6-60.

Instance	n	$\lvert K\rvert$	L	Q	T	Instance	n	$\lvert K\rvert$	L	Q	T
a2-16	16	2	30	3	480	b2-16	16	2	45	6	480
a2-20	20	2	30	3	600	b2-20	20	2	45	6	600
a2-24	24	2	30	3	720	b2-24	24	2	45	6	720
a3-18	18	3	30	3	360	b3-18	18	3	45	6	360
a3-24	24	3	30	3	480	b3-24	24	3	45	6	480
a3-36	36	3	30	3	720	b3-36	36	3	45	6	720
a4-16	16	4	30	3	240	b4-16	16	4	45	6	240
a4-24	24	4	30	3	360	b4-24	24	4	45	6	360
a4-32	32	4	30	3	480	b4-32	32	4	45	6	480
a4-40	40	4	30	3	600	b4-40	40	4	45	6	600
a4-48	48	4	30	3	720	b4-48	48	4	45	6	720
a5-40	40	5	30	3	480	b5-40	40	5	45	6	480
a5-50	50	5	30	3	600	b5-50	50	5	45	6	600
a5-60	60	5	30	3	720	b5-60	60	5	45	6	720
a6-48	48	6	30	3	480	b6-48	48	6	45	6	480
a6-60	60	6	30	3	600	b6-60	60	6	45	6	600
a6-72	72	6	30	3	720	b6-72	72	6	45	6	720
a7-56	56	7	30	3	480	b7-56	56	7	45	6	480
a7-70	70	7	30	3	600	b7-70	70	7	45	6	600
a7-84	84	7	30	3	720	b7-84	84	7	45	6	720
a8-64	64	8	30	3	480	b8-64	64	8	45	6	480
a8-80	80	8	30	3	600	b8-80	80	8	45	6	600
a8-96	96	8	30	3	720	b8-96	96	8	45	6	720

Table 4.1: Properties of the instances by Ropke et al. [152]. Per instance vehicle capacities as well as the maximum route durations are the same for all vehicles. The maximum route durations are the same as the time horizon. In group “a” all requests have a load of $q_i = 1$ and a service time of $d_i = 3$. For group “b” the loads are chosen uniformly at random from $\{1, \ldots, 6\}$ with proportional service times $d_i = q_i$.

The new instance are generated according to the procedure mentioned in [41]. We first place nodes randomly on a 20×20 grid; the depot is located in the center of this grid at coordinates $(0, 0)$. Travel times between the nodes are set to the Euclidean distances between the corresponding points. For each instance with n requests the first $n/2$ requests are considered to be outbound requests and the remaining ones are inbound requests. For the former we fix the time window at the drop-off location and derive the time window at the pick-up location and for the inbound requests we fix the time window at the pick-up location and derive the time window at the drop-off location.

We consider a time horizon of $T = 240$ which corresponds to a half working day (assuming minutes as unit of time). For outbound requests we set the time window at the drop-off location $(n + i)$ by first choosing e_{n+i} uniformly at random from the interval $[t_{0i} + d_i + t_{i,n+i}, T - t_{n+i,2n+1} - d_{n+i} - 15]$ and then set $l_{n+i} = e_{n+i} + 15$. This guarantees that the time window has a fixed length of 15. Furthermore, it ensures that we can always return feasibly to the depot. Similarly, we choose for inbound requests e_i of pick-up node i from the interval $[t_{0i}, T - t_{n+i,2n+1} - d_{n+i} - t_{i,n+i} - d_i - 15]$ and set $l_i = e_i + 15$. The remaining time windows are then tightened as described in Section 4.3.1. For each request we assume a unit load of $q_i = -q_{n+i} = 1$ and the service duration is $d_i = d_{n+i} = 3$ for $i \in P$. The maximum user ride time is set to $L = 30$. We consider different numbers of homogeneous vehicles with capacity $Q^k = 3$ and maximum route duration $T^k = T$. Table 4.2 provides an overview of the properties of the generated test instances. In the following we are going to refer to this instance set as “SDARP” instances. The SDARP instances are available at https://www.ac.tuwien.ac.at/research/problem-instances/#Dial-a-Ride_Problem.

To deal consistently with the Euclidean distances in the MILP and CP algorithms we restrict the precision to two fractional digits for both instance sets.

4.4.2 Computational Experiments

In this section we are going to present the computational results of our algorithms obtained on the introduced benchmark instances. The test runs have been executed on an Intel Xeon E5540 with 2.53 GHz. The execution time limit has been set to 7200 seconds and the memory limit up to 8 GB RAM. Test runs have been executed using CPLEX 12.7.1 with a single thread using dual simplex and traditional B&C. The CP part has been implemented using Gecode 5.1.0 [69], also utilizing only a single thread for each test run. For the Bron-Kerbosch algorithm we used the implementation from Boost 1.63.0. Since objective values are known to be integral, runs terminate once the absolute optimality gap falls below a threshold of $1 - n \cdot \varepsilon$, where ε is the reduced-cost optimality tolerance of the MILP solver.

We start by investigating the different approaches for solving the subproblems and generating Benders cuts. Then, we evaluate the heuristic boosting techniques before providing further insights regarding specific properties of our algorithms. For these parts we rely on our newly generated SDARP instances. Afterwards, we test our algorithms on

Instance	n	$\|K\|$	L	Q	T	Instance	n	$\|K\|$	L	Q	T
30N_4K_A	30	4	30	3	240	44N_5K_A	44	5	30	3	239
30N_4K_B	30	4	30	3	240	44N_5K_B	44	5	30	3	240
30N_4K_C	30	4	30	3	240	44N_5K_C	44	5	30	3	240
30N_5K_A	30	5	30	3	240	50N_4K_A	50	4	30	3	240
30N_5K_B	30	5	30	3	240	50N_4K_B	50	4	30	3	240
30N_5K_C	30	5	30	3	240	50N_4K_C	50	4	30	3	240
40N_4K_A	40	4	30	3	240	50N_5K_A	50	5	30	3	240
40N_4K_B	40	4	30	3	240	50N_5K_B	50	5	30	3	240
40N_4K_C	40	4	30	3	232	50N_5K_C	50	5	30	3	240
40N_5K_A	40	5	30	3	240	60N_4K_A	60	4	30	3	240
40N_5K_B	40	5	30	3	240	60N_4K_B	60	4	30	3	240
40N_5K_C	40	5	30	3	240	60N_4K_C	60	4	30	3	240
44N_4K_A	44	4	30	3	240	60N_5K_A	60	5	30	3	240
44N_4K_B	44	4	30	3	240	60N_5K_B	60	5	30	3	240
44N_4K_C	44	4	30	3	240	60N_5K_C	60	5	30	3	240

Table 4.2: Properties of the newly generated SDARP instance set. Per instance vehicle capacities as well as the maximum route durations are the same for all vehicles. The maximum route durations are the same as the time horizon. All requests have a load of $q_i = 1$ and a service time of $d_i = 3$.

the instances used in the previous literature and provide a comparison to the compact MILP reference model.

We try to condense the presented results as much as possible to highlight the core results. For more details we provide additional tables in Appendix A. Table 4.3 summarizes the abbreviations used to identify the algorithm variants.

Evaluation of Subproblem Algorithms and Cut Generation Strategies

In several of the upcoming figures we provide sums of the total number of served requests per algorithm. Due to the integrality of the objective function value, it is sometimes difficult to distinguish the marks when the time limit is reached. To resolve this issue we provide the respective numbers in Table 4.4. In addition, this table also provides results for algorithms that have been omitted from the figures to improve readability.

Figure 4.2 gives an overview of the computation times and lower bounds of the algorithms without the use of heuristic boosting techniques. The subproblems are solved using the MILP model.

A first observation is that the relative performance of the variants for computing Benders cuts is quite similar for both decomposition approaches. As expected, the naïve strategy performs quite poorly. It solves the fewest instances to optimality and also takes much

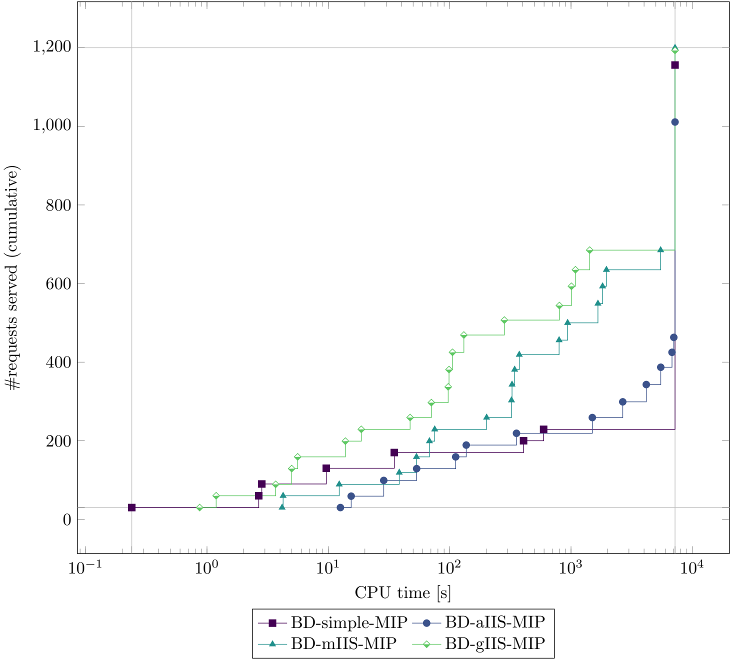
Abbreviation	Description
BaC	Branch-and-Check
LBBD	logic-based Benders decomposition
simple	unrefined Benders cuts
aIIS	Benders cut per IIS
mIIS	Benders cut per minimum cardinality IIS
gIIS	Benders cuts for up to two greedily computed IISs
MIP	MILP subproblems
CPMIP	combined CP-MILP subproblems
Hgaprel_it	heuristic boosting with thresholds for the relative optimality gap (iterative)
Hgaprel_ud	heuristic boosting with thresholds for the relative optimality gap (up-and-down)
Htime_it	heuristic boosting with time limits (iterative)
Htime_ud	heuristic boosting with time limits (up-and-down)
Compact	compact MILP model

Table 4.3: Summary of the abbreviations used to identify the tested algorithms.

Algorithm	$\sum$ LB	#	Algorithm	$\sum$ LB	#
Compact	773	24	BD-gIIS-CP	1189	30
BaC-simple-CP	808	30	BD-mIIS-CP	1192	30
BD-aIIS-CP	938	27	BD-gIIS-MIP	1193	30
BD-aIIS-CPMIP-Htime_ud	947	26	BD-gIIS-CPMIP	1193	30
BD-aIIS-CPMIP-Htime_it	952	26	BD-mIIS-MIP	1200	30
BD-aIIS-CPMIP-Hgaprel_ud	953	26	BD-mIIS-CPMIP	1201	30
BD-aIIS-CPMIP-Hgaprel_it	954	26	BD-mIIS-CPMIP-Hgaprel_it	1209	30
BD-aIIS-CPMIP	972	27	BD-mIIS-CPMIP-Hgaprel_ud	1209	30
BaC-simple-CPMIP	978	30	BaC-gIIS-CP	1212	30
BaC-simple-MIP	983	30	BaC-mIIS-CP	1212	30
BD-aIIS-MIP	1011	29	BD-gIIS-CPMIP-Hgaprel_it	1213	30
BaC-aIIS-MIP	1092	28	BD-gIIS-CPMIP-Hgaprel_ud	1213	30
BaC-aIIS-CPMIP	1096	28	BaC-mIIS-MIP	1221	30
BaC-aIIS-CP	1097	28	BaC-gIIS-MIP	1223	30
BD-simple-CP	1150	30	BaC-gIIS-CPMIP	1224	30
BD-simple-CPMIP-Htime_it	1155	30	BD-gIIS-CPMIP-Htime_it	1224	30
BD-simple-CPMIP-Htime_ud	1155	30	BaC-mIIS-CPMIP	1225	30
BD-simple-MIP	1156	30	BD-gIIS-CPMIP-Htime_ud	1226	30
BD-simple-CPMIP	1157	30	BD-mIIS-CPMIP-Htime_it	1232	30
BD-simple-CPMIP-Hgaprel_it	1170	30	BD-mIIS-CPMIP-Htime_ud	**1233**	30
BD-simple-CPMIP-Hgaprel_ud	1170	30			

Table 4.4: Summary of the total number of served requests ($\sum$ LB) across all tested algorithms for the SDARP instances. Column # denotes the number of instances for which the respective algorithm computed a feasible solution. Algorithms that could not solve all 30 instances terminated prematurely due to the memory limit.

(a) Logic-based Benders decomposition



(b) Branch-and-Check

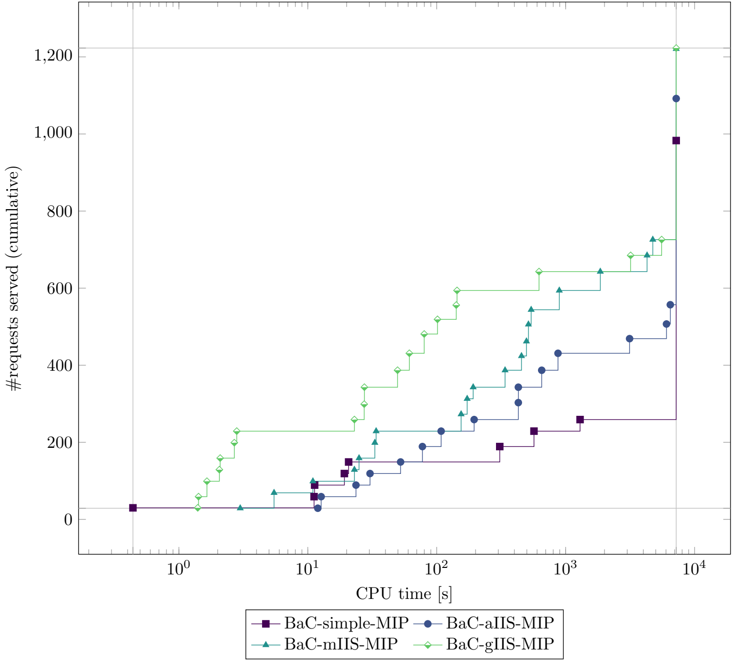


Figure 4.2: Comparison of LBBD and BaC with different types of MILP subproblems in terms of served requests on the SDARP instances. Marks are placed whenever an algorithm terminated, i.e., due to solving an instance to optimality or due the time limit of two hours. Both charts use a logarithmic x-axis.

longer to find results comparable to the other variants. Also, the strategy adding all IISs does not work well. Regarding the final number of served requests, it is even dominated by the simple approach for the LBBD. This is primarily due to several instances hitting the memory limit resulting in missing lower bounds (cf. Table 4.4). The bad performance of strategy aIIS compared to the more successful variants has two reasons: First, it takes a large amount of time to compute all the IISs. Second, the number of added cuts is typically rather large which increases the size of the master problem quite fast. Strategies gIIS and mIIS work much better in this respect. Of these two, the greedy variant exhibits slightly better results since the cuts it provides turned out to perform reasonably well but can be computed much faster than the minimum cardinality IISs. Nevertheless, our results show that the mIIS variant has large potential. Given the computational overhead for computing the minimum cardinality IISs, it is quite impressive that the approach is still competitive. This shows that the stronger cuts provide indeed a considerable improvement on the cuts obtained from the greedy approach. Having identified gIIS and mIIS as superior strategies for computing Benders cuts we focus on those two in the following.

Comparing the decomposition approaches we observe a slight advantage for BaC. It turned out that solutions are found faster by evaluating all integral solution candidates—instead of only optimal ones like LBBD does. For both algorithms finding good or even optimal solutions becomes harder the scarcer the available resources become compared to the demand. This relation can be expected as the number of combinatorial possibilities from which the algorithm needs to find an optimal one increases. In particular, the Benders algorithms are required to exclude a much larger number of infeasible assignments until only feasible options remain. Relaxations that bound the tour size (see Section 4.3.2) help to reduce this effect.

In addition to solving the subproblems via an MILP solver we also investigated the CP approach from Section 4.2.3. In general, computation times are superior but for some instances severe outliers occurred, including situations in which single subproblems required more time than half an hour. Some of these difficult subproblems appear at the very beginning of the solution process due to unbalanced assignments. However, they also continue to occur later on for request subsets with cardinalities similar to those in optimal solutions. To still profit from the mostly faster CP variant we further investigated a combination working as follows. We start with the CP solver using a time limit of half a second. If no result is obtained, we apply the MILP approach which is in general slightly slower but much more consistent featuring no practically noticeable outliers. Of course, we do not want to waste the work done by the CP solver. Therefore, we build the MILP model using the variable domains of the CP model. Thus, we can take advantage of the outcome of constraint propagation at the root node of CP search, which possibly yields a smaller model. The results for the combined CP-MILP subproblems are shown in Figure 4.3.

It can be seen that adding CP for solving the subproblems helps to find solutions faster than when using the pure MILP approach. In general the relation among the decomposition approaches and the cut generation techniques stays the same. However, the mIIS algorithms profit more since they spend more time on solving subproblems, see also Figure 4.4. As the addition of CP provides a clear improvement we selected the combined variant as subproblem algorithm for the remaining experiments.

Evaluation of the Heuristic Boosting Techniques

In the following we compare the heuristically boosted LBBD algorithms to their basic counterparts. We consider the adaptive boosting in the pure iterative variant (it) and also the variant with adjustments in both directions (ud). As criteria for early termination we use a set of thresholds with respect to the relative optimality gap and a set of time limits, see Section 4.3.4. For the time limit variant we consider time limits of 5, 10, and 30 seconds in ascending order. As final value we use the total remaining time according to the overall time limit. The gap variant uses relative optimality gaps of 0.1, 0.05, 0.025, and 0 in descending order.

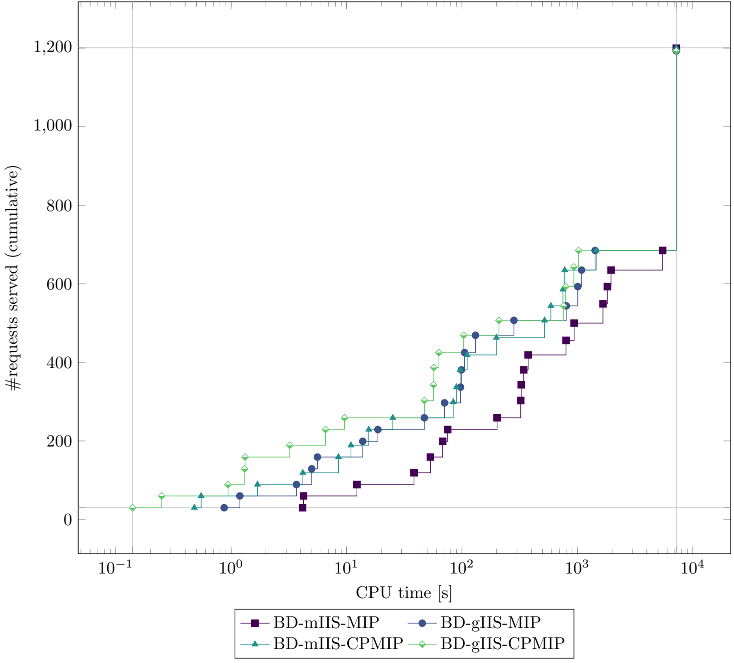
Table 4.5 shows the comparison for the heuristic boosting techniques. With both boosting techniques our algorithms could solve additional instances to optimality and also serve more requests in total. In general, we can observe that the time limit boosting technique works better than the gap boosting. It is mostly faster and also serves more requests overall. Compared to the un-boosted algorithms the time limit boosting is always at least as good in terms of the number of served requests. Except for one instance it provides improvements in all cases where the basic algorithm does not prove optimality. The highest improvements could be achieved by the up-and-down variants for which 33 (gIIS) and 32 (mIIS) additional requests could be served in total. In several cases also the computation times decreased, however, if already the basic variant works well, we sometimes observe a slowdown. To some extent this is related to the potentially worse cuts. The other reason are the required re-solves for proving optimality, which are not needed for the un-boosted algorithms. The gap boosting approaches feature a few outliers at which they serve fewer requests than the reference algorithm. Since the boosting is only a heuristic technique, such outliers are not unexpected: In certain cases it pays off to solve the master problem to optimality to obtain better cuts. However, considering the number of served requests in total, we still observe a reasonable improvement for the gap boosting technique.

In addition to the adaptive approaches presented above, we conducted preliminary tests using a single value as gap threshold or time limit. However, these variants turned out to be much less robust. For some instances they work exceptionally well but this is paid for exceedingly on the remaining ones.

Discussion of the Algorithm Properties

In Figure 4.4 we illustrate the amount of time spent in the master problem, the subproblems, and the repair algorithm for the different decomposition approaches. The first thing to note is that the time spent in the subproblems decreases significantly when switching

(a) Logic-based Benders decomposition



(b) Branch-and-Check

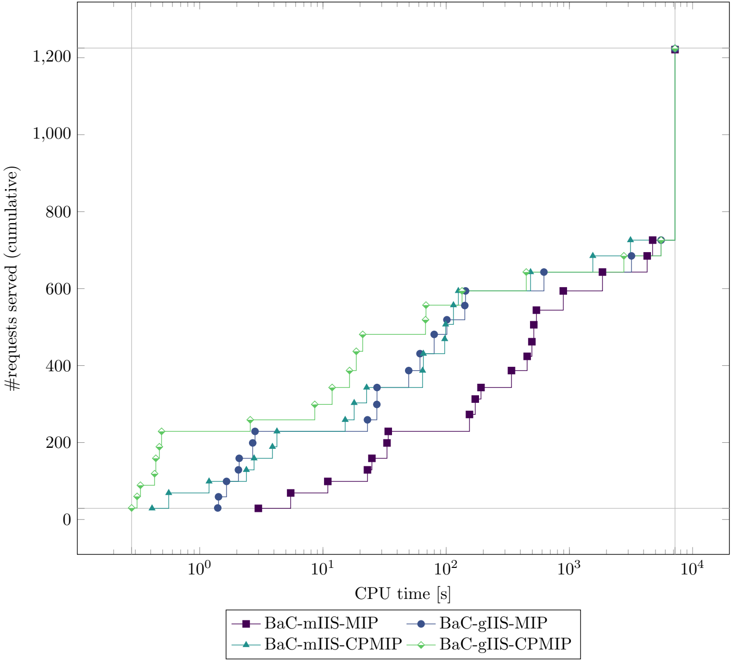


Figure 4.3: Comparison of LBBD and BaC with different types of MILP and combined CP-MILP subproblems in terms of served requests on the SDARP instances. Marks are placed whenever an algorithm terminated, i.e., due to solving an instance to optimality or due the time limit of two hours. Both charts use a logarithmic x-axis.

		ΔLB								Δcomputation time [s]							
		mIIS				gIIS				mIIS				gIIS			
		gap		time		gap		time		gap		time		gap		time	
	LB*	it	ud	it	ud	it	ud	it	ud	it	ud	it	ud	it	ud	it	ud
30N_4K_A	**30**	-	-	-	-	-	-	-	-	**-3**	**-3**	-	-	-	-	-	-
30N_4K_B	**29**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
30N_4K_C	**30**	-	-	-	-	-	-	-	-	11	17	-	-	9	12	-	-
30N_5K_A	**30**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
30N_5K_B	**30**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
30N_5K_C	**30**	-	-	-	-	-	-	-	-	**-3**	3	-	-	3	1	-	-
40N_4K_A	**38**	-	-	-	-	-	-	-	-	34	122	**2**	**2**	18	44	6	**2**
40N_4K_B	**38**	-	-	-	-	-	-	-	-	18	18	1	2	**-139**	-131	-64	-75
40N_4K_C	**37**	-	-	-	-	-	-	-	-	-379	-341	-237	-237	**-587**	-495	-89	-301
40N_5K_A	**40**	-	-	-	-	-	-	-	-	17	17	-	-	-	-	-	-
40N_5K_B	**40**	-	-	-	-	-	-	-	-	28	228	4	3	26	242	**1**	2
40N_5K_C	**40**	-	-	-	-	-	-	-	-	**-1**	**-1**	-	-	**-1**	**-1**	-	-
44N_4K_A	40	1	1	1	1	**2**	**2**	**2**	**2**	-	-	-	-	-	-	-	-
44N_4K_B	**42**	-	-	-	-	-	-	-	-	923	2099	504	452	**-95**	1075	199	639
44N_4K_C	**41**	-	1	1	1	1	**2**	**2**	**2**	-	-2772	-2761	-666	-	-4261	**-4479**	-3640
44N_5K_A	**44**	-	-	-	-	-	-	-	-	169	897	**-2**	1	124	431	1	3
44N_5K_B	**44**	-	-	-	-	-	-1	-	-	**-151**	-59	-2	2	-2	208	1	2
44N_5K_C	**44**	-	-	-	-	-	-	-	-	190	1136	**-4**	6	121	1297	1	2
50N_4K_A	41	1	1	**3**	**3**	-	-	1	2	-	-	-	-	-	-	-	-
50N_4K_B	43	1	1	**4**	**4**	1	1	3	3	-	-	-	-	-	-	-	-
50N_4K_C	44	1	1	3	3	3	3	**4**	**4**	-	-	-	-	-	-	-	-
50N_5K_A	48	**3**	2	2	**3**	1	2	1	1	-	-	-	-	-	-	-	-
50N_5K_B	**49**	-	-	-	-	-	-	-	-	95	2007	302	104	**86**	1442	127	345
50N_5K_C	**50**	-	-	-	-	-	-1	-	-	212	135	**-492**	-471	-240	6267	517	-288
60N_4K_A	44	1	1	**5**	**5**	3	3	4	4	-	-	-	-	-	-	-	-
60N_4K_B	45	-2	-2	1	1	**3**	**3**	**3**	**3**	-	-	-	-	-	-	-	-
60N_4K_C	44	-	-	**3**	**3**	-1	-1	2	2	-	-	-	-	-	-	-	-
60N_5K_A	56	-1	-1	-	-	-	-	-	**1**	-	-	-	-	-	-	-	-
60N_5K_B	50	2	2	**4**	**4**	3	3	**4**	**4**	-	-	-	-	-	-	-	-
60N_5K_C	53	1	1	4	4	4	4	**5**	**5**	-	-	-	-	-	-	-	-
Total		8	8	31	32	20	20	31	**33**	1161	3504	-2685	-799	-677	6132	**-3779**	-3310

Table 4.5: Results of the heuristic boosting techniques. Column LB* denotes the best lower bound, provably optimal values marked bold. Columns ΔLB and "Δcolumn computation time" report the difference of the lower bounds and computation times, respectively, to the un-boosted algorithm variants. For the lower bound, positive values indicate that additional requests could be served and negative values indicate the contrary. Negative values for the computation times indicate a speedup and positive values a slowdown. Cells that contain "-" indicate that the respective value did not change. The largest improvements per column and instance are marked bold.

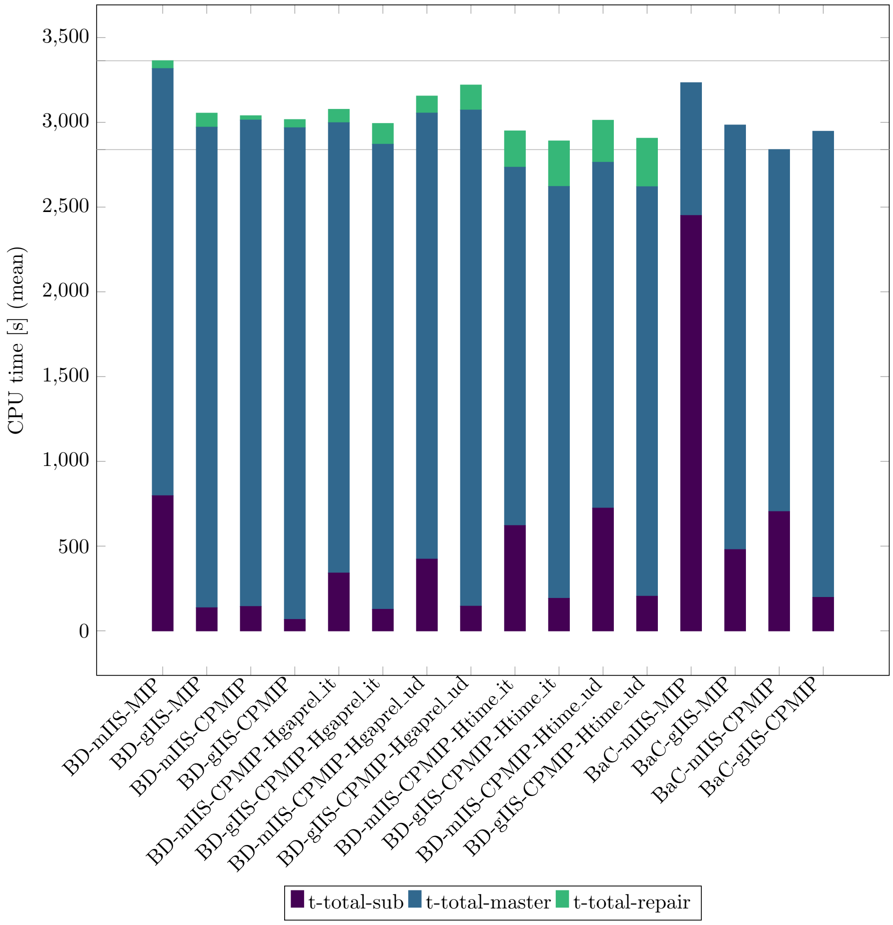


Figure 4.4: Average computation time spent in the master problem, the subproblem, and the repair algorithm for different LBBD and BaC approaches on the SDARP instances.

from the pure MILP algorithm to the combination with CP. This is most notable for the mIIS algorithms where the reduction is the largest due to the high number of solved subproblems.

The idea of the heuristic boosting techniques is to reduce the time spent in the master problem. Most of this time is really saved. However, part of it also shifts into the subproblems or the repair algorithm. In particular for the time limit boosting we observe a significant increase regarding the time spent for repairing solutions. The advantage of the time limit boosting is that it consistently reduces the time spent per master iteration. However, for more challenging master iterations this might lead to worse solutions that leave more work for the repair algorithm.

Figure 4.5 provides details on the gaps with respect to the best known bounds. We compute lower bound gaps by $100 \cdot (\mathrm{LB}^* - \mathrm{LB})/\mathrm{LB}^*$ and upper bound gaps by $100 \cdot$

$(\text{UB} - \text{UB}^*)/\text{UB}^*$ where LB and UB are the lower and upper bound obtained by the considered algorithm and LB^* and UB^* are the respective best bounds known.

Observe that the heuristically boosted LBBD as well as the BaC algorithms perform particularly well with respect to the lower bound gaps. However, they mostly do not perform so well when it comes to finding good upper bounds. In general, the heuristically boosted LBBD provides the better balance, featuring an acceptable performance for both parts. The un-boosted LBBD works not so well for the lower bounds but in general provides the best upper bounds. According to the design of the algorithms this is exactly what one would expect. BaC as well as the heuristically boosted LBBD both derive cuts from potentially suboptimal master solutions. On the one hand, this helps to reduce the time spent for solving the master problem and to derive feasible solutions earlier. On the other hand, this typically slows down progress with respect to the upper bound. In contrast, the un-boosted LBBD solves the master problem always to optimality which helps to find tight upper bounds and strong Benders cuts while taking longer to find good feasible solutions—even with the repair heuristic.

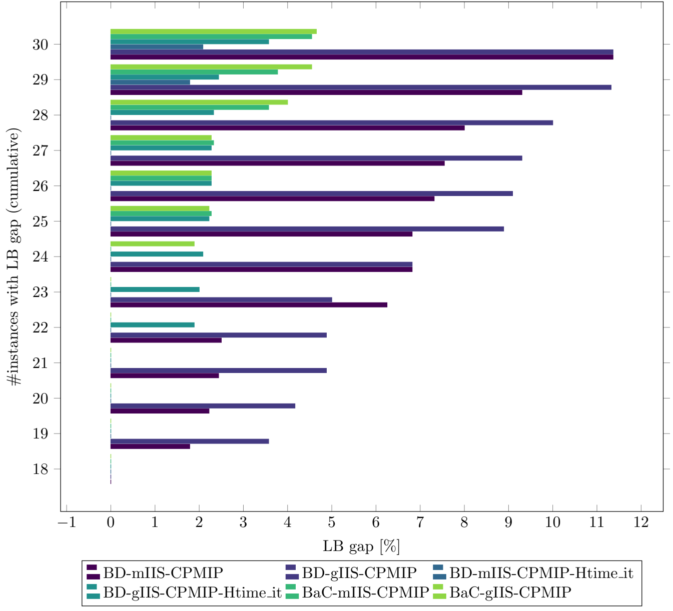
Comparison to the Literature

In the following we test our algorithms on the instances by Ropke et al. [152] including the modified variants by Berbeglia et al. [15] to establish a connection to the existing literature. Different from the SDARP instances, the instances by Ropke et al. [152] feature a significantly larger time horizon relative to the number of available requests. The unmodified instances are guaranteed to be feasible, i.e., all requests can be served. For most of the modified instances it is also possible to serve all requests. In case not all requests can be served, only few have to be rejected. This means that the master problem is much easier to solve than for the SDARP instances. We illustrate this behavior in Figure 4.6. To improve readability we omitted the mIIS variants of BaC with quite excessive computation times of 218 (CP-MILP subproblems) and 793 (MILP subproblems) seconds on average since their relative time distribution is similar to the BaC algorithms included in the figure.

Observe that the Benders algorithms spend almost no time on solving the master problem, little time in the repair routine, and most of the time in the subproblems. The behavior on the modified instances is quite similar, except that the overall computation times increase and that a little more time is spent in the repair routine. Due to this distribution the heuristic boosting techniques do not have a noticeable effect. Moreover, solving the subproblems is more challenging since they typically involve a higher number of requests. This leads to advantages for the gIIS approach that still provides a reasonable guidance for the master problem but solves considerably fewer subproblems than the mIIS variant. Therefore, we focus on the gIIS algorithms without heuristic boosting for the upcoming comparison.

Figures 4.7 and 4.8 provide an overview regarding the results of the obtained lower bounds and computation times. The investigated algorithms solve all instances to optimality.

(a) LB gap



(b) UB gap

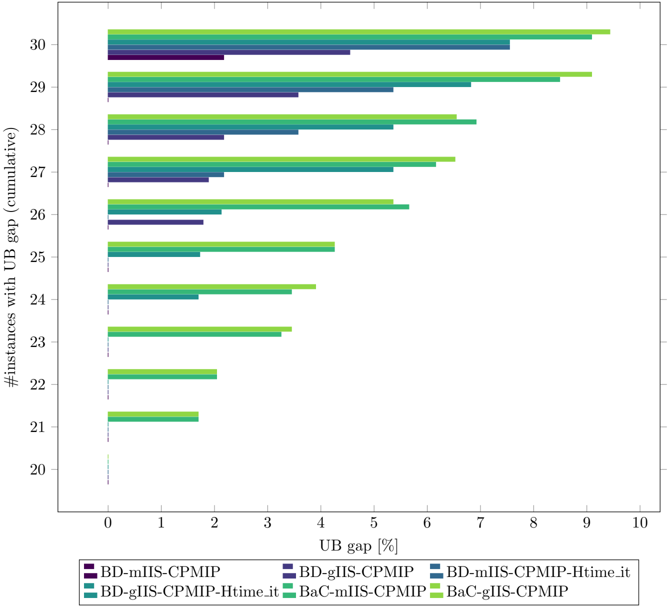


Figure 4.5: Lower and upper bound gaps for different LBBD and BaC algorithms on the SDARP instances. For each algorithm the length of the bar at coordinate y_i corresponds to the largest gap among the y_i instances with the smallest gaps.

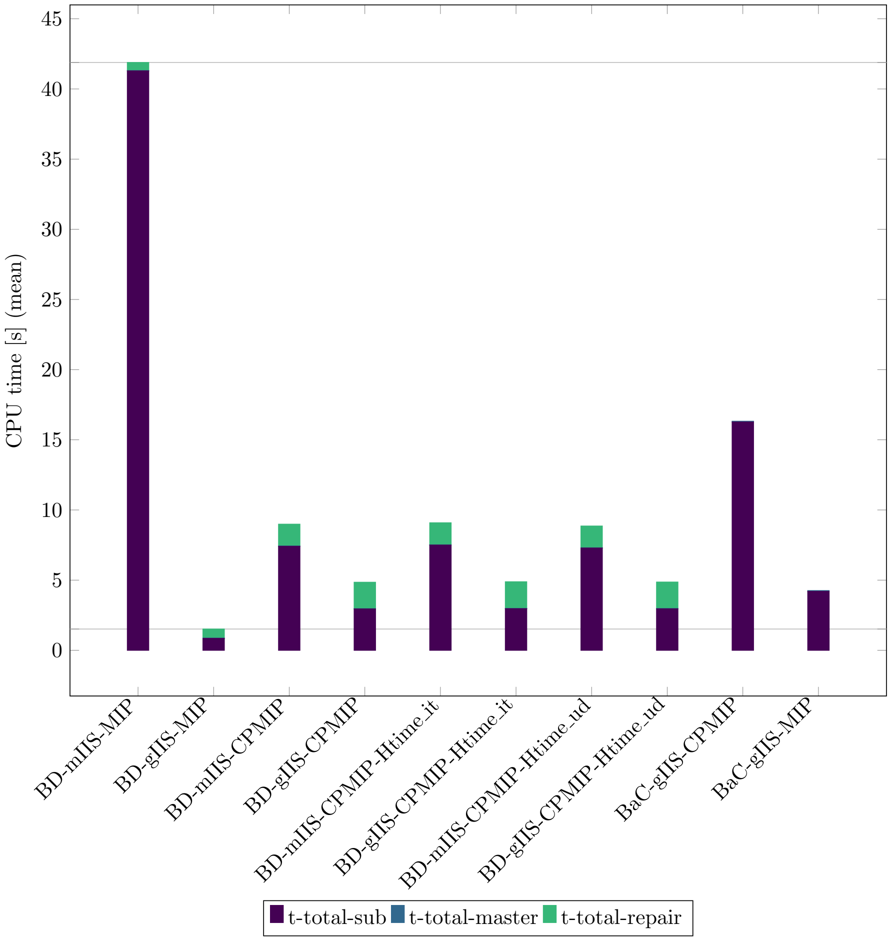


Figure 4.6: Average computation time spent in the master problem, the subproblem, and the repair algorithm for different LBBD and BaC algorithms on the instances by Ropke et al. [152].

The performance with respect to the original instances and those with restricted ride time is quite similar, which is not unexpected because half of the instances (group "a") is not affected by the modification. In contrast, the instances with a reduced number of vehicles are significantly more challenging due to the higher number of requests that have to be rejected. The success of LBBD on the unmodified and the $L = 30$ instances has two reasons. First, the master problem is much easier to solve than for the SDARP instances. Therefore, it is more affordable to always solve it to optimality allowing the LBBD algorithm to converge faster. Second, the repair heuristic turned out to be particularly successful: For the unmodified instances it is often possible to prove optimality after the repair operation in the very first iteration, i.e., without adding any Benders cuts and resolving the master problem. For the instances with only 75% of the original vehicles this is usually not possible which makes BaC the superior algorithm here.

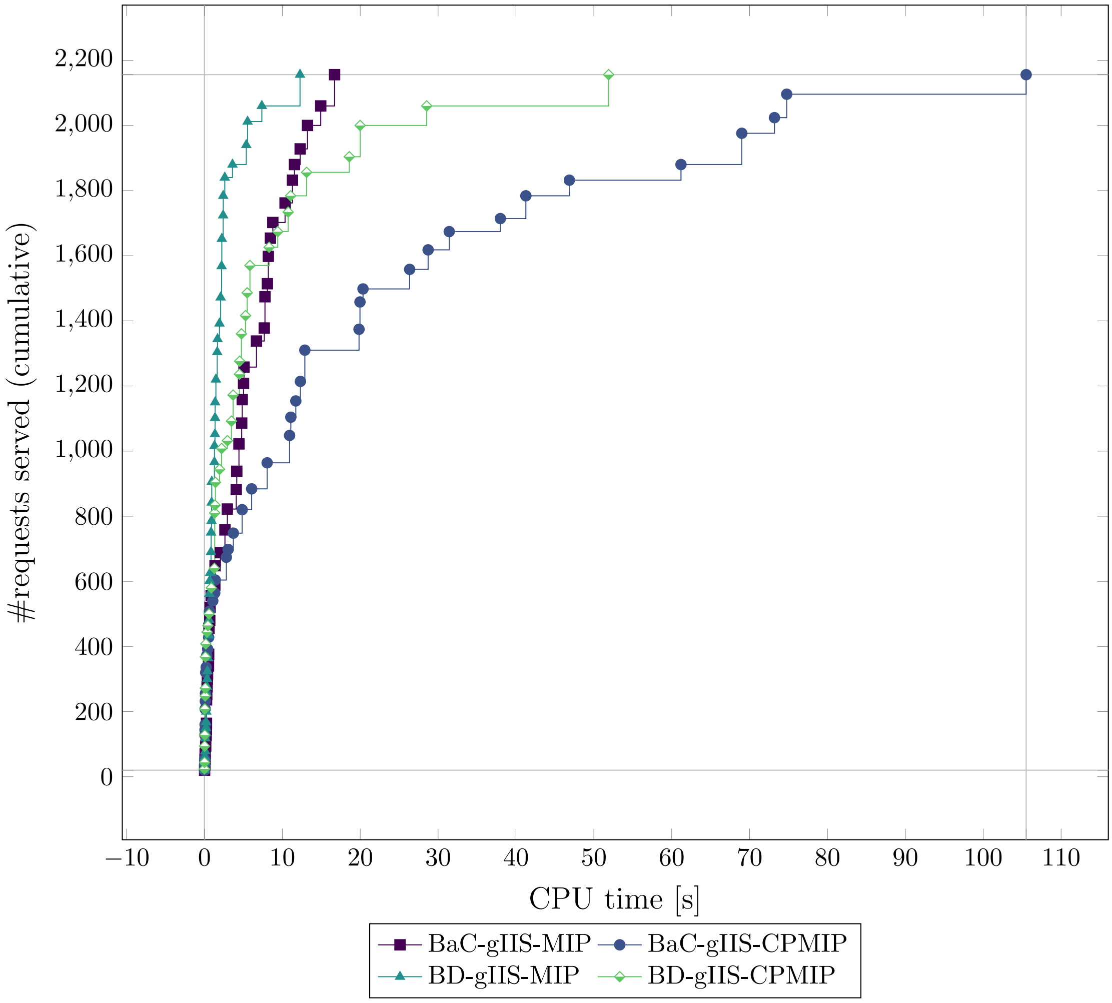
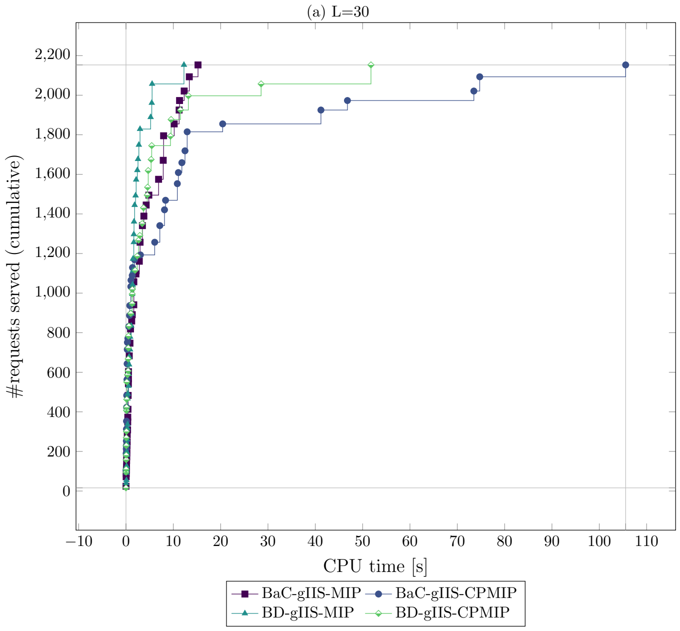


Figure 4.7: Comparison of LBBD and BaC with different types of MILP and combined CP-MILP subproblems in terms of served requests on the instances by Ropke et al. [152]. Marks are placed whenever an algorithm terminated, i.e., due to solving an instance to optimality.

(a) L=30



(b) 75% vehicles

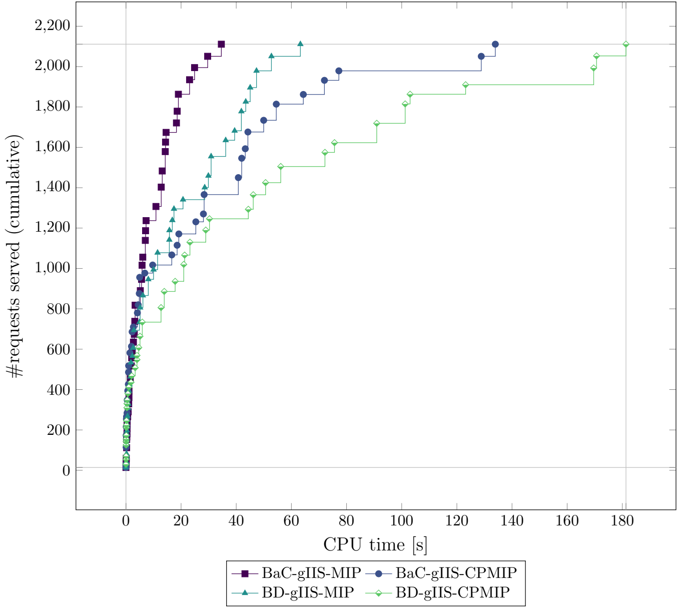


Figure 4.8: Comparison of LBBD and BaC with different types of MILP and combined CP-MILP subproblems in terms of served requests on the instances by Berbeglia et al. [15]. Marks are placed whenever an algorithm terminated, i.e., due to solving an instance to optimality.

Instance	Opt.
b2-16_L30	15
b2-20_L30	19
b3-24_L30	23

Table 4.6: Summary of the number of served requests for the instances by Ropke et al. [152] with $L = 30$. We only list the instances for which not all requests can be served. Column "opt." provides the maximal number of requests that can be served.

Another interesting observation is that using pure MILP subproblems turned out to work better here when using the greedy approach for computing Benders cuts. The reason is that the subproblems are typically less constrained than for the SDARP instances. Moreover, the initial subproblems tend to be larger due to the higher number of requests. However, the mIIS algorithm in general works better with the combined CP-MILP approach due to the bottom-up construction of the IISs, which involves solving many small subproblems.

In Tables 4.6 and 4.7 we provide the results for the modified instances by Berbeglia et al. [15], restricted to those cases in which not all requests can be served. The feasibility checking algorithm by Häme and Hakula [86] also provides a partial solution in case not all requests can be served. We compare our results to theirs showing that further requests can be served in an optimal solution for some instances. Considering the pure feasibility checking task, our algorithms are not as fast in terms of computation times. The maximum cluster algorithm by Häme and Hakula [86] computes the feasibility status in less than a second for most instances, except for a few outliers for the modification with the reduced fleet size taking up to 47 seconds. Our algorithms are able to solve all instances in up to 35 seconds (BaC-gIIS-MIP). Given that our main goal is to determine the maximum number of requests that can be served—the feasibility status is only obtained as a side result—the computational performance appears to be reasonable.

Comparison to the Compact MILP Model

In the following we compare our decomposition approaches to the compact MILP model provided in Section 4.2.2. Since ease of implementation is often a major concern, we selected algorithm variants as competitors that are most comparable in this respect. Therefore, we choose BaC with unrefined Benders cuts and Benders cuts computed by the greedy approach. Note that, using, e.g., CPLEX, BaC can be implemented in terms of a model with a lazy constraint-callback that solves a compact MILP model as subproblem. The subproblem is solved exactly once for the naïve approach and multiple times according to Algorithm 4.1 when using the greedy strategy.

Figures 4.9 and 4.10 compare the BaC algorithms and the compact MILP model. Already the naïve strategy for generating Benders cuts turns out to perform better in terms of optimally solved instances. However, considering the finally obtained lower bounds it is

Instance	Opt.	Prev. best	Instance	Opt.	Prev. best
a2-16_K75	**12**	-	b4-24_K75	**22**	-
a2-20_K75	**18**	-	b4-40_K75	**39**	38
a2-24_K75	**21**	-	b4-48_K75	**47**	**47**
a3-36_K75	**32**	-	b5-40_K75	**39**	**39**
b2-16_K75	**12**	-	b5-50_K75	**47**	44
b2-20_K75	**14**	-	b5-60_K75	**58**	56
b2-24_K75	**20**	-	b6-48_K75	**46**	**46**
b3-24_K75	**23**	-	b7-84_K75	**83**	82
b3-36_K75	**32**	-			

Table 4.7: Summary of the number of served requests for the instances by Ropke et al. [152] with only 75% of the original vehicles. We only list the instances for which not all requests can be served. Column "opt." provides the maximal number of requests that can be served according to our experiments. Column "prev. best" reports the results by Häme and Hakula [86]; instances not considered by them are marked with "-". Provably optimal solution values are marked bold.

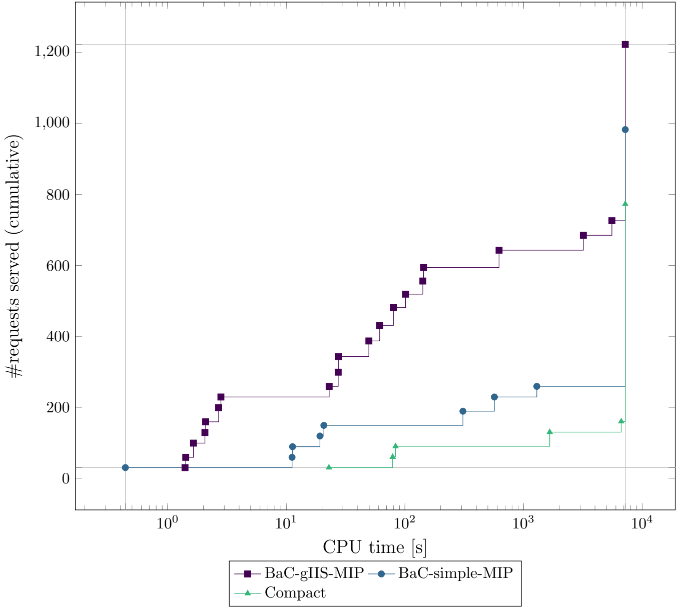
only superior on the SDARP instances. The greedy Benders cut generation approach, on the other hand, dominates the compact model in all aspects. It is not only much faster but its final solutions in general also serve significantly more requests. Across all instances BaC-gIIS-MIP solves 45 more instances to optimality than the compact model: 14 of the SDARP instances, 11 of the original Cordeau instances, 7 of the Cordeau instances with reduced user ride time, and 13 of the Cordeau instances with reduced fleet size.

4.5 Conclusion

In this chapter we considered a variant of the DARP that aims at serving a maximal number of requests rather than minimizing routing costs. We proposed a simple compact reference model and a decomposition approach. The master problem was formulated as MILP model and the subproblems were stated as MILP model and also as CP model. We reviewed preprocessing techniques from the literature and suggested improvements. The master problem of the decomposition approach is supplemented by inequalities representing subproblem relaxations.

In the computational study we solved the decomposition model using LBBD and BaC. Subproblems have been solved using MILP and a combination with CP. The latter hybrid turned out to be most successful. We considered four strategies to construct Benders feasibility cuts. Experiments have shown that a fast greedy approach and the enumeration of all minimum cardinality IISs work best. It is most crucial to base the Benders cuts on IISs to avoid unnecessarily weak cuts. Interestingly, it turned out that the rather time-consuming approach for constructing the minimum cardinality IISs is

(a) SDARP



(b) Ropke et al.

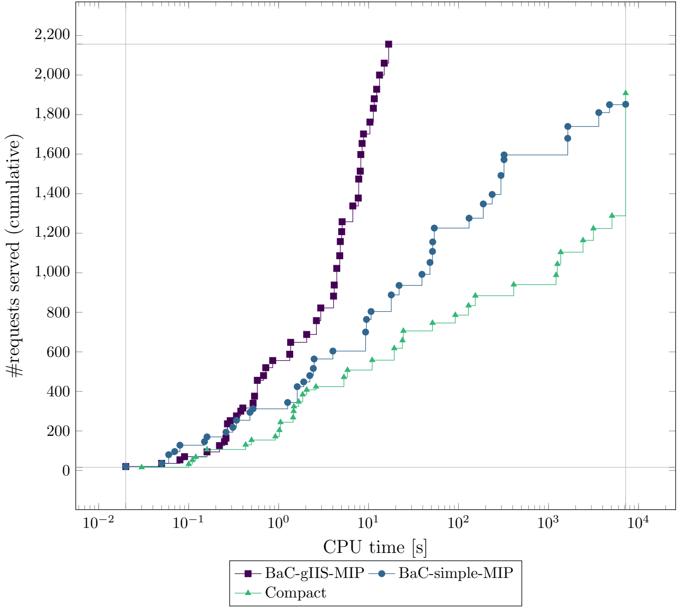
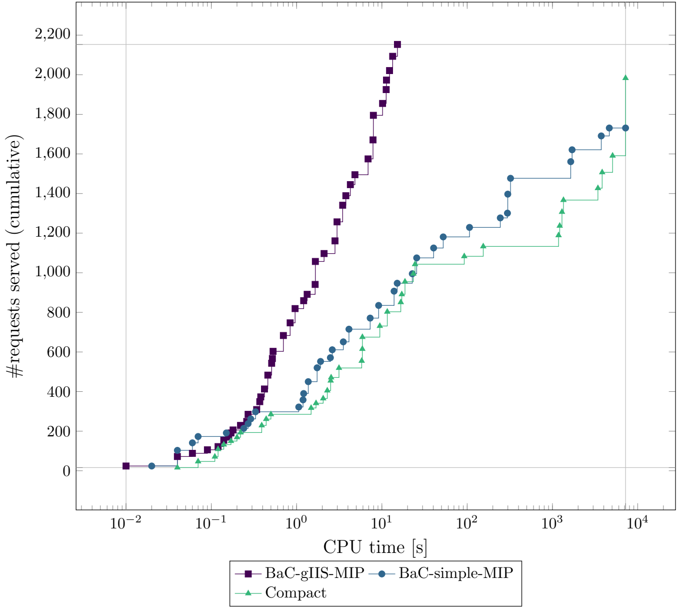


Figure 4.9: Comparison of BaC with the compact reference MILP model in terms of served requests on the SDARP instances and the instances by Ropke et al. [152]. Marks are placed whenever an algorithm terminated, i.e., due to solving an instance to optimality or reaching the time limit. Both charts use a logarithmic x-axis.

(a) Ropke et al. (L=30)



(b) Ropke et al. (75% vehicles)

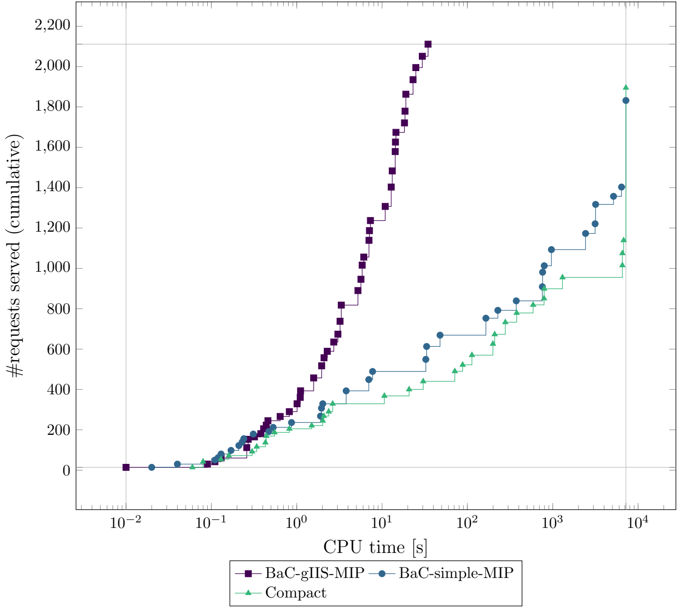


Figure 4.10: Comparison of BaC with the compact reference MILP model in terms of served requests on the instances by Ropke et al. [152] with the modifications by Berbeglia et al. [15]. Marks are placed whenever an algorithm terminated, i.e., due to solving an instance to optimality or reaching the time limit. Both charts use a logarithmic x-axis.

still competitive. This shows that the mIIS approach has significant potential. The Benders cuts obtained this way are non-dominated within the considered class and our experiments show that computing them is worthwhile. We think that using them might also be interesting for other applications of LBBD.

To speed up solving the Benders master problems we considered heuristic boosting techniques: Instead of always solving the master problem to optimality, we stop once a certain time limit or threshold with respect to the relative optimality gap has been reached. In particular the time limit boosting helped to improve the Benders algorithm, making it possible to find better solutions and to reduce computation times. The suggested boosting techniques are conceptually simple and more generally promising also in the context of LBBD approaches for other applications.

Comparing LBBD and BaC we observed that in general the former excels at computing good dual bounds whereas the latter is superior for computing primal bounds. This effect can be reduced by including an algorithm for repairing infeasible master solutions obtained by LBBD. In situations where solving the master problem is time-consuming the boosting techniques may provide a middle way. For the SDARP instances, they slightly decrease progress with respect to the dual bound but provide a significant speedup for finding good primal bounds.

In general, we can draw the conclusion that the decomposition approach works best if the proportion of requests that can be accepted is not too low.

4.5.1 Future work

In practical applications not all requests might be equally important. Thus, a natural extension of the considered DARP variant would be to consider weights for the requests. Due to the focus on request selection we do not consider routing costs in the objective. The easiest extension would be to consider cost-optimal routing for each vehicle separately, keeping the problem complexity more or less the same. However, this may lead to globally suboptimal solutions since selecting different requests might reduce the routing costs while retaining the number of served requests. Considering globally optimal routing costs makes the problem much more challenging since the objectives of the subproblems now influence the master problem and thus also Benders optimality cuts are needed. Moreover, also other second-level objectives might be worth considering like additional user-inconvenience considerations, e.g., limiting the direct route to actual route ratio. Additionally, investigating further strategies and testing with heterogeneous vehicles would be interesting.

In our experiments heuristic boosting techniques turned out to be beneficial for solving the master problem. Applying a similar strategy for the subproblems did not work that well. However, by replacing the basic heuristic we considered with a more sophisticated approach it might be possible to also speed up solving the subproblems.

We considered four strategies for constructing Benders feasibility cuts. Our algorithms are based on enumeration and a greedy approach. In this respect it would be interesting to design problem specific approaches that are able to find structures close to the minimum cardinality IISs requiring less time than enumeration. The work by Häme and Hakula [86] could serve as a starting point for research in this direction.

CHAPTER 5

An Iterative Time-Bucket Refinement Algorithm for a High-Resolution Resource-Constrained Project Scheduling Problem

In the previous two chapters we considered decomposition approaches that pursued the principle of disregarding an aspect of the problem and reintroducing the necessary information incrementally. For the column generation approach in Chapter 3 this helped to avoid many variables. In terms of the logic-based Benders decomposition (LBBD) approach from Chapter 4 we managed to defer computational effort to the subproblem and also achieved a structural decomposition leading to problems that could be tackled with specialized techniques. In this chapter we consider a different decomposition strategy. Instead of disregarding part of the problem we design an individual relaxation. This relaxation is then successively refined until it becomes feasible or provides a tight dual bound that can prove optimality of a heuristically obtained solution. We present our approach in terms of a matheuristic that computes converging sequences of primal and dual bounds. The dual bounds are directly obtained from the aforementioned relaxations and give rise to primal bounds that are derived from the relaxed solutions by heuristics.

We designed this algorithm to deal with a specific problem scenario arising from patient scheduling in cancer treatment. This approach was developed in terms of a cooperation with cancer treatment center Medaustron[1], whose support is greatly appreciated. MedAustron provides modern particle therapy via a particularly precious particle beam.

---

[1] https://www.medaustron.at

The goal is to exploit this resource as efficiently as possible to maximize the number of patients who can be treated. Observe that the resulting scheduling scenario is quite complex and involves a rather long planning horizon. Here we concentrate on the operations within a day with a special focus on a fine-grained time discretization. For further details on the full problem we refer to a number of co-authored publications ([122, 123, 124, 125]) that describe the practical context more extensively.

With the proposed matheuristic we aim at overcoming limitations of classical mixed integer linear programming (MILP) techniques such as time-indexed formulations (TIFs) or discrete-event formulations (DEFs) that are known to struggle under the investigated scenario due to the large resulting time horizon. The considered relaxation is based on aggregating time into so-called time-buckets. Depending on the degree of aggregation this significantly reduces the problem size but requires an iterative refinement step to preserve feasibility. Our experiments indicate that the matheuristic performs significantly better than the classical MILP approaches.

This chapter has been accepted for publication in *International Transactions in Operational Research*:

> M. Riedler, T. Jatschka, J. Maschler, and G. R. Raidl. An iterative time-bucket refinement algorithm for a high-resolution resource-constrained project scheduling problem. *International Transactions in Operational Research*, 2017. doi: 10.1111/itor.12445. available online

5.1 Introduction

Scheduling problems arise in a variety of practical applications. Prominent examples are job shop or project scheduling problems that require a set of activities to be scheduled over time. The execution of the activities typically depends on certain resources of limited availability and diverse other restrictions such as precedence constraints. The goal is to find a feasible schedule that minimizes some objective function like the makespan. In certain cases, scheduling has to be done in a very fine grained way, i.e., in high resolution, using, e.g., seconds or even milliseconds as unit of time.

Classical MILP formulations are known to struggle under these conditions. On the one hand, time discretized models provide strong linear programming (LP) bounds but grow too quickly with the instance size due to the fine time discretization. Event-based and sequencing-based models on the other hand typically have trouble as a result of their weak LP bounds.

In the following, we focus on problems with a large, very fine-grained scheduling horizon and consider a simplified scheduling problem arising in the context of modern particle therapy used for cancer treatment. The problem is motivated by a real world patient scheduling scenario at the cancer treatment center MedAustron located in Wiener Neustadt, Austria. The tasks involved in providing a given set of patients with their individual particle treatments shall be scheduled in such a way that given precedence constraints with minimum and maximum time lags are respected. Each task needs certain resources for its execution. One of the resources is the particle beam, which is particularly scarce as it is required by every treatment and shared between several treatment rooms. The motivation therefore is to exploit in particular the availability of the beam as good as possible by suitably scheduling all activities in high time resolution. Ideally, the beam is switched immediately after an irradiation has taken place in one room to another room where the next irradiation session starts without delay.

Our goal is to minimize the makespan. This objective emerges from the practical scenario as tasks need to be executed as densely as possible to avoid idle time within the day as well as to allow treating as many patients as possible within the operating hours. However, makespan minimization is clearly an abstraction from the real world scenario where more specific considerations need to be taken into account (see, e.g., [122]). In the terminology of the scientific literature in scheduling, the considered problem corresponds to a resource-constrained project scheduling problem (RCPSP) with minimum and maximum time lags.

In this work, we introduce the simplified intraday particle therapy patient scheduling problem (SI-PTPSP) and present a DEF and a TIF for it as reference models. We propose a time-bucket relaxation (TBR) and prove some theoretical properties. In the main part, we deal with the iterative time-bucket refinement algorithm (ITBRA) that aims at closing the gap between dual solutions obtained by solving TBR based on iteratively refined bucket partitionings and heuristically determined primal solutions exploiting dual solutions. Various strategies for refining the bucket partitioning are suggested. Experimental results clearly indicate the superiority of the new matheuristic approach over the reference MILP models as well as a basic greedy randomized adaptive search procedure (GRASP).

The remainder of the chapter is organized as follows. In Section 5.2 we provide a formal definition of the SI-PTPSP. Then, we review the related literature. In the following section we provide two reference MILP formulations. The main part consists of the description of TBR and its properties in Section 5.5 and the presentation of ITBRA in Section 5.6. We provide the fundamental iterative framework with its specifically used sub-algorithms, which are the gap closing heuristic (GCH), the activity block construction heuristic (ABCH), a GRASP metaheuristic, and the investigated bucket refinement strategies. Further implementation details such as preprocessing procedures are covered in Section 5.7. Finally, we discuss computational experiments conducted on two sets of benchmark instances in Section 5.8, before concluding and giving an outlook on promising future research directions in Section 5.9.

5.2 Simplified Intraday Particle Therapy Patient Scheduling Problem

The SI-PTPSP is defined on a set of activities $A = \{1, \ldots, \alpha\}$ and a set of unit-capacity resources $R = \{1, \ldots, \rho\}$. Each activity $a \in A$ is associated with a processing time $p_a \in \mathbb{N}_{>0}$, a release time $t_a^{\mathrm{r}} \in \mathbb{N}_{\geq 0}$ and a deadline $t_a^{\mathrm{d}} \in \mathbb{N}_{\geq 0}$ with $t_a^{\mathrm{r}} + p_a \leq t_a^{\mathrm{d}}$. For its execution an activity $a \in A$ requires a subset $Q_a \subseteq R$ of the resources. Activities have to be executed without preemption. The considered set of time slots $T = \{T^{\min}, \ldots, T^{\max}\}$ is derived from the properties of the activities as follows: $T^{\min} = \min_{a \in A} t_a^{\mathrm{r}}$ and $T^{\max} = \max_{a \in A} t_a^{\mathrm{d}} - 1$. We denote by $Y_a(t)$ the set of time points during which activity $a \in A$ executes when starting at time t, i.e., $Y_a(t) = \{t, \ldots, t + p_a - 1\}$. To model dependencies among the activities, we consider a directed acyclic precedence graph $G = (A, P)$ with $P \subset A \times A$. Each arc $(a, a') \in P$ is associated with a minimum and a maximum time lag $L^{\min}_{a,a'}, L^{\max}_{a,a'} \in \mathbb{N}_{\geq 0}$ with $L^{\min}_{a,a'} \leq L^{\max}_{a,a'}$. For each resource $r \in R$ a set of availability time windows $W_r = \bigcup_{w=1,\ldots,\omega_r} W_{r,w}$ with $W_{r,w} = \{W^{\mathrm{start}}_{r,w}, \ldots, W^{\mathrm{end}}_{r,w}\} \subseteq T$ is given. Resource availability windows are non-overlapping and ordered according to starting time $W^{\mathrm{start}}_{r,w}$. In accordance with the resource availabilities and the precedence relations among the activities, we can deduce for each activity a set of feasible starting times, denoted by $T_a \subseteq \{t_a^{\mathrm{r}}, \ldots, t_a^{\mathrm{d}} - p_a\}$; for details on the computation of this set see Section 5.7.1.

A feasible solution S (also called schedule) to SI-PTPSP is a vector of values $S_a \in T_a$ assigning each activity $a \in A$ a starting time within its release time and deadline such that the availabilities of the required resources and all precedence relations are respected. The goal is to find a feasible solution having minimum makespan.

Using the notation introduced in Brucker et al. [29] our problem can be classified as $\mathrm{PS}m, \cdot, 1 | r_j, d_j, \mathit{temp} | C_{\max}$.

Computational complexity. Lawler and Lenstra [106] have shown that finding a solution for the non preemptive single machine scheduling problem with deadlines and release times ($1|r_j|C_{\max}$ according to the notation by Graham et al. [83]) is $\mathcal{NP}$-hard. We can easily reduce an instance of $1|r_j|C_{\max}$ to an instance of SI-PTPSP by assigning the same resource to each activity of the $1|r_j|C_{\max}$ instance. Processing times, release times, and deadlines of the activities remain unchanged. Since there are no precedence constraints in $1|r_j|C_{\max}$, the set of precedence arcs is empty. Consequently, SI-PTPSP is $\mathcal{NP}$-hard.

5.3 Related Work

In this section, we discuss the related work relevant for our contribution. We start with a brief overview of RCPSPs. Afterwards, we review the derivation of dual bounds for such scheduling problems. Then, we give a short introduction on matheuristics applied in this domain. Finally, we review previous work that is important from the methodological point of view, i.e., that deals with time-buckets or similar aggregation techniques.

5.3.1 Resource-Constrained Project Scheduling

The RCPSP considers scheduling of a project subject to resource and precedence constraints, where a project is represented by a graph with each node being an activity of the project. Precedence relations between activities are represented as directed edges between the nodes. The RCPSP is a well-studied problem with many extensions and variations. SI-PTPSP is a combination of multiple such extensions: We use minimum and maximum time lags, release times and deadlines, and dedicated renewable resources. For a detailed description of those terms and a broader overview of RCPSP variants we refer to Hartmann and Briskorn [88].

There exists a wide range of exact and heuristic approaches for the RCPSP and its extensions, for an overview see [4, 29, 131]. Here we specifically want to focus on exact approaches. Branch-and-bound (B&B) algorithms [21, 51] and MILP techniques are often used. However, also constraint programming (CP), SAT, and combinations thereof gained importance, see, e.g., Berthold et al. [16]. For our work we are primarily interested in MILP-based approaches and thus focus on them in the following.

A well-known technique are so-called time-indexed models, see Artigues [2]. The classical variant uses binary variables for each time slot representing the start of an activity. In addition, there are also so-called step-based formulations, in which variables indicate whether an activity has started at or before a certain time instant. This might lead to a more balanced B&B tree. Both variants typically provide strong LP bounds but struggle with larger time horizons due to the related model growth.

Also quite well-known are event-based formulations. Koné et al. [101] and Artigues et al. [5] provide an extensive overview. These models are based on a set of ordered events to which activity starts and ends need to be assigned, which makes it possible to model starting times as continuous variables. On/Off event-based formulations use the same idea but require even fewer variables. These models are usually independent of any time discretization and the time horizon but feature significantly weaker LP bounds compared to time-indexed models.

There also exist formulations combining continuous-time and discrete-time formulations, so-called mixed-time models, see [13, 169]. Further MILP techniques make use of exponentially sized models and apply advanced techniques such as column generation, Lagrangian decomposition, or Benders decomposition, see, e.g., [90].

5.3.2 Dual Bounds for Scheduling Problems

The most common method for deriving lower bounds is based on solving LP relaxations, often strengthened by cutting plane methods. This approach is widely applicable but often provides only weak bounds.

Also rather well-known are algorithms based on Lagrangian relaxation, see Fisher [62]. The basic idea is to relax a set of complex constraints by adding corresponding penalty terms to the objective function to simplify the model. Its strong reliance on a suitable problem structure limits the applicability of this technique. For an application to the RCPSP see Bianco and Caramia [20].

Other techniques to obtain dual bounds are less common. Li et al. [112] consider a dual heuristic for MILP. For some nodes of the B&B tree, the heuristic attempts to improve the current dual bound by computing relaxations based on simply dropping, dualizing, or aggregating constraints. The heuristic uses dual variables and slack variables of the LP solution in order to decide which constraints to relax.

Apart from such general approaches, there are some works that consider problem specific methods. In the RCPSP context this includes, among others, Bianco and Caramia [19], Carlier et al. [33], and Dupin and Talbi [57].

5.3.3 Matheuristics for Scheduling Problems

So far, Matheuristics have only been rarely considered for tackling the RCPSP. Palpant et al. [132] present an approach based on large neighborhood search. Subproblems are generated dynamically and solved using MILP, CP, or a heuristic approach.

Further matheuristic approaches can be found in terms of the multi-mode resource-constrained multi-project scheduling problem (MRCMPSP). This is an extension of the RCPSP in which each activity is associated with a set of modes that decide the processing time and resource demand. Artigues and Hebrard [3] solve the MRCMPSP with an algorithm consisting of four phases. In the first phase initial modes are assigned to each activity using MILP. Phases 2 and 3 generate a schedule based on the modes assigned to the activities using CP. The last phase uses a large neighborhood search procedure to improve the schedule by changing the modes of some activities. CP is used to solve the subproblems. Phases 2 to 4 are repeated until a termination criterion is met. Toffolo et al. [162] solve the problem using a decomposition-based matheuristic. After fixing execution modes, the problem is decomposed into time windows that are solved using MILP models. Finally, a hybrid local search is employed to improve the obtained solutions.

Moreover, note that there are resemblances to Benders and Lagrangian-based techniques, e.g., Maniezzo and Mingozzi [120], Möhring et al. [128].

5.3.4 Time Aggregation Models

Note that the contributions mentioned in this section stay in contrast to a more common approach in which the time-discretization is coarsened in order to possibly obtain feasible but also less precise solutions, which are in general not optimal for the original problem. The approaches discussed here are characterized by iteratively refining a relaxation of the original problem until a provably optimal solution is found.

Boland et al. [27] consider such an approach for the countinuous time service network design problem (CTSNDP). The authors solve the problem using a time-expanded network. Initially, only a partially time-expanded network is considered to avoid the substantial size of the complete network. The MILP model associated with the reduced network constitutes a relaxation to the original problem. If the optimal solution to this relaxation turns out to be feasible with respect to the original problem, the algorithm terminates. Otherwise, the partially time-expanded network is extended based on the current solution to obtain a more refined model. Iteratively applying this approach converges to an optimal solution due to the finite size of the full time-expanded network.

A different type of relaxation is to partition the given time horizon into subsets. Approaches of this type are considered in Bigras et al. [22], Baptiste and Sadykov [10], and Boland et al. [25] for single machine scheduling problems. Iterative approaches based on these techniques have been primarily considered in terms of routing problems. Wang and Regan [167, 168] consider such an algorithm for the traveling salesman problem with time windows (TSPTW). First, the time windows of each node are partitioned into subsets. Then, for a given time window partitioning a lower bound and an upper bound are calculated, using an underconstrained MILP model and an overconstrained one. If the gap between lower and upper bound is not sufficiently small, the scheduling horizon gets further refined and the problem is solved anew.

Another algorithm of this type has been considered by Macedo et al. [118] for solving the vehicle routing problem with time windows and multiple routes (MVRPTW). They solve a relaxation which is modeled as a network flow such that nodes of the graph correspond to time instants. The idea of the initial relaxation is to aggregate several time instants into each node. If the solution to the relaxation turns out to be infeasible with respect to the original problem, the current time discretization is locally refined by considering further time instants individually, i.e., by disaggregating nodes.

Dash et al. [48] combine the ideas of Wang and Regan [167] and Bigras et al. [22] in order to solve the TSPTW. The time windows of the nodes are partitioned into buckets using an iterative refinement heuristic. Refinement decisions are based on the solution to the current LP relaxation. Afterwards, the resulting formulation is turned into an exact approach by adding valid inequalities and solved using branch-and-cut (B&C). In each node of the B&B tree a primal heuristic is applied using the reduced costs of the variables of the current LP relaxation.

Recently, Clautiaux et al. [39] introduced an approach that is more generally applicable to problems that can be modeled as minimum-cost circulation problems with linking bound constraints. The proposed algorithm projects the original problem onto an aggregated approximate one. This aggregated model is iteratively refined until a provably optimal solution is found. Experiments have been conducted on a routing problem and a cutting-stock problem.

5.4 Reference MILP Models

In this section, we present two MILP models for SI-PTPSP following classical approaches: a DEF and a TIF. Both serve as reference formulations to which we will compare our ITBRA.

5.4.1 Discrete Event Formulation

The DEF is based on the idea of considering certain events that need to be ordered and for which respective times need to be found, see also model SEE in Artigues et al. [5]. Resource constraints then only have to be checked at the times associated with these events.

In regard to our problem, the considered events are the start and the end of each activity (activity events), and times at which the availability of a resource changes (resource events). To simplify the model, we transform all resource events into activity events by introducing a new artificial activity for each period during which a resource $r \in R$ is unavailable. To this end, we create a new activity for each maximal interval in $T \setminus W_r$ requiring the resource. Start, end, and processing time of this activity are set to match the unavailability period. Accordingly, we define a new set of activities A' being the union of A and the artificial activities; let $\alpha' = |A'|$. Consequently, we denote by $K = \{1, \dots, 2\alpha'\}$ the set of chronologically ordered events.

To state the model we use binary variables $x_{a,k}$ that are one if event $k \in K$ is the start of activity $a \in A$ and zero otherwise. Similarly, binary variables $y_{a,k}$ indicate whether event k is the end of activity a. Variables E_k represent the time assigned to each event k. The starting times of the activities $a \in A'$ are modeled using variables S_a. Finally, binary variables $D_{r,k}$ are one if resource $r \in R$ is used by any activity immediately after event k and zero otherwise, and variable MS denotes the makespan. The model reads as follows:

$$
\begin{aligned}
\min \quad & MS && && (5.1)\\
\text{subject to} \quad & S_a + p_a \le MS && \forall a \in A && (5.2)\\
& S_{a'} - S_a \ge p_a + L^{\min}_{a,a'} && \forall (a,a') \in P, && (5.3)\\
& S_{a'} - S_a \le p_a + L^{\max}_{a,a'} && \forall (a,a') \in P, && (5.4)\\
& \sum_{k \in K} x_{a,k} = 1 && \forall a \in A', && (5.5)\\
& \sum_{k \in K} y_{a,k} = 1 && \forall a \in A', && (5.6)\\
& \sum_{a \in A'} (x_{a,k} + y_{a,k}) = 1 && \forall k \in K && (5.7)\\
& E_{k-1} \le E_k && \forall k \in K \setminus \{1\}, && (5.8)\\
& E_k - M^{(5.9)}_{a,k}(1 - x_{a,k}) \le S_a && \forall k \in K, a \in A', && (5.9)\\
& E_k + M^{(5.10)}_{a,k}(1 - x_{a,k}) \ge S_a && \forall k \in K, a \in A', && (5.10)\\
& E_k - M^{(5.11)}_{a,k}(1 - y_{a,k}) \le S_a + p_a && \forall k \in K, a \in A', && (5.11)\\
& E_k + M^{(5.12)}_{a,k}(1 - y_{a,k}) \ge S_a + p_a && \forall k \in K, a \in A', && (5.12)\\
& D_{r,0} = \sum_{a \in A' : r \in Q_a} x_{a,0} && \forall r \in R, && (5.13)
\end{aligned}
$$

$$D_{r,k} = D_{r,k-1} + \sum_{a \in A': r \in Q_a} x_{a,k} - \sum_{a \in A': r \in Q_a} y_{a,k} \quad \forall k \in K \setminus \{1\}, r \in R, \tag{5.14}$$

$$D_{r,k} \leq 1 \quad \forall k \in K, r \in R, \tag{5.15}$$

$$t^{\mathrm{r}}_a \leq S_a \leq t^{\mathrm{d}}_a - p_a \quad \forall a \in A', \tag{5.16}$$

$$MS, E_k, D_{r,k} \geq 0 \quad \forall k \in K, a \in A', r \in R, \tag{5.17}$$

$$x_{a,k}, y_{a,k} \in \{0,1\} \quad \forall k \in K, a \in A'. \tag{5.18}$$

Inequalities (5.2) are used for determining the makespan. Precedence relations are enforced by inequalities (5.3) and (5.4). According to equalities (5.5) and (5.6) each activity starts and ends at precisely one event. Equalities (5.7) ensure that each event is assigned to either exactly one starting time or exactly one ending time of an activity. Events are ordered chronologically by inequalities (5.8). Starting times of activities are linked to the corresponding start events by inequalities (5.9) and (5.10). Similarly, inequalities (5.11) and (5.12) link the event at which an activity a ends to the time at which the activity ends. Big-M constants used in these inequalities will be explained below. Equalities (5.13) and (5.14) compute the total demand of a resource of all activities running during an event. Finally, inequalities (5.15) ensure that all resource demands are met at all events.

Choosing the smallest possible big-M constants for Inequalities (5.9)–(5.12) in DEF is important for making its LP relaxation as tight as possible. An easy way to set them is $M^{(5.9)}_{a,k} = T^{\max} - t^{\mathrm{r}}_a$, $M^{(5.10)}_{a,k} = t^{\mathrm{d}}_a - p_a - T^{\min}$, $M^{(5.11)}_{a,k} = T^{\max} - t^{\mathrm{r}}_a - p_a$, and $M^{(5.12)}_{a,k} = t^{\mathrm{d}}_a - T^{\min}$. However by computing sets of activities that must precede or succeed a certain event in any feasible schedule, respectively, it is possible to fix some of the constants to zero. For details, we refer to [94].

The formulation has $O(|A'|^2)$ variables and $O(|R| \cdot |A'|^2)$ constraints. Thus, DEF is a compact model, but its LP relaxation typically yields rather weak LP bounds, primarily due to the inequalities involving the big-M constants.

5.4.2 Time-indexed Formulation

In a classical MILP way, we can model SI-PTPSP by the following TIF using binary variables $x_{a,t}$ for indicating whether an activity $a \in A$ starts at time $t \in T_a$:

$$\min MS \tag{5.19}$$

$$\text{subject to} \sum_{t \in T_a} x_{a,t} = 1 \quad \forall a \in A, \tag{5.20}$$

$$\sum_{t \in T_a} t \cdot x_{a,t} + p_a \leq MS \quad \forall a \in A, \tag{5.21}$$

$$\sum_{a \in A: r \in Q_a} \sum_{t' \in T_a : t \in Y_a(t')} x_{a,t'} \leq 1 \quad \forall r \in R,\ t \in W_r, \tag{5.22}$$

$$\sum_{t \in T_{a'}} t x_{a',t} - \sum_{t \in T_a} t x_{a,t} \geq p_a + L^{\min}_{a,a'} \quad \forall (a,a') \in P, \tag{5.23}$$

$$\sum_{t \in T_{a'}} t x_{a',t} - \sum_{t \in T_a} t x_{a,t} \leq p_a + L^{\max}_{a,a'} \qquad \forall (a,a') \in P, \qquad (5.24)$$

$$x_{a,t} \in \{0,1\} \qquad \forall a \in A,\ t \in T_a, \qquad (5.25)$$

$$MS \geq 0. \qquad (5.26)$$

Equations (5.20) ensure that exactly one starting time is chosen for each activity. Inequalities (5.21) are used to determine the makespan MS. Resource restrictions are enforced by inequalities (5.22). Last but not least, constraints (5.23) and (5.24) guarantee that the precedence relations with their minimum and maximum time lags are respected.

The model has $O(|A| \cdot |T|)$ variables and $O(|T| \cdot (|A| + |R| + |P|))$ constraints. Typically, the LP relaxation of TIF yields substantially tighter dual bounds than the LP relaxation of DEF but its size and solvability strongly depend on the used time discretization.

5.5 Time-Bucket Relaxation

As the number of variables and constraints of TIF becomes fairly large when considering a fine-grained time discretization, directly solving the model may not be a viable approach in practice. We therefore consider a relaxation of it, in which we combine subsequent time slots into so-called *time-buckets*. This model, which we call TBR, yields a dual bound to the optimal value of the original problem but in general not directly a valid solution. Based on TBR we will build our iterative refinement approach in Section 5.6.

Let $B = \{B_1, \dots, B_\beta\}$ be a partitioning of T into subsequent time-buckets. Note that the individual buckets do not need to have the same size. We denote by $I(B) = \{1, \dots, \beta\}$ the index set of B. For all $b \in I(B)$ we define the set of consecutive time slots $B_b = \{B^{\text{start}}_b, \dots, B^{\text{end}}_b\}$ contained in the bucket. Since B is a chronologically ordered partitioning of T, we have $B^{\text{start}}_1 = T^{\min}$, $B^{\text{end}}_\beta = T^{\max}$, and $B^{\text{end}}_b + 1 = B^{\text{start}}_{b+1}$, $\forall b \in I(B) \setminus \{\beta\}$. Additionally, let $W^B_r(b) = |B_b \cap W_r|$ denote the aggregated amount of resource $r \in R$ available over the whole bucket $b \in I(B)$. For an illustration of a bucket partitioning see Figure 5.1.

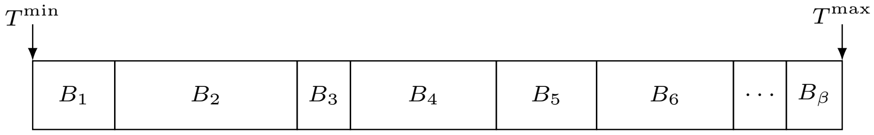


Figure 5.1: Bucket partitioning of T.

Considering a bucket partitioning we now derive for each activity $a \in A$ all subsets of buckets in which the activity can be completely performed such that it executes at least partially in every bucket. We call these subsets *bucket sequences* of activity a and denote them by $C_a = \{C_{a,1}, \dots, C_{a,\gamma_a}\} \subseteq 2^{I(B)}$. Let functions bfirst$(a, c)$ and blast(a, c) for $a \in A$ and $c = 1, \dots, \gamma_a$ provide the index of the first and the last bucket of bucket sequence $C_{a,c}$, respectively. The bucket sequences in C_a are assumed to be ordered according to increasing starting time, or, more precisely, lexicographically ordered according to $(\mathrm{bfirst}(a, c), \mathrm{blast}(a, c))$. We can determine all bucket sequences for an activity in time $O(|B| \log |B|)$, for details see Section 5.7.2. Analogous to set T_a we omit bucket sequences that involve only infeasible starting times.

For each bucket sequence $C_{a,c} \in C_a$ of activity $a \in A$, let $S^{\min}_{a,c} \in T$ be the earliest time slot at which the activity can feasibly start when it is assigned to the bucket sequence. Similarly, let $S^{\max}_{a,c} \in T$ be the latest possible starting point. Moreover, values $z^{\min}_{a,b,c}$ and $z^{\max}_{a,b,c}$ provide bounds on the number of utilized time slots within bucket $b \in C_{a,c}$ when activity a uses bucket-sequence $C_{a,c}$. Note that for inner buckets b with $\mathrm{bfirst}(a, c) < b < \mathrm{blast}(a, c)$ we always have $z^{\min}_{a,b,c} = z^{\max}_{a,b,c} = |B_b|$.

Figure 5.2 shows a set of bucket sequences for a given activity. Observe that for bucket sequence $C_{a,2}$ we need to shift the execution window such that the activity executes at least for one time slot in bucket B_3, i.e., we require $z^{\min}_{a,3,2} > 0$ to avoid an overlap with bucket sequence $C_{a,1}$.

Our relaxation of TIF uses binary variables $y_{a,c}$ indicating whether activity $a \in A$ is performed in bucket sequence $C_{a,c}$ for $c \in \{1, \dots, \gamma_a\}$. Model TBR is stated as follows:

$$
\begin{aligned}
& \min \ MS && && (5.27)\\
& \text{subject to } \sum_{c=1}^{\gamma_a} y_{a,c} = 1 && \forall a \in A, && (5.28)\\
& \sum_{c=1}^{\gamma_a} S^{\min}_{a,c} \cdot y_{a,c} + p_a \le MS && \forall a \in A, && (5.29)\\
& \sum_{a \in A: r \in Q_a} \ \sum_{C_{a,c} \in C_a : b \in C_{a,c}} z^{\min}_{a,b,c} \cdot y_{a,c} \le W^B_r(b) && \forall r \in R,\ b \in I(B), && (5.30)\\
& \sum_{c'=1}^{\gamma_{a'}} S^{\max}_{a',c'} \cdot y_{a',c'} - \sum_{c=1}^{\gamma_a} S^{\min}_{a,c} \cdot y_{a,c} \ge p_a + L^{\min}_{a,a'} && \forall (a, a') \in P, && (5.31)\\
& \sum_{c'=1}^{\gamma_{a'}} S^{\min}_{a',c'} \cdot y_{a',c'} - \sum_{c=1}^{\gamma_a} S^{\max}_{a,c} \cdot y_{a,c} \le p_a + L^{\max}_{a,a'} && \forall (a, a') \in P, && (5.32)\\
& y_{a,c} \in \{0, 1\} && \forall a \in A, c = 1, \dots, \gamma_a, && (5.33)\\
& MS \ge 0. && && (5.34)
\end{aligned}
$$

Equations (5.28) ensure that exactly one bucket sequence is chosen for each activity. The makespan MS is determined using inequalities (5.29). Constraints (5.30) consider the resource availabilities individually for each bucket in an accumulated fashion. Determined resource consumptions of activities are precise for all used inner buckets of a sequence but might underestimate the actually required amount in the first and last bucket. Finally, inequalities (5.31) and (5.32) realize the precedence constraints with their minimum and

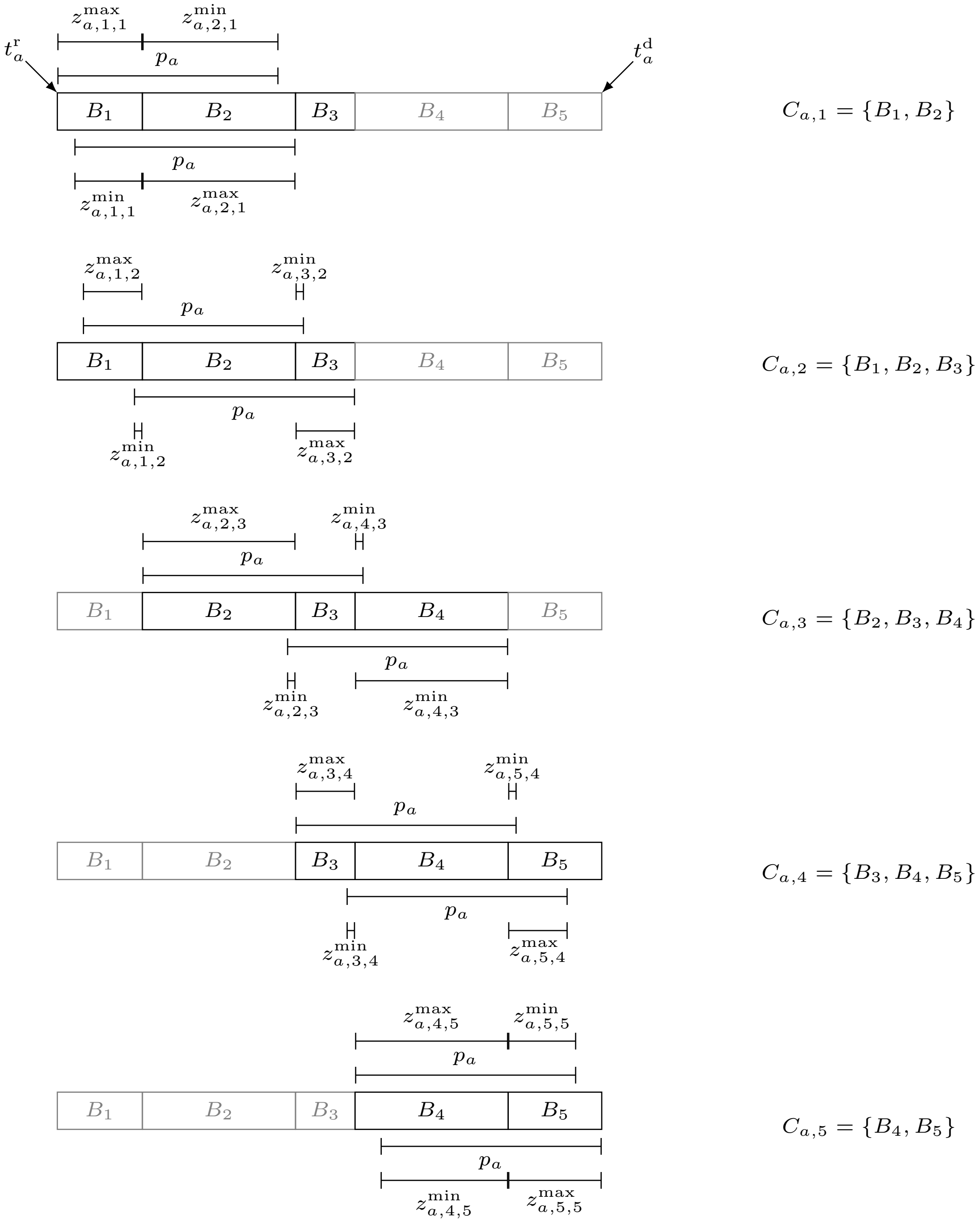


Figure 5.2: Bucket sequences C_a of an activity a with processing time p_a. Descriptions of inner buckets of a sequence are omitted since $z^{\min}_{a,b,c} = z^{\max}_{a,b,c} = |B_b|$ holds for them.

maximum time lags, respectively. These restrictions also constitute a relaxation of the corresponding ones in TIF since the precise starting times within the buckets are not known (unless dealing with buckets of unit size).

The model has $O(|A| \cdot |B|)$ variables and $O(|A| + |R| \cdot |B| + |P|)$ constraints, and thus its size does not depend on $|T|$.

5.5.1 Polyhedral Comparison of TIF and TBR

We start by considering a specific variant of TBR in which all buckets have unit size, i.e., $B = \{\{T^{\min}\}, \{T^{\min} + 1\}, \dots, \{T^{\max}\}\}$. Let us denote this special case by TBR$_1$. This leads to several simplifications. All buckets b belonging to some sequence $C_{a,c}$ are fully used, i.e., $z^{\min}_{a,b,c} = z^{\max}_{a,b,c} = |B_b| = 1$. Moreover, minimum and maximum starting times are equal and equivalent to the first time slot of the initial bucket of the sequence: $S^{\min}_{a,c} = S^{\max}_{a,c} = B^{\text{start}}_{\text{bfirst}(a,c)}$. Essentially, this means that $T_a = \{S^{\min}_{a,c} \mid C_{a,c} \in \mathcal{C}_a\} = \{S^{\max}_{a,c} \mid C_{a,c} \in \mathcal{C}_a\}$ and $|T_a| = |\mathcal{C}_a|$ for all $a \in A$. Furthermore, since buckets correspond to original time slots in this scenario, resource availabilities become binary for each bucket.

For TIF and TBR$_1$ we consider function $\varphi_a \colon \{1, \dots \gamma_a\} \to T_a$ for each activity $a \in A$ with $\varphi_a(c) \coloneqq S^{\min}_{a,c}$.

Proposition 5.1. *Function φ is bijective.*

Proof. Each bucket sequence with respect to TBR$_1$ corresponds to a specific starting time. For each activity, $\mathcal{C}_a$ considers all feasible bucket sequences and T_a all feasible starting times. Thus, there exists a unique mapping between these sets. ☐

Proposition 5.2. *The polyhedra of TBR$_1$ and TIF are isomorphic.*

Proof. We establish an isomorphism between the variables of the models using function φ_a and its inverse: $x_{a,t} = y_{a,\varphi_a^{-1}(t)}$ and $y_{a,c} = x_{a,\varphi_a(c)}$. Moreover, we can use these functions to immediately transform (5.20) into (5.28), (5.21) into (5.29), (5.23) into (5.31), and (5.24) into (5.32) and vice versa. To provide the isomorphism between (5.22) and (5.30) we need a few further considerations. First, recall that all $z^{\min}_{a,b,c}$ constants are equal to 1. Secondly, using $t \leftrightarrow \{t\}$ as isomorphism between T and the set of unit buckets we obtain $W^B_r(b) = 1$ if the corresponding time point $t \in W_r$, and $W^B_r(b) = 0$ otherwise. Finally, the correspondence between time points and unit buckets guarantees that $Y_a(t)$ and $C_{a,c}$ are isomorphic for $\varphi_a^{-1}(t) = c$. Putting things together also the resource constraints can be transformed into one another. ☐

Corollary 5.1. *The LP relaxations of TBR$_1$ and TIF are equally strong.*

Definition 5.1. *Let TBR$_B$ and TBR$_{B'}$ be two TBR models with bucket partitionings B and B', respectively. TBR$_{B'}$ is called a refined model of TBR$_B$ iff $\forall b' \in I(B')\ \exists b \in I(B)\ (B'_{b'} \subseteq B_b)$.*

In the following we show that TBR_B is a relaxation of $TBR_{B'}$ and thus of TIF.

Definition 5.2. *Let TBR_B be a TBR model and let $TBR_{B'}$ be a refined model of TBR_B. Then, $\sigma\colon C'_a \to C_a$ defines a (surjective) mapping from bucket sequences C'_a with respect to $TBR_{B'}$ to bucket sequences C_a with respect to TBR_B satisfying for all $C'_{a,c'} \in C'_a$:*

$$\bigcup_{b' \in C'_{a,c'}} b' \subseteq \bigcup_{b \in \sigma(C'_{a,c'})} b \wedge \forall C_{a,\hat{c}} \in C_a \left(\bigcup_{b' \in C'_{a,c'}} b' \not\subseteq \bigcup_{b \in C_{a,\hat{c}}} b \vee \sigma(C'_{a,c'}) \subseteq C_{a,\hat{c}} \right).$$

This means σ provides the inclusion minimal bucket sequence from TBR_B that contains at least the time slots that the bucket sequence from $TBR_{B'}$ contains.

Lemma 5.1. *Function σ can be implemented by:*

$$\sigma(C'_{a,c'}) = C_{a,c} \textit{ such that } C_{a,c} \in C_a \wedge S^{\min}_{a,c'} \in \mathrm{bfirst}(a,c) \wedge (S^{\min}_{a,c'} + p_a) \in \mathrm{blast}(a,c)$$

Proof. Feasibility of $C'_{a,c'}$ together with the fact that buckets in $TBR_{B'}$ are subsets of those in TBR_B implies that there exists a sequence $C_{a,c} \in C_a$ satisfying $S^{\min}_{a,c'} \in \mathrm{bfirst}(a,c)$ and $(S^{\min}_{a,c'} + p_a) \in \mathrm{blast}(a,c)$. Buckets of sequences $C'_{a,c'}$ are a subset of those from $C_{a,c}$, i.e., $\bigcup_{b' \in C'_{a,c'}} b' \subseteq \bigcup_{b \in C_{a,c}} b$. Moreover, $C_{a,c}$ is uniquely determined since by definition two different bucket sequences cannot have the same first and last buckets. Therefore, every other sequence covering the buckets from $C'_{a,c'}$ must be strictly larger than $C_{a,c}$. ☐

Theorem 5.1. *Let TBR_B be a TBR model and let $TBR_{B'}$ be a refined model of TBR_B. Then, TBR_B is a relaxation of $TBR_{B'}$.*

Proof. Using function σ according to Lemma 5.1, we create a solution y to TBR_B from an optimal solution y^* to $TBR_{B'}$ as follows:

$$y_{a,c} = \sum_{C'_{a,c'} \in C'_a : \sigma(C'_{a,c'}) = C_{a,c}} y^*_{a,c'} \qquad \forall a \in A, c \in \{1, \dots, \gamma_a\}$$

We first show that y is a feasible solution to TBR_B. Constraints (5.28) are satisfied since $y^*_{a,c'}$ is feasible and σ is surjective. As $\mathrm{bfirst}(a,c') \subseteq \mathrm{bfirst}(a,c)$ for all $C_{a,c} = \sigma(C'_{a,c'})$, it holds that $S^{\min}_{a,c} \leq S^{\min}_{a,c'}$ and $S^{\max}_{a,c'} \leq S^{\max}_{a,c}$. Hence, constraints (5.29), (5.31), and (5.32) must hold. If inequalities (5.30) are satisfied for y^*, then the resource constraints are also satisfied for y since the refined resource allocation entails the coarser one. Therefore, y is a feasible solution to TBR_B.

Since $S^{\min}_{a,c} \leq S^{\min}_{a,c'}$, the objective can only decline due to the transformation. Thus, the optimal solution value to TBR_B can be at most as large as the value of the optimal solution to $TBR_{B'}$. Thus, TBR_B is a relaxation of $TBR_{B'}$. ☐

Corollary 5.2. *TBR is a relaxation of TIF.*

5.5.2 Strengthening TBR by Valid Inequalities

In the following we introduce two types of valid inequalities to compensate for the loss of accuracy in TBR due to the bucket aggregation. Note that these inequalities strengthen the relaxation in general but might become redundant for more fine-grained bucket partitionings.

Clique Inequalities

Observe that two activities, represented by non-unit bucket sequences, cannot feasibly start in the same bucket if both require a certain resource. The same holds for two or more bucket sequences with these properties ending in the same bucket. This can be used to derive sets of incompatible bucket sequences that give rise to clique inequalities, see Demassey et al. [50], Hardin et al. [87].

To formulate respective constraints we determine for each $b \in I(B)$ sets $\mathcal{S}_b = \{(a,c) \mid a \in A, c \in C_a, z^{\min}_{a,b,c} < |B_b|, |C_{a,c}| > 1, \mathrm{bfirst}(a,c) = b\}$ and $\mathcal{F}_b = \{(a,c) \mid a \in A, c \in C_a, z^{\min}_{a,b,c} < |B_b|, |C_{a,c}| > 1, \mathrm{blast}(a,c) = b\}$ of non-unit bucket sequences starting and ending in bucket B_b, respectively. From each of these sets we derive a graph having the respective set as vertices and an edge between two vertices if the activities of the corresponding bucket sequences share a resource. Let $\mathcal{C}^{\mathcal{S}}_b$ and $\mathcal{C}^{\mathcal{F}}_b$ be the sets of all maximal cliques with a minimum size of two within these graphs. This leads to the following inequalities:

$$\sum_{(a,c)\in\kappa} y_{a,c} \leq 1 \qquad \forall b \in I(B), \forall \kappa \in \mathcal{C}^{\mathcal{S}}_b, \tag{5.35}$$

$$\sum_{(a,c)\in\kappa} y_{a,c} \leq 1 \qquad \forall b \in I(B), \forall \kappa \in \mathcal{C}^{\mathcal{F}}_b. \tag{5.36}$$

Some of these constraints might be redundant if the sum of $z^{\min}_{a,b,c}$ of the smallest two sequences is already large enough to prohibit them from being in the same bucket by means of inequalities (5.30). The most trivial form of this case is excluded in the above sets by the condition $z^{\min}_{a,b,c} < |B_b|$.

The considered cliques can be computed using the algorithm by Bron and Kerbosch [28]. Cazals and Karande [34]) show that this algorithm is worst-case optimal, i.e., it runs in $O(3^{\frac{n}{3}})$ which is the largest possible number of maximal cliques in a graph on n vertices. Although problematic in general, this might still be reasonable considering the rather small expected size of the conflict graphs.

Nevertheless, in our implementation we decided to avoid clique computations and resort to a simpler variant. We do so by considering a separate graph per resource to obtain a set of not necessarily maximal cliques. This leads to conceptually weaker inequalities but requires almost no computational overhead. More specifically, we consider subsets $\mathcal{S}_{b,r} = \mathcal{S}_b \cap \{(a,c) \mid a \in A, c \in C_a, r \in Q_a\}$ of $\mathcal{S}_b$ and subsets $\mathcal{F}_{b,r} = \mathcal{F}_b \cap \{(a,c) \mid a \in A, c \in C_a, r \in Q_a\}$ of $\mathcal{F}_b$, respectively, for $b \in I(B)$ and $r \in R$, such that within these subsets all activities require a common resource. Using these sets we formulate the following constraints:

$$\sum_{(a,c)\in\mathcal{S}_{b,r}} y_{a,c} \leq 1 \qquad \forall b \in I(B), \forall r \in R : |\mathcal{S}_{b,r}| \geq 2, \tag{5.37}$$

$$\sum_{(a,c)\in\mathcal{F}_{b,r}} y_{a,c} \leq 1 \qquad \forall b \in I(B), \forall r \in R : |\mathcal{F}_{b,r}| \geq 2. \tag{5.38}$$

If mutual overlap of the resources required by the activities is rare, the simpler inequalities are often almost as powerful as the full clique inequalities.

Path Inequalities

The idea of this kind of inequalities is to extend the precedence constraints (5.31) and (5.32), and the makespan constraints (5.29) from pairs of adjacent activities or single activities, respectively, to paths within the precedence graph.

We consider the acyclic directed precedence graph $G = (A, P)$. Let $\pi_{a_0,a_m} = (a_0, a_1, \ldots, a_m)$ be a directed path from activity a_0 to activity a_m in G. Moreover, let $d_{L^{\min}}(\pi_{a_0,a_m}) = \sum_{i=0}^{m-1} p_{a_i} + L^{\min}_{a_i,a_{i+1}}$ and $d_{L^{\max}}(\pi_{a_0,a_m}) = \sum_{i=0}^{m-1} p_{a_i} + L^{\max}_{a_i,a_{i+1}}$ be the minimum and maximum makespan of the activities within the path, respectively. Let $\Pi_{a,a'}$ denote the set of all distinct paths from node a to node a'. Since G is acyclic, $\Pi_{a,a'}$ is finite (but in general exponential in the number of edges) for all pairs of nodes $(a, a') \in A \times A : a \neq a'$. Let $\Pi = \bigcup_{a,a'\in A:a\neq a} \Pi_{a,a'}$ denote the union of all these paths between any two different nodes.

Let S be a feasible solution to SI-PTPSP. Then, for each path $\pi_{a,a'}$ in G it must hold that $S_a + d_{L^{\min}}(\pi_{a,a'}) \leq S_{a'}$ and $S_a + d_{L^{\max}}(\pi_{a,a'}) \geq S_{a'}$. Hence, adding the following inequalities to TBR yields a strengthened relaxation of TIF:

$$\sum_{c=0}^{\gamma_a-1} S^{\min}_{a,c} \cdot y_{a,c} + d_{L^{\min}}(\pi_{a,a'}) \leq \sum_{c'=0}^{\gamma_{a'}-1} S^{\max}_{a',c'} \cdot y_{a',c'} \qquad \forall \pi_{a,a'} \in \Pi, \tag{5.39}$$

$$\sum_{c=0}^{\gamma_a-1} S^{\max}_{a,c} \cdot y_{a,c} + d_{L^{\max}}(\pi_{a,a'}) \geq \sum_{c'=0}^{\gamma_{a'}-1} S^{\min}_{a',c'} \cdot y_{a',c'} \qquad \forall \pi_{a,a'} \in \Pi, \tag{5.40}$$

$$\sum_{c=0}^{\gamma_a-1} S^{\min}_{a,c} \cdot y_{a,c} + d_{L^{\min}}(\pi_{a,a'}) + p_{a'} \leq MS \qquad \forall \pi_{a,a'} \in \Pi. \tag{5.41}$$

Due to the exponential number of these inequalities we only consider a reasonable subset of them in our implementation, for details see Section 5.7.3.

5.6 Iterative Time-Bucket Refinement Algorithm

For the original SI-PTPSP, TBR on its own is a method yielding a lower bound but no concrete feasible solution. The basic idea of ITBRA is to solve TBR repeatedly and to

```
Algorithm 5.1: Iterative time-bucket refinement algorithm
Input: SI-PTPSP instance
Output: solution to SI-PTPSP and lower bound
compute initial bucket partitioning
compute initial primal solution
do
    solve TBR for the current bucket partitioning
    apply GCH: try to find an SI-PTPSP solution in accordance with the TBR
      solution
    if unscheduled activities remain then
        apply follow-up heuristic to find feasible SI-PTPSP solution
    end
    if gap closed then
        return optimal solution
    end
    derive refined bucket partitioning for the next iteration
while termination criteria not met
return best heuristic solution and lower bound from TBR
```

refine the bucket partitioning in each iteration until a proven optimal solution can be derived via primal heuristics or some other termination criterion is met. We will show that, given enough time, our algorithm converges to an optimal SI-PTPSP solution.

More specifically, we start by solving TBR with an initial bucket partitioning. Then, we try to heuristically derive an SI-PTPSP solution that matches the objective value of TBR with a so-called GCH. This heuristic fixes times for the activities in accordance with the TBR solution and guarantees to never violate resource or precedence constraints. If all activities can be scheduled in this way, we have found an optimal solution and the algorithm terminates. Otherwise, some activities remain unscheduled and we apply a follow-up heuristic to augment and repair the partial solution, possibly obtaining a feasible approximate solution and a primal bound. Here it can again be the case that we are able to close the optimality gap. If no provably optimal solution has been found thus far, we refine the bucket partitioning by splitting buckets and solve TBR again. For selecting the buckets to be refined and doing the splitting, we exploit information obtained from the TBR solution and the applied primal heuristics. This process is iterated until specified termination criteria are met or an optimal solution is found. The whole procedure is outlined in Algorithm 5.1. The individual components of this approach will be explained in detail in the upcoming sections.

5.6.1 Initial Bucket Partitioning

We create the initial bucket partitioning B in such a way that buckets start/end at any time where a resource availability interval starts or ends and at any release time and

Algorithm 5.2: Computing an initial bucket partitioning

```
Output: the initial bucket partitioning
$B \leftarrow \emptyset$ // bucket partitioning
$\mathcal{T} \leftarrow \{T^{\min}\} \cup \{T^{\max}+1\}$ // bucket starting times
$\mathcal{T} \leftarrow \mathcal{T} \cup \{W^{\text{start}}_{r,w}, W^{\text{end}}_{r,w}+1 \mid r \in R,\ w = 1,\dots,\omega_r\}$
$\mathcal{T} \leftarrow \mathcal{T} \cup \{t^{\text{r}}_a, t^{\text{d}}_a \mid a \in A\}$
sort $\mathcal{T}$
for $i \leftarrow 1$ to $|\mathcal{T}|-1$ do
    $B \leftarrow B \cup \{\{\mathcal{T}[i],\dots,\mathcal{T}[i+1]-1\}\}$ // add bucket
end
return $B$
```

deadline of the activities. The first restriction ensures that resources are either available for the whole duration of a bucket or not at all. For details see Algorithm 5.2.

5.6.2 Primal Heuristics

We consider heuristics that attempt to derive feasible SI-PTPSP solutions and corresponding primal bounds based on TBR solutions. If ITBRA is terminated early, the best solution found in this way is returned. Note, however, that depending on the instance properties, these heuristics might also fail and yield no feasible solution.

Gap Closing Heuristic

This is the first heuristic applied during an iteration of ITBRA. It attempts to construct an optimal solution according to TBR's result to close the optimality gap. Thus, it may only fully succeed when the relaxation's objective value does not underestimate the optimal SI-PTPSP solution value, i.e., provides a tight dual bound. If the gap cannot be closed, GCH provides only a partial solution and no primal bound. Information on the unscheduled activities then forms an important basis for the subsequent bucket refinement.

Let (y^*, MS^*) be the current optimal TBR solution. Initially, GCH receives for each activity $a \in A$ the interval $S^{\text{TBR}}_a = \{S^{\text{TBR,min}}_a, \dots, S^{\text{TBR,max}}_a\}$ of potential starting times, where $S^{\text{TBR,min}}_a = \sum_{c=0}^{\gamma_a-1} S^{\min}_{a,c} \cdot y^*_{a,c}$ and $S^{\text{TBR,max}}_a = \sum_{c=0}^{\gamma_a-1} S^{\max}_{a,c} \cdot y^*_{a,c}$. These intervals can in general be further reduced by removing for each $a \in A$ all time slots $t \in S^{\text{TBR}}_a$ violating at least one of the following conditions in relation to the precedence constraints and the calculation of the makespan:

$$\exists t' \in S^{\text{TBR}}_{a'}\ (t + p_a + L^{\min}_{a,a'} \le t' \le t + p_a + L^{\max}_{a,a'}) \qquad \forall (a,a') \in P, \tag{5.42}$$

$$\exists t' \in S^{\text{TBR}}_{a'}\ (t' + p_{a'} + L^{\min}_{a',a} \le t \le t' + p_{a'} + L^{\max}_{a',a}) \qquad \forall (a',a) \in P, \tag{5.43}$$

$$t + p_a \le MS^*. \tag{5.44}$$

We prune the set of intervals of potential activity starting times for all activities $S^{\mathrm{TBR}} = \{S_a^{\mathrm{TBR}} \mid a \in A\}$ so that *arc consistency* is achieved with respect to conditions (5.42)–(5.44). This is done by constraint propagation with a method like the well-known AC3 algorithm, see Mackworth [119]. Note that constraint propagation may yield empty intervals for some activities, indicating that there remains no feasible starting time assignment respecting all constraints. In this case GCH gives up on this activity and continues with the remaining ones, deviating from the usual arc consistency concept, to allow further activities to be scheduled.

The pseudocode of GCH is shown in Algorithm 5.3. After the initial pruning of starting time intervals, GCH constructs the (partial) schedule S by iteratively scheduling the activities while respecting all constraints. If this is not possible for some activities, they remain unscheduled. Using a greedy strategy, the activities are considered in non-decreasing order of $S_a^{\mathrm{TBR,max}} + p_a$, i.e., according to their earliest possible finishing times. Activities are always scheduled at the earliest feasible time from S_a^{TBR}. Note that any explicit enumeration of time slots from an interval can be efficiently avoided by using basic interval arithmetic. Whenever an activity starting time is set, constraint propagation is repeated to ensure arc consistency according to conditions (5.42)–(5.44).

If GCH fails to close the gap, we attempt to compute a feasible solution instead that might have a larger objective value than the current TBR bound.

Activity Block Construction Heuristic

This algorithm is based on the idea of first constructing so-called *activity blocks*, which correspond to the weakly connected components of the precedence graph. All the activities belonging to one such weakly connected component are statically linked considering the precedence constraints and minimum time lags between them. ABCH then greedily schedules the activity blocks that have not been scheduled completely by GCH instead of the individual activities. The activity blocks are considered in order of their release times and are scheduled at the first time slot where no resource constraint is violated with respect to the activity block's individual activities and resource requirements. Details are provided in Algorithm 5.4.

Greedy Randomized Adaptive Search Procedure

We consider GRASP as an advanced alternative to ABCH within ITBRA. The approach provides a reasonable balance between being still relatively simple but providing considerably better results than ABCH. There are clearly other options but our aim here is to keep standard metaheuristic aspects simple in order to put more emphasis on the fundamentals of TBR and ITBRA.

Both, GCH and ABCH can be randomized. We do so by allowing the order in which the activities or activity blocks are considered to deviate from the strict greedy criterion. In particular, we choose uniformly at random from the $k_{\mathrm{GCH}}^{\mathrm{grand}}$ ($k_{\mathrm{ABCH}}^{\mathrm{grand}}$) candidates with the highest priority. Parameters $k_{\mathrm{GCH}}^{\mathrm{grand}}$ and $k_{\mathrm{ABCH}}^{\mathrm{grand}}$ control the strength of the randomization.

Algorithm 5.3: Gap closing heuristic

```
Input: intervals of potential starting times $S^{\mathrm{TBR}} = \{S_a^{\mathrm{TBR}} \mid a \in A\}$ with
       $S_a^{\mathrm{TBR}} = \{S_a^{\mathrm{TBR,min}}, \ldots, S_a^{\mathrm{TBR,max}}\}$
Output: (partial) schedule $S$ and all activities that cannot be scheduled with
        respect to $S^{\mathrm{TBR}}$ grouped by violation type
$A_P \leftarrow \emptyset$ // activities with violated precedence constraints
$A_R \leftarrow \emptyset$ // activities with violated resource constraints
$A_U \leftarrow A$ // unscheduled activities
$W'_r \leftarrow W_r$ // resource availabilities
prune potential starting time intervals $S^{\mathrm{TBR}}$
while $A_U \neq \emptyset$ do
    select and remove an activity $a \in A_U$ with minimal $S_a^{\mathrm{TBR,max}} + p_a$
    if $S_a^{\mathrm{TBR}} = \emptyset$ then // precedence constraints violated
        $A_P \leftarrow A_P \cup \{a\}$
        continue
    end
    $\overline{S_a^{\mathrm{TBR}}} \leftarrow \{t \in S_a^{\mathrm{TBR}} \mid \{t, \ldots, t + p_a - 1\} \subseteq W'_r, \forall r \in Q_a\}$
    if $\overline{S_a^{\mathrm{TBR}}} = \emptyset$ then // resource constraints violated
        $A_R \leftarrow A_R \cup \{a\}$
        continue
    end
    $S_a \leftarrow \min\ \overline{S_a^{\mathrm{TBR}}}$
    $S_a^{\mathrm{TBR}} \leftarrow \{S_a\}$
    $W'_r \leftarrow W'_r \setminus \{t, \ldots, t + p_a - 1\},\ \forall r \in Q_a$
    prune potential starting time intervals $S^{\mathrm{TBR}}$
end
return $S, A_P, A_R$;
```

Note that the success of ABCH and hence also of the GRASP strongly depends on the partial solution provided by GCH. Therefore, we primarily choose to randomize GCH. Within ITBRA we also try to compute a primal solution at the very beginning before solving TBR for the first time. Hence, there is no GCH solution available at this point. In this case we randomize ABCH instead.

To get a strong guidance for the bucket refinement process we prefer GCH solutions that schedule as many activities as possible. However, these solutions might not necessarily work best in conjunction with ABCH. Therefore, we track the best complete solution and the best partial GCH solution separately. This means that our GRASP returns a feasible SI-PTPSP solution as well as a partial GCH solution (which might be unrelated). Since GRASP combines the functionalities of GCH and ABCH, it effectively replaces Lines 5–8 in Algorithm 5.1.

Algorithm 5.4: Activity block construction heuristic

```
Input: a partial schedule $S^{\mathrm{GCH}}$ computed by GCH
Output: a feasible schedule $S$ or no solution if $S^{\mathrm{GCH}}$ cannot be completed
$C \leftarrow$ set of subsets of $A$ corresponding to the weakly connected components of the
    precedence graph which are not completely scheduled in $S^{\mathrm{GCH}}$
$A^C \leftarrow \emptyset$ // the set of activity blocks
forall weakly connected components $c \in C$ do
    $S^c \leftarrow \emptyset$ // a schedule representing the activity block of $c$
    forall activities $a \in c$ in topological order do
        schedule $a$ in $S^c$ at the earliest possible time with respect to the precedence
            constraints and resource consumptions of activities in $c$ but ignoring all
            other activities as well as release times and deadlines, and resource
            availabilities
    end
    the release time of the activity block is $\min_{a \in c} t^{\mathrm{r}}_a$
    $A^C \leftarrow A^C \cup \{S^c\}$
end
forall activity blocks $S^c \in A^C$ ordered according to release time do
    try to schedule the activity block at the earliest feasible time in $S$ such that
        activity release times and deadlines as well as resource constraints are satisfied
    if no feasible time found then
        return no solution
    end
end
return $S$
```

We consider a local search component using a classical 2-exchange neighborhood on the order of the activity blocks scheduled by ABCH. The local search is always performed until a local optimum is reached.

As termination criterion for the GRASP a combination of a time limit and a maximal number of iterations without improvement is used, details will be given in Section 5.8. Moreover, in the first iteration of the GRASP the deterministic versions of GCH and ABCH are used. This guarantees, especially for short executions, that the final result of the GRASP is never worse than the one of the pure heuristics.

5.6.3 Bucket Refinement Strategies

In general, the bucket refinement is done by selecting one or more existing buckets and splitting each of them at selected points into two or more new buckets. If a bucket consists of only a single time slot, it cannot be subdivided further and becomes irrelevant for subsequent splitting decisions. Buckets are never merged or extended in our approach, i.e., the number of buckets always strictly increases due to the refinement. This guarantees

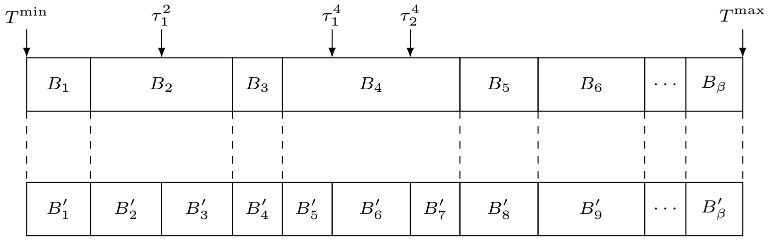


Figure 5.3: An example of a bucket refinement for $\tau^2 = \{\tau_1^2\}$, $\tau^4 = \{\tau_1^4, \tau_2^4\}$, and $\tau^b = \emptyset$ for $b \in I(B) \setminus \{2,4\}$.

that ITBRA eventually terminates if at least one bucket is subdivided in each iteration (cf. Theorem 5.1).

More formally, a refinement of some bucket $B_b \in B$ is specified by an ordered set of splitting points $\tau^b = \{\tau_1^b, \ldots, \tau_m^b\} \subseteq \{B_b^{\text{start}} + 1, \ldots, B_b^{\text{end}}\}$ with $\tau_1^b < \ldots < \tau_m^b$. Based on τ^b we get $|\tau^b| + 1$ new buckets replacing the original one: $\{B_b^{\text{start}}, \ldots, \tau_1^b - 1\}, \{\tau_1^b, \ldots, \tau_2^b - 1\}, \ldots, \{\tau_m^b, \ldots, B_b^{\text{end}}\}$. For an example see Figure 5.3.

In general, the decisions to be made in the bucket refinement process are (1) which buckets are to be refined, (2) at which positions, and (3) how many splits to apply. To address these tasks we need criteria that identify promising bucket refinements. Most importantly, a bucket refinement should affect the current optimal TBR solution in order to guarantee that not the same bucket sequences as before comprise an optimal solution again. In this way, it is ensured that we obtain a more refined solution in each iteration. Furthermore, bucket splitting should be done in such a way that it is beneficial for the heuristics, helping them to find good feasible solutions. Therefore, constraints that were responsible for leaving activities unscheduled in the heuristics should be exploited to prevent these situations from occurring again. Last but not least, we want to obtain a dual bound for the SI-PTPSP that is as tight as possible. Hence, a bucket refinement that likely has implications on TBR's objective value is desirable.

Selecting Buckets to Refine

Observe that refining inner buckets of selected bucket sequences does not directly affect the current TBR solution. Refining first and last buckets (if they are non-unit buckets), however, ensures that the bucket sequence that contained them is not present in the refined TBR and therefore cannot be used again. Furthermore, some of the newly introduced buckets might not be part of feasible bucket sequences, resulting in a more restricted scenario. Hence, we want to either split only first or last buckets of selected sequences or both. If we use just one of these options, we need to resort to the other whenever no progress can be made otherwise. During preliminary tests it turned out that always using both boundary buckets for refinement is superior. Another question is for which bucket sequences the bounding buckets shall be refined. In the following we propose different strategies that will be experimentally compared in Section 5.8.2.

All selected (ASEL). Using this strategy we refine all first and last buckets of all bucket sequences selected in the current optimal TBR solution. This can, however, be inefficient as it may increase the total number of buckets in each iteration substantially. The following strategies will therefore only consider certain subsets.

All in GCH schedule (AIGS). We refine all first and last buckets of only those bucket sequences whose corresponding activities could be feasibly scheduled by GCH. The idea is to improve accuracy for the scheduled activities in order to reveal sources of infeasibility with respect to the activities that could not be scheduled.

Violated due (VDUE). If GCH fails to schedule all activities, it provides a set of activities A_P that cannot be scheduled due to the precedence constraints and a set of activities A_R that cannot be scheduled due to the resource constraints. The basic idea is to refine buckets related to activities in the schedule that immediately prevent the activities in A_P and A_R from being scheduled. To identify these activities we consider the partial schedule S generated by GCH.

Let $A^{\mathrm{GCH}} = A \setminus (A_R \cup A_P)$ be the set of feasibly scheduled activities. Refinements based on resource infeasibilities are derived from sets $N_R(a) = \{a' \in A^{\mathrm{GCH}} \mid Q_a \cap Q_{a'} \neq \emptyset \wedge \{S_{a'}, \ldots, S_{a'} + p_{a'} - 1\} \cap \{S_a^{\mathrm{TBR,min}}, \ldots, S_a^{\mathrm{TBR,max}} + p_a - 1\} \neq \emptyset\}$ for $a \in A_R$. For each activity $a' \in N_R(a)$ we refine the first and last bucket of the bucket sequence $C_{a',c}$ in the TBR solution.

The activities potentially responsible for $a \in A_P$ having no valid starting time are the activities a' in A^{GCH} such that $(a, a') \in P$ or $(a', a) \in P$. However, we do not have to consider all activities adjacent to a for the refinement. Let $N_P^-(a) = \{a' \mid (a', a) \in P \wedge a' \in A^{\mathrm{GCH}}\}$ and $N_P^+(a) = \{a' \mid (a, a') \in P \wedge a' \in A^{\mathrm{GCH}}\}$ for all $a \in A_P$. Then, calculate:

$$\begin{aligned} N_P(a) = & \operatorname*{arg\,max}_{a' \in N_P^-(a)} \{S_{a'} + p_{a'} + L_{a',a}^{\min}\} && \cup \quad \operatorname*{arg\,min}_{a' \in N_P^-(a)} \{S_{a'} + p_{a'} + L_{a',a}^{\max}\} \quad \cup \\ & \operatorname*{arg\,min}_{a' \in N_P^+(a)} \{S_{a'} - L_{a,a'}^{\max}\} && \cup \quad \operatorname*{arg\,max}_{a' \in N_P^+(a)} \{S_{a'} - L_{a,a'}^{\min}\}. \end{aligned} \tag{5.45}$$

We refine the first and last buckets of all bucket sequences of activities $a' \in N_P(a)$ that are selected in the current TBR solution. If no refinement is possible for bucket sequences corresponding to $a' \in N_R(a) \cup N_P(a)$, we refine the first and last bucket of $C_{a,c}$ instead.

Identifying Splitting Positions

Once a bucket has been selected for refinement, we have to decide at which position(s) it shall be subdivided. Again, we consider different strategies. The challenge is to identify candidate positions that usually have a large impact on the subsequent TBR and its solution while resulting in well-balanced sub-buckets.

Binary (B). Let $C_{a,c}$ be the bucket sequence causing its first and last buckets to be selected for refinement. We split the associated buckets in such a way that the interval of potential starting and finishing times of the respective activity is bisected. In particular, for $\mathrm{bfirst}(a,c)$ and $\mathrm{blast}(a,c)$, we consider the splitting positions $\lceil (S^{\min}_{a,c} + S^{\max}_{a,c})/2 \rceil$ and $\lceil (S^{\max}_{a,c} + S^{\min}_{a,c})/2 \rceil + p_a$, respectively. We have to round up in case of non integral refinement positions since it is not feasible to refine with respect to the bucket start. Although this approach typically leads to well-balanced sub-buckets, it might often have a rather weak impact on the subsequent TBR solution because the resulting buckets might still be too large to reveal certain sources of infeasibility.

Start and end time (SET). Let a be an activity that could be scheduled by GCH and $C_{a,c}$ the corresponding bucket sequence in TBR whose first and last buckets shall be refined. We split $\mathrm{bfirst}(a,c)$ at the activity's starting time S_a and $\mathrm{blast}(a,c)$ at $S_a + p_a$, i.e., after activity a has ended according to GCH's schedule. Thus, the specifically chosen time assignment of GCH gets an individual bucket sequence in the next iteration.

Because this method is defined only for activities that could be scheduled by GCH, it is applicable only in direct combination with AIGS. To overcome this limitation we resort to B if SET is not applicable. The obtained strategy is denoted by SET+B.

Selecting Splitting Positions

The strategies introduced above may yield several splitting positions for a single bucket, especially since the same bucket may be selected multiple times for refinement for different activities. In principle, we want to generate as few new buckets as possible while ensuring strong progress with respect to the dual bound and narrowing down the activities' possible starting time intervals. Splitting at all identified positions might therefore not be the best option. In the following we propose different strategies to determine for each selected bucket the splitting positions to be actually used from all positions collected in the previous step. Let set τ^b be this union of identified splitting positions for bucket b.

Union refinement (UR). We simply use all identified splitting positions. As already mentioned, however, this approach may lead to a high increase in the number of buckets and may therefore not be justified.

Binary refinement (BR). We use the splitting position $\tau' \in \tau^b$ closest to the center of the bucket, i.e., $\tau' = \arg\min_{t \in \tau^b} \left| (B^{\mathrm{start}}_b + B^{\mathrm{end}}_b)/2 - t \right|$; ties are broken according to the order in which the splitting positions have been obtained. This approach clearly tends to keep the number of buckets low but may increase the total number iterations required by ITBRA to prove optimality.

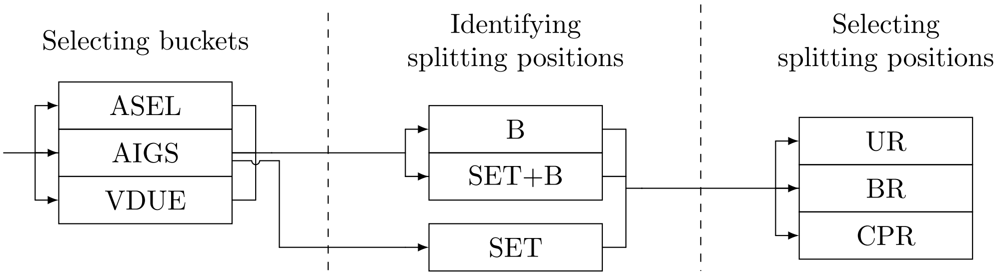

Selecting buckets
Identifying splitting positions
Selecting splitting positions
ASEL
AIGS
VDUE
B
SET+B
SET
UR
BR
CPR

Figure 5.4: Overview of the proposed strategies to perform a bucket refinement and how they can be combined.

Centered partition refinement (CPR). We first partition τ^b into two sets at $\bar{t} = (B_b^{\text{start}} + B_b^{\text{end}})/2$. Let $\tau^{b,\text{l}} = \{t \in \tau^b \mid t \leq \bar{t}\}$ and $\tau^{b,\text{r}} = \{t \in \tau^b \mid t > \bar{t}\}$. To obtain up to three new buckets we choose as splitting points the two "innermost" elements, i.e., we apply the refinement $\{\max \tau^{b,\text{l}}, \min \tau^{b,\text{r}}\}$. If one of the sets is empty, we apply only a single split.

The idea of this partitioning is to give candidate positions close to either boundary of the bucket equal chances of being selected. Splitting a bucket close to its end usually has a strong influence on (non-unit) bucket sequences starting in the bucket while choosing a splitting position close to the start typically has a higher impact on (non-unit) bucket sequences ending in this bucket. Prioritizing splitting positions close to the center of the bucket results in a more balanced subdivision.

Further Considerations

We also investigated bucket selection techniques based on critical paths, see Guerriero and Talarico [84]. This means that we consider sequences of activities that directly define the makespan. However, our experiments indicate that bucket refinements based on this strategy do not work well. We therefore omit them, as well as a few other inferior techniques, here and refer the interested reader to Jatschka [94] for further details.

Figure 5.4 provides an overview of the discussed bucket selection, splitting position identification, and splitting position selection strategies.

5.7 Implementation Details

In this section, we discuss further algorithmic details that are important for an efficient implementation of ITBRA and the associated heuristics.

5.7.1 Preprocessing Activity Starting Times

To obtain the restricted set of possible activity starting times T_a we start by discarding the starting times leading to resource infeasibilities:

$$T_a = \{t \in T \mid t^{\mathrm{r}}_a \le t \le t^{\mathrm{d}}_a - p_a, \forall r \in Q_a \; \forall t' \in Y_a(t) \; (t' \in W_r)\}$$

The obtained set is then further reduced by taking also precedence relations into account. In particular, only starting times respecting the following conditions are feasible:

$$\begin{aligned} &\forall (a, a') \in P \; \exists t' \in T_{a'} \; (t + p_a + L^{\min}_{a,a'} \le t' \le t + p_a + L^{\max}_{a,a'}) \\ &\forall (a', a) \in P \; \exists t' \in T_{a'} \; (t' + p_{a'} + L^{\min}_{a',a} \le t \le t' + p_{a'} + L^{\max}_{a',a}). \end{aligned}$$

To achieve arc consistency with respect to them we can use constraint propagation similar as in GCH. All these calculations can be performed based on interval arithmetic without enumerating individual time slots, and thus in time independent of $|T|$.

Finally, the originally given release times and deadlines can be tightened according to the pruned sets T_a, i.e., we set

$$\begin{aligned} t^{\mathrm{r}}_a &\leftarrow \min T_a && \forall a \in A, \\ t^{\mathrm{d}}_a &\leftarrow p_a + \max T_a && \forall a \in A. \end{aligned}$$

5.7.2 Computing Bucket Sequences

Algorithm 5.5 calculates the bucket sequences C_a for an activity $a \in A$ using the fact that bucket sequences are uniquely determined by their earliest possible starting times $S^{\min}_{a,c}$. In particular, we can efficiently compute the next such time point that needs to be considered from the previous one.

If the current bucket sequence consists of a single bucket, we proceed with the time point ensuring that only $p_a - 1$ time can be spent in the current bucket, see Line 12. Otherwise, we try to find the earliest time point that guarantees that we start in b^{first} and finish in bucket $b^{\text{last}} + 1$. If no such time point exists, we proceed with the earliest time slot in bucket $b^{\text{first}} + 1$ instead or stop if the activity's deadline has already been reached. The offset, denoted by δ, to the sought time point can be computed according to Line 16.

Iterating over the earliest starting times is linear in the number of buckets. The bucket to which a certain time slot belongs can be determined in logarithmic time with respect to the number of buckets. Hence, the overall time required by the algorithm is in $O(|B| \log |B|)$. Note that the $z^{\min}_{a,b,c}$ and $z^{\max}_{a,b,c}$ values are only set for the first and last buckets of the computed sequences since these values are always equal to the bucket size for all inner buckets.

For $C_{a,c} \in C_a$ let $T^{\mathrm{s}}_{a,c} = \{S^{\min}_{a,c}, \ldots, S^{\max}_{a,c}\} \cap T_a$. We can discard all bucket sequences for which $T^{\mathrm{s}}_{a,c} = \emptyset$. Moreover, $S^{\min}_{a,c}$ and $S^{\max}_{a,c}$ can be tightened by setting $S^{\min}_{a,c}$ to $\min(T^{\mathrm{s}}_{a,c})$ and $S^{\max}_{a,c}$ to $\max(T^{\mathrm{s}}_{a,c})$.

Algorithm 5.5: Computing all bucket sequences for an activity

```
Input: activity $a \in A$
Output: set of bucket sequences $C_a$, associated values $S^{\min}_{a,c}$, $S^{\max}_{a,c}$, $z^{\min}_{a,b,c}$, $z^{\max}_{a,b,c}$
$C_a \leftarrow \emptyset$
$t \leftarrow t^{\mathrm{r}}_a$
$c \leftarrow 1$
while $t \leq t^{\mathrm{d}}_a - p_a$ do
    $b^{\mathrm{first}} \leftarrow b : t \in B_b$
    $b^{\mathrm{last}} \leftarrow b : (t + p_a - 1) \in B_b$
    $C_{a,c} \leftarrow \{B_{b^{\mathrm{first}}}, \dots, B_{b^{\mathrm{last}}}\}$
    $S^{\min}_{a,c} \leftarrow t$
    if $b^{\mathrm{first}} = b^{\mathrm{last}}$ then
        $z^{\min}_{a,b^{\mathrm{last}},c} \leftarrow p_a$
        $z^{\max}_{a,b^{\mathrm{last}},c} \leftarrow p_a$
        $t \leftarrow B^{\mathrm{end}}_{b^{\mathrm{last}}} - p_a + 2$
    else
        $z^{\max}_{a,b^{\mathrm{first}},c} \leftarrow B^{\mathrm{end}}_{b^{\mathrm{first}}} - t + 1$
        $z^{\min}_{a,b^{\mathrm{last}},c} \leftarrow S^{\min}_{a,c} + p_a - B^{\mathrm{start}}_{b^{\mathrm{last}}}$
        $\delta \leftarrow \min\left\{z^{\max}_{a,b^{\mathrm{first}},c} - 1, \min\left\{B^{\mathrm{end}}_{b^{\mathrm{last}}}, t^{\mathrm{d}}_a - 1\right\} - \left(S^{\min}_{a,c} + p_a - 1\right)\right\}$
        $z^{\min}_{a,b^{\mathrm{first}},c} \leftarrow z^{\max}_{a,b^{\mathrm{first}},c} - \delta$
        $z^{\max}_{a,b^{\mathrm{last}},c} \leftarrow z^{\min}_{a,b^{\mathrm{last}},c} + \delta$
        $t \leftarrow S^{\min}_{a,c} + \delta + 1$
    end
    $S^{\max}_{a,c} = B^{\mathrm{end}}_{b^{\mathrm{first}}} - z^{\min}_{a,b^{\mathrm{first}},c} + 1$
    $C_a \leftarrow C_a \cup \{C_{a,c}\}$
    $c \leftarrow c + 1$
end
return $C_a$;
```

5.7.3 Valid Inequalities

As already mentioned, we only consider the simplified version of the clique inequalities (5.37) and (5.38) to avoid the overhead for computing maximal cliques. The number of these inequalities grows significantly as the buckets get more fine-grained. Fortunately, the final bucket partitionings turned out to be sufficiently coarse to add all inequalities of this type to the initial formulation.

Recall that the number of path inequalities (5.39)–(5.41) is in general exponential. In favor of keeping the model compact we avoided dynamic separation and consider only a reasonable subset of these inequalities that is added in the beginning. Clearly, we want to use a subset of the paths Π still having a strong influence on the relaxation. The idea is to use all paths targeting vertices of the precedence graph with an out-degree of zero. This guarantees that precedence relations are enforced more strictly between all sinks and their predecessors. Since the sinks in the precedence graph are the nodes that will define the makespan, this appears to be particularly important.

To this end, we consider the following subsets of Π with $\deg^+(\cdot)$ denoting the out-degree of a node:

$$\begin{aligned}
\Pi_{L^{\min}} &= \bigcup_{a,a' \in A: a \neq a'} \{ \operatorname*{arg\,max}_{\pi_{a,a'} \in \Pi_{a,a'}} d_{L^{\min}}(\pi_{a,a'}) \mid \Pi_{a,a'} \neq \emptyset, \deg^+(a') = 0 \},\\
\Pi_{L^{\max}} &= \bigcup_{a,a' \in A: a \neq a'} \{ \operatorname*{arg\,min}_{\pi_{a,a'} \in \Pi_{a,a'}} d_{L^{\max}}(\pi_{a,a'}) \mid \Pi_{a,a'} \neq \emptyset, \deg^+(a') = 0 \}.
\end{aligned}$$

We then add inequalities (5.39) and (5.41) only for paths $\pi_{a,a'} \in \Pi_{L^{\min}}$ and inequalities (5.40) only for paths $\pi_{a,a'} \in \Pi_{L^{\max}}$.

5.8 Computational Study

In this section we are going to present the computational results for the considered algorithms with their variants. We start by giving details on the used test instances and the motivation for their selection. Then, we provide details on the actually used configurations. Finally, we present the obtained results.

5.8.1 Test Instances

The benchmark instances are motivated by the real world patient scheduling scenario at cancer treatment center MedAustron that requires scheduling of particle therapies. In general, each treatment session consists of five activities that have to be performed sequentially. The modeled resources are the particle beam, the irradiation rooms, the radio oncologists, and the anesthetist. In principle, resources are assumed to be available for the whole time horizon except for short periods of time. The most critical resource is the particle beam, which is required by exactly one activity of each treatment. The particle beam is shared between three irradiation rooms, in which also additional preparation and follow-up tasks have to be performed. A radio oncologist is required for the first and the last activity, respectively. In addition, some patients require sedation, which means that the anesthetist is involved in all activities.

The main characteristic of our benchmark instances is the number of activities. We have generated two groups of benchmark instances, each consisting of 15 instances per number of activities $\alpha \in \{20, 30, \dots, 100\}$. These two groups differ in the size of the interval between release time and deadline of the activities and with it their difficulty.

Activities are generated treatment-wise, i.e., by considering sequences of five activities at a time. The particle beam resource is required by the middle (third) activity. Activities two to four demand one of the room resources selected uniformly at random. We assume that $\lceil \alpha/10 \rceil$ radio oncologists are available and select one of them for the first and last activity. Moreover, 25 % of the treatments are assumed to require sedation and are therefore associated with the anesthetist resource. We add for each consecutive activity in the treatment sequence a minimum and maximum time lag. Hence, the resulting precedence graph consists of connected components, each being a path of length five. In the following we refer to these paths, which are essentially equivalent to the treatments, also as *chains*. The processing times of the activities are randomly chosen from the set $\{100, \dots, 10000\}$. Minimum lags are always 100 and maximum lags are always 10000.

It remains to set the release times and deadlines of the activities and the resources' availability windows in such a way that the resulting benchmark instances are feasible with high probability but not trivial. For this reason a preliminary naïve schedule is generated from which release times and deadlines are derived. To this end, the activities are placed treatment-wise in the tentative time horizon $\{0, \dots, \sum_{a \in A}(p_a + 10000)\}$ by randomly selecting a starting time for the first activity of each connected component. For the subsequent activities a random time lag in $\{L^{\min}_{a,a'}, \dots, L^{\max}_{a,a'}\}$ is enforced. If a determined starting time of an activity conflicts with an already scheduled one, the connected component is reconsidered. From this preliminary schedule we derive tentative release times and deadlines which are then scaled to receive a challenging instance. We consider two variants to generate a group of "easy" and a group of "hard" instances. The latter features larger release time deadline windows that make the respective instances more challenging. For details on the used scaling factors see Jatschka [94].

Finally, the availability of the resources is restricted. Each resource has five to seven time windows during which it is unavailable. The duration of these time windows is randomly chosen from the set $\{700, \dots, 1500\}$. The positions of these unavailability windows are chosen uniformly at random from the set $\{0, \dots, T^{\max}\}$.

To our best knowledge benchmark instances considering a comparable scenario do not exist. Our newly introduced test instances are made available at http://www.ac.tuwien.ac.at/research/problem-instances. An overview of the basic characteristics of the test instances is provided in Table 5.1. Instance sets are named according to $[e|h]_\alpha$ for the "easy" (e) and "hard" (h) instances with α indicating the considered number of activities. Each instance set consists of 15 instances.

Set	$T^{\max}$	ρ	Chains	Set	$T^{\max}$	ρ	Chains
e_{20}	104 649	7	4	h_{20}	104 575	7	4
e_{30}	138 808	8	6	h_{30}	137 745	8	6
e_{40}	169 642	9	8	h_{40}	167 003	9	8
e_{50}	198 386	10	10	h_{50}	201 269	10	10
e_{60}	220 792	11	12	h_{60}	220 606	11	12
e_{70}	244 279	12	14	h_{70}	244 788	12	14
e_{80}	271 461	13	16	h_{80}	271 327	13	16
e_{90}	293 110	14	18	h_{90}	289 278	14	18
e_{100}	316 316	15	20	h_{100}	317 324	15	20

Table 5.1: Characteristics of the test instances grouped by difficulty and number of activities. The subscripts indicate the number of activities per instance. $T^{\max}$ denotes the average scheduling horizon. The number of resources ρ and the number of chains is the same for all instance of a set.

5.8.2 Computational Experiments

The test runs have been executed on an Intel Xeon E5540 with 2.53 GHz using a time limit of 7200 seconds and a memory limit of 4 GB RAM. MILP models have been solved using Gurobi 7 with a single thread. We used irace in version 2.1 for parameter tuning, see [116].

The results of the test instances are grouped by difficulty and number of activities. Unless otherwise indicated, computation times are stated using the median, for all other properties we use the mean. Let *pb* denote the primal bound and *db* the dual bound of the investigated algorithm. The starred versions denote the respective best bounds obtained across all algorithms. Optimality gaps are computed by $100 \cdot (pb - db^*)db^*$. Primal bounds are compared using $100 \cdot (pb - pb^*)pb^*$ and dual bounds are compared using $100 \cdot (db^* - db)db^*$.

We first deal with the parametrization of the primal heuristics used within ITBRA. Then, we compare different combinations of refinement strategies for use within the matheuristic. Finally, we compare ITBRA to a simple metaheuristic and the reference MILP models.

Parametrization of the Primal Heuristics

The GRASP from Section 5.6.2 can also be applied outside the context of the matheuristic, thus, as standalone algorithm for SI-PTPSP, based on an empty initial schedule. We start by explaining how the involved parameters are set since they serve as basis for deriving appropriate values for use within the matheuristic.

The standalone GRASP terminates if a time limit of two hours is reached. We chose this criterion primarily to match the time limit of the other approaches, a reasonable degree of convergence is usually reached much earlier. Parameter $k_{\mathrm{ABCH}}^{\mathrm{grand}}$ has been set to 8 for all benchmark instances. We applied irace to determine this value. However, it turned out that the performance of our GRASP is very robust against changes to $k^{\text{grand}}_{\text{ABCH}}$.

For the GRASP embedded in ITBRA we imposed a time limit of 300 seconds and a maximal number of 10000 iterations without improvement as termination criteria. The latter is set high enough to be non-restrictive in most cases but avoid wasting time if the algorithm already converged sufficiently. The values of the parameters $k^{\text{grand}}_{\text{GCH}}$ and $k^{\text{grand}}_{\text{ABCH}}$ of the embedded GRASP have been determined experimentally starting with the values from the standalone variant. For the parameter $k^{\text{grand}}_{\text{GCH}}$ we first assumed a value of $k^{\text{grand}}_{\text{GCH}} = 5 \cdot k^{\text{grand}}_{\text{ABCH}}$ as all activity chains in the test instances consist of five activities. Afterwards, we fine-tuned these parameters by iterative adjustment. The parameter $k^{\text{grand}}_{\text{ABCH}}$ is set to 6 and $k^{\text{grand}}_{\text{GCH}}$ is set to 35. The randomization itself is based on a fixed seed. Tests showed that the chosen termination criteria provide a reasonable balance between result quality and execution speed. Objective values obtained from the embedded GRASP are on average only 0.21 % larger tan those obtained from the standalone variant. The embedded GRASP provides on average solutions with 16.7 % smaller objective value than ABCH.

The local search uses a best improvement strategy. Preliminary experiments confirmed that this strategy works slightly better than a first improvement strategy since the aggregation in terms of activity blocks typically results in only few moves with improvement potential. For the same reason the local optimum is usually reached after a few iterations. Thus, the overhead of the best improvement strategy is not that large. The solutions obtained by the best improvement strategy, however, turned out to pay off in terms of a better average quality. Tests with irace confirmed this observation, although the differences are quite small. However, for instances with different properties this might not be the case. For a larger number of activity blocks a first improvement strategy might be superior.

Comparison of Bucket Refinement Strategies

Due to the large number of possible combinations of refinement techniques (including further ones not presented in this work) we did not test every variant. Instead, we employed a local search strategy to identify good options. Experiments with the matheuristic terminate if optimality is proven or the time limit of two hours is reached.

We started with variant ASEL,B,UR and then step by step investigated the impact of exchanging each of the three components, making use of statistical tests. It turned out that the best refinement strategies are VDUE,B,CPR and VDUE,SET+B,CPR. In addition to the these variants we also consider ASEL,B,UR and the best strategy based on AIGS (AIGS,SET,CPR) in the following. The former mainly serves as naïve reference strategy. The latter is used to discuss certain particularities of the bucket refinement process. We shortly summarize the made observations here and refer to [94] for a more detailed discussion.

	ASEL B UR			AIGS SET CPR			VDUE B CPR			VDUE SET+B CPR		
Set	Gap [%]	Opt	$t[s]$	Gap [%]	Opt	$t[s]$	Gap [%]	Opt	$t[s]$	Gap [%]	Opt	$t[s]$
e_{20}	**0.0**	**15**	**12**	**0.0**	**15**	13	**0.0**	**15**	27	**0.0**	**15**	13
e_{30}	0.6	14	219	0.6	14	**36**	**0.0**	**15**	91	0.6	14	66
e_{40}	5.3	**10**	836	**4.3**	**10**	268	4.9	**10**	200	5.6	**10**	**160**
e_{50}	1.1	14	189	1.1	14	220	**0.0**	**15**	183	**0.0**	**15**	**105**
e_{60}	**1.2**	**14**	82	**1.2**	**14**	**54**	**1.2**	**14**	100	**1.2**	**14**	78
e_{70}	2.1	8	6957	1.2	11	2664	0.4	13	742	**0.3**	**14**	**528**
e_{80}	2.0	7	TL	1.7	9	1687	0.8	10	1197	**0.4**	**13**	**266**
e_{90}	1.7	6	TL	1.6	8	4090	1.6	6	TL	**1.5**	**9**	**1908**
e_{100}	0.9	5	TL	1.3	6	TL	**0.7**	7	TL	1.2	**8**	**4740**
summary	1.7	93	836	1.4	101	268	**1.1**	105	200	1.2	**112**	**160**
h_{20}	**0.0**	**15**	17	**0.0**	**15**	**10**	**0.0**	**15**	22	**0.0**	**15**	20
h_{30}	11.4	9	4341	8.0	10	1726	**6.4**	**12**	1860	**6.4**	**12**	**985**
h_{40}	14.4	4	TL	10.7	6	TL	9.5	6	TL	**8.9**	**7**	TL
h_{50}	18.3	2	TL	18.7	3	TL	**17.0**	**4**	TL	18.2	**4**	TL
h_{60}	18.0	1	TL	17.5	3	TL	17.6	3	TL	**16.4**	**4**	TL
h_{70}	21.9	0	TL	21.0	0	TL	21.2	0	TL	**20.8**	**3**	TL
h_{80}	13.0	1	TL	12.9	1	TL	13.0	1	TL	**12.6**	**2**	TL
h_{90}	11.1	**1**	TL	10.9	**1**	TL	**10.4**	**1**	TL	10.6	**1**	TL
h_{100}	**10.6**	**0**	TL	**10.6**	**0**	TL	**10.6**	**0**	TL	**10.6**	**0**	TL
summary	13.2	33	TL	12.3	39	TL	11.7	42	TL	**11.6**	**48**	TL

Table 5.2: Comparison of selected bucket refinement strategies. We consider the average optimality gaps (gap), the number of solved instances (opt) and the median computation times (t). Entries marked with "TL" indicate that the experiment terminated due to the time limit. The summary is obtained by aggregating over the preceding rows using the same function as for the respective column.

We compared the four strategies in a pairwise fashion checking the assumption that one strategy yields smaller gaps than the other by a one-tailed Wilcoxon signed-rank test with a significance level of 0.05 per difficulty setting and in total. All considered algorithms perform significantly better than the reference strategy on both instance groups and also in total. The VDUE algorithms outperform AIGS on the easy set of instances and in total. However, VDUE,B,CPR is not significantly better than the AIGS variant on the hard set of instances. The VDUE variants perform quite similar and none can be shown to work significantly better than the other.

Table 5.2 provides the results of the selected matheuristic variants. VDUE,SET+B,CPR is clearly the dominant strategy when taking computation times into account but is closely followed by VDUE,B,CPR. To discuss the results in depth we present more specific characteristics of the matheuristic variants in Table 5.3. In particular, we consider the increase in the number of buckets and the average computation time spent per iteration. The former is considered as ratio between the final and the initial number of buckets. Thereby a higher ratio indicates that more buckets were needed to solve the instance.

Reference strategy ASEL,B,UR generates significantly more buckets than the remaining approaches. This typically keeps the number of iterations low. However, this is paid for

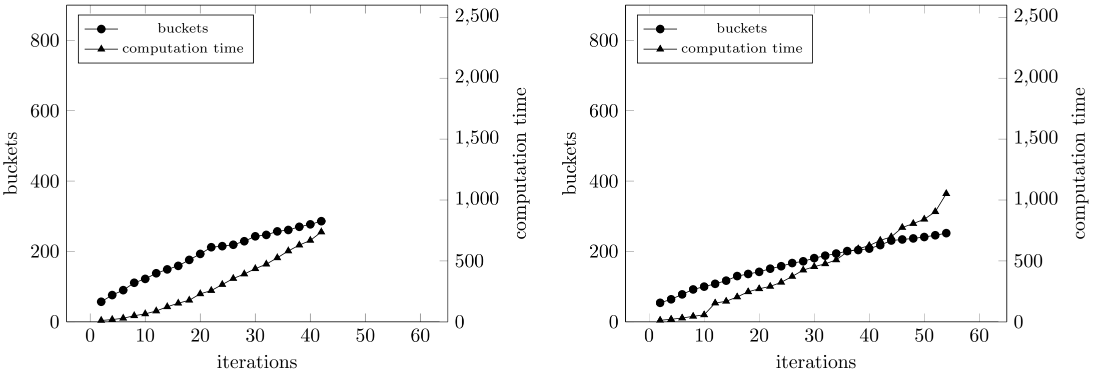
		ASEL B UR			AIGS SET CPR			VDUE B CPR			VDUE SET+B CPR		
Set	$\lvert B^{\text{init}}\rvert$	Ratio$^{\text{B}}$	n^{it}	$t^{\text{it}}[s]$	Ratio$^{\text{B}}$	n^{it}	$t^{\text{it}}[s]$	Ratio$^{\text{B}}$	n^{it}	$t^{\text{it}}[s]$	Ratio$^{\text{B}}$	n^{it}	$t^{\text{it}}[s]$
e_{20}	43	4.73	9	5	2.73	**5**	**2**	1.93	16	**2**	**1.69**	9	**2**
e_{30}	44	7.30	**9**	85	5.77	**9**	62	**2.89**	24	**17**	2.91	16	22
e_{40}	47	7.14	**7**	454	5.75	**7**	330	2.99	19	**158**	**2.91**	13	191
e_{50}	45	8.53	**7**	152	10.13	9	147	**2.83**	15	**25**	2.87	12	33
e_{60}	49	5.30	**4**	218	5.09	5	189	2.50	11	**72**	**2.30**	6	81
e_{70}	49	10.18	**6**	573	8.25	7	395	3.59	18	**89**	**3.52**	12	103
e_{80}	52	7.45	**4**	629	7.01	5	380	3.35	13	131	**3.23**	9	**118**
e_{90}	52	6.94	**3**	809	6.08	4	565	3.54	11	270	**3.53**	8	**235**
e_{100}	58	7.69	**3**	951	6.62	4	708	3.99	13	**266**	**3.75**	9	312
summary	48	7.25	**6**	431	6.38	**6**	309	3.07	16	**114**	**2.97**	10	122
h_{20}	43	4.25	**8**	8	3.73	**8**	6	**1.93**	17	**3**	2.00	13	**3**
h_{30}	44	6.84	**9**	418	6.13	**9**	295	**3.08**	23	121	3.37	19	**109**
h_{40}	43	6.98	**6**	924	6.63	8	621	3.39	17	**276**	**3.29**	12	313
h_{50}	48	5.27	**3**	1812	4.90	5	1160	**2.93**	11	**743**	3.06	9	820
h_{60}	44	5.49	**2**	1926	6.22	5	1166	**3.50**	11	**803**	3.59	9	855
h_{70}	48	4.86	**1**	2629	3.96	3	2611	**3.08**	7	**1280**	3.17	6	1332
h_{80}	49	4.94	**1**	2362	4.97	3	1489	**3.01**	7	**1043**	3.15	5	1112
h_{90}	54	4.97	**1**	2239	4.61	3	1538	**3.04**	6	**1017**	3.22	5	1161
h_{100}	55	4.96	**1**	2617	4.79	3	1648	**2.86**	4	**1287**	3.02	4	1430
summary	48	5.40	**4**	1659	5.10	5	1170	**2.98**	11	**730**	3.10	9	793

Table 5.3: Comparison of the characteristics of selected bucket refinement strategies. We consider the ratio between the number of buckets at the start and at the end of the algorithm (*ratio*$^{\text{B}}$), the average number of iterations (n^{it}), and the average computation time spent per iteration (t^{it}). Column $|B^{\text{init}}|$ provides the average number of buckets contained in the initial bucket partitioning. The summary is obtained by aggregating over the preceding rows using the same function as for the respective column.

excessively in terms of higher computation times per iteration due to the fast increase in model size. In general, the number of buckets grows too fast and unguided to obtain a successful approach.

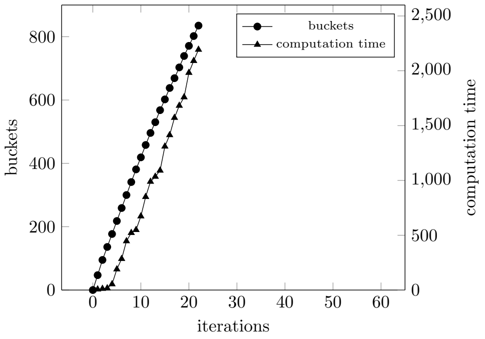
One could expect AIGS to require the fewest buckets among the introduced strategies due to the potentially small number of refinement candidates. However, using only buckets related to activities scheduled by GCH turned out to be too restrictive. This strategy causes some important splits to be delayed until the bucket partitioning is rather fine-grained.

VDUE is again a strategy that can be expected to generate only few new buckets per iteration. However, compared to AIGS their choice appears to be much more meaningful. Nevertheless, splitting only few buckets leads to a high number of iterations. Fortunately, this is not too problematic due to the small computation times per iteration. Identifying splitting positions with the pure binary strategy leads to only few bucket splits which proves to be beneficial. As SET+B typically selects more candidates, one could expect this strategy to be inferior. However, this is compensated for by incorporating more information obtained from the TBR solution.



(a) VDUE,SET+B,CPR

(b) VDUE,B,CPR



(c) ASEL,B,UR

Figure 5.5: Comparison of the relation between computation time and increase in the number of buckets for the same e_{40} instance when using different bucket refinement strategies.

In general, it can be observed that the number of applied splits has a strong influence on performance. However, the quality of the bucket refinement is also very important. For an illustration see Figure 5.5. As mentioned before, the large number of buckets generated by ASEL raises the computation time within a few iterations to a problematic level, causing an overall bad performance. VDUE,B,CPR features the smallest increase in buckets but requires more iterations to converge. Here it becomes clearly visible that SET+B excels by incorporating more knowledge for making its decision.

Finally, we also want to discuss the properties of the remaining variants in excerpts. Figure 5.6 shows a comparison of the average number of iterations and the average final number of buckets for a broad selection of refinement strategies on the set of easy instances with 30 activities. A successful approach is typically characterized by being able to solve an instance by refining only relatively few buckets. Variants that generate many buckets within few iterations usually do not work well. Observe that ASEL and the AIGS variants are all located in the upper half of the figure. The superior strategies are situated near the bottom. It is also clearly visible that SET+B allows to solve an

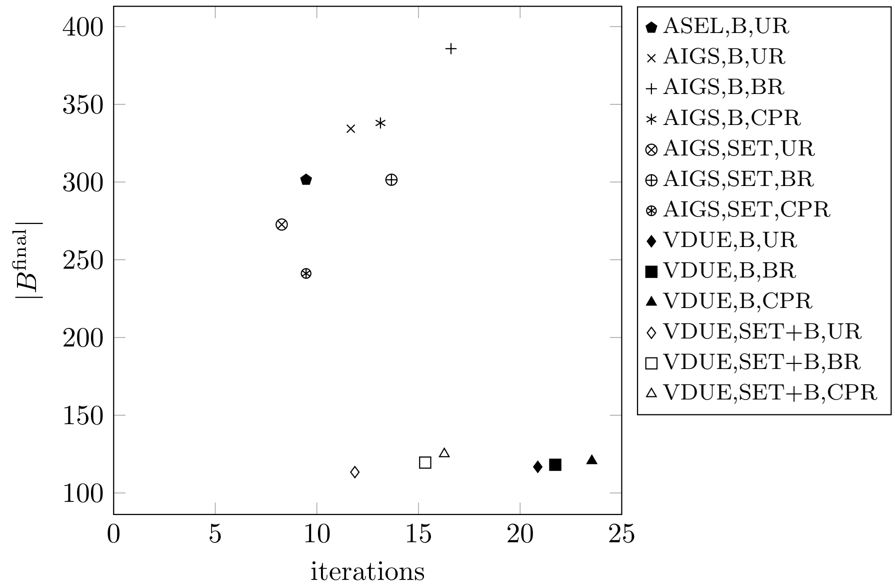


Figure 5.6: Comparison of the average number of iterations and the average final number of buckets on the set of e_{30} instances.

instance in fewer iterations than the pure binary variant. UR and BR are able to solve an instance using fewer buckets and iterations than CPR. This is a peculiarity of the small instances considered here that does not generalize to the larger ones.

Comparing ITBRA to Other Algorithms

We start by comparing the matheuristic to the standalone GRASP, see Table 5.4. ITBRA is in general able to provide better results. However, when dealing with the most difficult instances, it is sometimes the case that the matheuristic only completes very few iterations and GRASP is able to compute a slightly better solution. As the number of activities increases, ITBRA struggles more and more to improve upon the initially obtained primal bound. This is caused by the originally high computation times per iteration that prevent the algorithm from reaching a sufficient degree of convergence. Remember, however, that ITBRA also puts much effort in determining good lower bounds which GRASP cannot provide at all. In the following we investigate the quality of the dual bounds provided by ITBRA in comparison to those obtained by DEF.

DEF was not able to find a primal solution for any instance but at least always computed a dual bound. Table 5.5 provides the comparison with the matheuristic. The bounds obtained from DEF are always worse than those found by ITBRA and turned out to be particularly weak for the group of hard instances, which is a consequence of the looser restrictions featured in this instance group.

	VDUE B CPR			VDUE SET+B CPR			GRASP		
Set	Gap [%]	σ [%]	$t[s]$	Gap [%]	σ [%]	$t[s]$	Gap [%]	σ [%]	Feas
e_{20}	**0.0**	**0.0**	27	**0.0**	**0.0**	**13**	38.6	17.3	**12**
e_{30}	**0.0**	**0.0**	91	0.6	2.1	**66**	28.0	16.0	**14**
e_{40}	**3.5**	**7.4**	200	4.2	**7.4**	**160**	12.6	7.7	**15**
e_{50}	**0.0**	**0.0**	183	**0.0**	**0.0**	**105**	7.8	8.0	**15**
e_{60}	**0.8**	**2.9**	100	**0.8**	**2.9**	**78**	3.4	4.6	**15**
e_{70}	0.4	1.6	742	**0.3**	**1.0**	**528**	3.8	3.9	**15**
e_{80}	0.5	1.1	1197	**0.1**	**0.4**	**266**	2.2	3.7	**15**
e_{90}	0.8	**1.0**	TL	**0.7**	**1.0**	**1908**	0.9	**1.0**	**15**
e_{100}	**0.2**	**0.5**	TL	0.7	1.5	**4740**	1.3	2.1	**15**
summary	**0.7**	**1.6**	200	0.8	1.8	**160**	11.0	7.1	**131**
h_{20}	**0.0**	**0.0**	22	**0.0**	**0.0**	**20**	23.3	12.7	**12**
h_{30}	**5.9**	13.1	1860	**5.9**	13.1	**985**	34.9	**11.9**	**15**
h_{40}	6.5	8.2	TL	**5.9**	**8.1**	TL	21.6	11.6	**15**
h_{50}	**6.6**	7.5	TL	7.6	7.9	TL	10.6	**7.1**	**15**
h_{60}	6.9	5.8	TL	**5.6**	**5.5**	TL	8.4	5.6	**15**
h_{70}	2.7	**3.2**	TL	**2.2**	3.3	TL	4.5	5.9	**15**
h_{80}	1.3	2.9	TL	**0.9**	**2.0**	TL	1.1	2.9	**15**
h_{90}	**2.0**	**3.4**	TL	2.2	4.1	TL	2.6	6.0	**15**
h_{100}	1.0	0.9	TL	1.0	0.9	TL	**0.1**	**0.5**	**15**
summary	3.7	**5.0**	TL	**3.5**	**5.0**	TL	11.9	7.1	**132**

Table 5.4: Comparison of the best found refinement strategies with GRASP. For each algorithm the average gaps to the best primal bound (gap), the standard deviation of the gaps (σ), and the median computation times (t) are presented. Entries marked with "TL" indicate the termination of the experiment due to the time limit. For GRASP, we also provide the number of instances for which a feasible solution could be computed (feas). For the calculation of the gaps we considered only instances for which all algorithms were able to compute a primal bound. The summary is obtained by aggregating over the preceding rows using the same function as for the respective column.

The second MILP-based approach to which we compare ITBRA is TIF. As a result of the extremely large time horizons and the memory restriction of 4 GB RAM, none of the TIF models even fit into the RAM. Therefore, we consider coarsened TIF models by only taking a subset of the original time horizon into account. Let $\kappa \in \mathbb{N}_{>1}$ be the coarsening measure and TIF^{κ} the associated model. Then, the new time horizon T^{κ} of TIF^{κ} is defined as $T^{\kappa} = \{t \in T \mid t \equiv 0 \pmod{\kappa}\}$. Consequently, we obtain reduced sets of feasible starting times $T_a^{\kappa} = T_a \cap T^{\kappa}$ for the activities $a \in A$. Reducing the number of considered time slots decreases the size of the model, which leads to faster computation times. However, an optimal solution to TIF^{κ} is in general not optimal with respect to the original problem due to the disregarded time slots, making it a heuristic approach. A coarsened model might even become infeasible when discarding too many time slots.

Table 5.6 provides the results of the differently coarsened TIF models. We increase the value of κ stepwise until all instances can either be solved within the time limit or do

	VDUE B CPR			VDUE SET+B CPR			DEF		
Set	Gap [%]	σ [%]	$t[s]$	Gap [%]	σ [%]	$t[s]$	Gap [%]	σ [%]	$t[s]$
e_{20}	**0.0**	**0.0**	27	**0.0**	**0.0**	**13**	15.3	10.6	TL
e_{30}	**0.0**	**0.0**	91	**0.0**	0.1	**66**	7.6	6.5	TL
e_{40}	**0.2**	**0.4**	200	**0.2**	0.5	**160**	4.5	7.8	TL
e_{50}	**0.0**	**0.0**	183	**0.0**	**0.0**	**105**	3.0	6.4	TL
e_{60}	**0.0**	**0.0**	100	**0.0**	**0.0**	**78**	1.8	3.2	TL
e_{70}	**0.0**	**0.0**	742	**0.0**	**0.0**	**528**	1.6	2.4	TL
e_{80}	**0.0**	**0.0**	1197	**0.0**	0.1	**266**	1.0	1.7	TL
e_{90}	**0.0**	**0.1**	TL	0.1	0.2	**1908**	1.5	2.8	TL
e_{100}	**0.0**	**0.1**	TL	0.1	**0.1**	**4740**	2.0	1.8	TL
summary	**0.0**	**0.1**	200	**0.0**	**0.1**	**160**	4.3	4.8	TL
h_{20}	**0.0**	**0.0**	22	**0.0**	**0.0**	**20**	19.6	11.4	TL
h_{30}	**0.3**	**1.1**	1860	0.4	**1.1**	**985**	31.3	12.2	TL
h_{40}	0.2	0.4	TL	**0.1**	**0.2**	TL	18.6	13.3	TL
h_{50}	2.0	3.3	TL	**1.6**	**1.5**	TL	15.4	9.0	TL
h_{60}	**0.9**	**1.7**	TL	1.3	2.1	TL	4.3	4.8	TL
h_{70}	**2.8**	4.0	TL	2.9	**3.9**	TL	6.7	7.4	TL
h_{80}	1.5	**4.8**	TL	**1.4**	4.9	TL	2.3	4.9	TL
h_{90}	**0.3**	0.7	TL	**0.3**	**0.6**	TL	3.8	5.9	TL
h_{100}	**0.3**	**0.4**	TL	**0.3**	**0.4**	TL	1.0	1.3	TL
summary	**0.9**	1.8	TL	**0.9**	**1.6**	TL	11.4	7.8	TL

Table 5.5: Comparison between ITBRA and DEF. For each algorithm we provide the average gaps to the best dual bound (gap), the standard deviation of the gaps (σ) and the median computation times (t). Entries marked with "TL" indicate the termination of the experiment due to the time limit. The summary is obtained by aggregating over the preceding rows using the same function as for the respective column.

not permit feasible solutions anymore. For $\kappa < 100$ the models fail to generate a primal bound for almost all instances due to the memory or time limitations. Missing table entries (marked with "-") indicate that the coarsened model is not able to find a primal bound for any instance of the corresponding set. For smaller instances the TIF^{κ} models are able to produce reasonable primal solutions. However, the quality of the solutions deteriorates drastically as more time slots are disregarded. No TIF^{κ} variant is able to find a primal solution for all instances. When using a small value for κ, many instances cannot be solved due to the time limit. For larger κ-values we can solve more instances but at the expense of considerably larger gaps. Moreover, as we reduce the precision even further, the models start to become infeasible. The number of infeasible instances strongly increases for $\kappa \geq 10000$ and the few instances that still permit feasible solutions feature gaps of over 120 %. Therefore, further increasing the value of κ does not seem meaningful. We draw the conclusion that there does not exist an appropriate value for κ allowing a reasonable balance between computation time and result quality. Due to the many missing entries we decided to use median instead of average gaps in the summary table.

According to our experiments the best variants are those with $\kappa = 1000$ and $\kappa = 2000$, respectively. The former provides better solutions but the latter is able to find more feasible solutions. For some instances the coarsened TIF variants even find better primal solutions than the matheuristic. Especially for instance sets h_{40}, h_{50}, and h_{60} we obtain a high number of good solutions such that also the median gaps are smaller here. Overall, however, ITBRA still provides the better results. Moreover, recall that the TIF^{κ} models can only provide heuristic solutions and no dual bounds.

Last but not least, we also investigated the use of disaggregated precedence constraints (see Artigues [2]) but this did not lead to significant improvements.

5.9 Conclusion

In this chapter we considered a matheuristic, referred to as ITBRA, intended to solve an RCPSP that requires scheduling in high resolution. We proposed a relaxation for the original problem based on aggregating consecutive integral time points into so-called time-buckets. Exploiting this relaxation we constructed a matheuristic that solves this relaxation based on iteratively refined bucket partitionings. Moreover, we heuristically derive primal bounds incorporating information from the relaxed solution. The matheuristic then attempts to close the gap between dual bounds obtained from the relaxation and primal bounds determined by (meta-)heuristics. The crucial part of this approach is how to determine the (more refined) bucket partitioning for the next iteration. We considered a variety of strategies and compared them on a novel benchmark set motivated by an application arising in particle therapy for cancer treatment.

Our experiments indicate that it is most critical to limit the increase in the number of buckets. However, the quality of the applied bucket splits has a substantial impact on the convergence speed. Strategy VDUE,SET+B,CPR turned out to work best in this respect.

The matheuristic works better than a simple GRASP on all instance sets except for the most difficult one. There it fails to complete a sufficient number of iterations to make reasonable improvements to the primal bound.

ITBRA clearly outperforms the compact MILP formulations. The considered DEF provides dual bounds for all investigated benchmark instances but no primal solutions. In case of the considered TIF, on the other hand, not even the LP relaxation can be solved due to its excessive model size. Variants of TIF based on a coarsened time horizon are manageable but become infeasible once too many time points are disregarded. For some instances good primal solutions could be obtained but there exists no coarsening factor that works well in general by providing a good balance between model size and result quality.

We primarily focused on MILP-based algorithms here. Another well-known exact technique often used to deal with scheduling problems is CP. In a more comprehensive study

	TIF^{100}		TIF^{200}		TIF^{1000}		TIF^{2000}		TIF^{10000}		VDUE B CPR		VDUE SET+B CPR	
Set	Gap$^{\text{med}}$ [%]	$t[s]$	Gap$^{\text{med}}$ [%]	$t[s]$	Gap$^{\text{med}}$ [%]	$t[s]$	Gap$^{\text{med}}$ [%]	$t[s]$	Gap$^{\text{med}}$ [%]	$t[s]$	Gap$^{\text{med}}$ [%]	$t[s]$	Gap$^{\text{med}}$ [%]	$t[s]$
e_{20}	0.3	15	0.6	4	21.7	**<1**	24.0	**<1**	-	-	**0.0**	27	**0.0**	13
e_{30}	0.1	310	0.4	58	4.0	4	18.3	**2**	-	-	**0.0**	91	**0.0**	66
e_{40}	0.3	2940	0.5	698	3.8	40	12.3	**6**	-	-	**0.0**	200	**0.0**	160
e_{50}	0.2	409	0.4	113	1.9	17	6.4	5	-	**<1**	**0.0**	183	**0.0**	105
e_{60}	0.3	5569	0.4	839	2.4	52	4.7	21	-	**<1**	**0.0**	100	**0.0**	78
e_{70}	14.3	TL	0.3	2713	1.9	165	4.7	77	-	**<1**	**0.0**	742	**0.0**	528
e_{80}	-	TL	0.3	5630	1.7	292	3.4	108	-	**<1**	**0.0**	1197	**0.0**	266
e_{90}	-	TL	-	TL	0.9	555	3.0	327	-	**1**	**0.0**	TL	**0.0**	1908
e_{100}	-	TL	-	TL	1.8	652	4.0	263	-	**6**	**0.0**	TL	**0.0**	4740
summary	0.3	5569	0.4	839	1.9	52	4.7	21	-	**<1**	**0.0**	200	**0.0**	160
h_{20}	0.4	39	0.5	8	6.4	1	19.8	**<1**	-	**<1**	**0.0**	22	**0.0**	20
h_{30}	11.8	6106	1.0	1129	11.1	42	24.6	**13**	-	-	**0.0**	1860	**0.0**	985
h_{40}	35.4	TL	**0.6**	TL	5.5	227	13.7	65	-	**<1**	3.9	TL	3.2	TL
h_{50}	-	TL	-	TL	**0.0**	2815	9.5	381	-	**<1**	3.5	TL	3.9	TL
h_{60}	-	TL	8.9	TL	**2.3**	1532	7.0	940	-	**<1**	9.2	TL	6.5	TL
h_{70}	-	TL	-	TL	14.3	TL	10.4	3052	82.5	**<1**	1.6	TL	**0.0**	TL
h_{80}	-	TL	-	TL	5.0	TL	12.1	TL	77.0	**3**	**0.0**	TL	**0.0**	TL
h_{90}	-	TL	-	TL	9.0	TL	14.2	TL	93.8	**8**	**0.3**	TL	**0.3**	TL
h_{100}	-	TL	-	TL	39.6	TL	22.7	TL	-	**16**	**1.2**	TL	**1.2**	TL
summary	-	TL	-	TL	6.4	2815	13.7	940	-	**<1**	1.2	TL	**0.3**	TL

	TIF^{100}					*TIF*^{200}				*TIF*^{1000}				*TIF*^{2000}				*TIF*^{10000}			
Set	Opt^{c}	Feas	Infeas	TL	ML	Opt^{c}	Feas	Infeas	TL	Opt^{c}	Feas	Infeas	TL	Opt^{c}	Feas	Infeas	TL	Opt^{c}	Feas	Infeas	TL
e_{20}	**15**	**15**	**0**	**0**	**0**	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	13	13	2	**0**	0	0	15	**0**
e_{30}	**15**	**15**	**0**	**0**	**0**	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	0	0	15	**0**
e_{40}	9	12	**0**	6	**0**	12	14	**0**	3	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	0	0	15	**0**
e_{50}	13	14	**0**	2	**0**	14	14	**0**	1	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	2	2	13	**0**
e_{60}	9	9	**0**	6	**0**	14	**15**	**0**	1	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	6	6	9	**0**
e_{70}	6	9	**0**	9	**0**	12	14	**0**	3	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	2	2	13	**0**
e_{80}	3	3	**0**	12	**0**	9	11	**0**	6	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	7	7	8	**0**
e_{90}	0	0	**0**	13	**2**	6	7	**0**	9	**14**	14	**0**	1	**14**	**15**	**0**	1	4	4	11	**0**
e_{100}	1	1	**0**	8	**6**	4	5	**0**	11	13	**15**	**0**	2	**15**	**15**	**0**	**0**	2	2	13	**0**
summary	71	78	**0**	56	**8**	101	110	**0**	34	**132**	**134**	**0**	3	**132**	133	2	1	23	23	112	**0**
h_{20}	**15**	**15**	**0**	**0**	**0**	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	2	2	13	**0**
h_{30}	8	12	**0**	7	**0**	14	**15**	**0**	1	**15**	**15**	**0**	**0**	14	14	1	**0**	0	0	15	**0**
h_{40}	4	9	**0**	11	**0**	8	12	**0**	7	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	5	5	10	**0**
h_{50}	1	4	**0**	14	**0**	4	7	**0**	11	12	**15**	**0**	3	**15**	**15**	**0**	**0**	6	6	9	**0**
h_{60}	0	0	**0**	15	**0**	2	9	**0**	13	9	**15**	**0**	6	**14**	**15**	**0**	1	6	6	9	**0**
h_{70}	0	0	**0**	15	**0**	0	1	**0**	15	4	10	**0**	11	**10**	**15**	**0**	5	8	8	7	**0**
h_{80}	0	0	**0**	11	**4**	1	4	**0**	14	5	11	**0**	10	4	**15**	**0**	11	**9**	9	6	**0**
h_{90}	0	0	**0**	12	**3**	1	1	**0**	14	4	**12**	**0**	11	6	**12**	**0**	9	**8**	8	7	**0**
h_{100}	0	0	**0**	1	**14**	0	0	**0**	15	1	8	**0**	14	0	**11**	**0**	15	**5**	5	10	**0**
summary	28	40	**0**	86	**21**	45	64	**0**	90	80	116	**0**	55	**93**	**127**	1	41	49	49	86	**0**

Table 5.6: Comparison of differently coarsened TIF models with ITBRA. We provide the median gaps to the best primal bound of the original problem (gap) and the median computation times (t). Missing entries ("-") indicate that the coarsened model is not able to find a primal bound for any instance of the corresponding set. Moreover, for each instance set we indicate the number of optimally (opt^{c}) and feasibly (feas) solved instances with respect to the coarsened model. Column *infeas* denotes the number of instances with proven infeasible model. Finally, we indicate the number of instances that terminated due to the time limit (TL) or the memory limit (ML), respectively. The summary is obtained by aggregating over the preceding rows using the same function as for the respective column.

it appears to be interesting to compare ITBRA to a suitable CP approach. Moreover, it might also be relevant to consider CP techniques within our matheuristic to improve its performance. In general the (meta-)heuristics currently used within ITBRA are rather simple. In particular, they suffer from the effects of fixing the time lags which prevents them from considering a large variety of possible solutions. This is a crucial part of the matheuristic for which more elaborated techniques should be identified and tested.

In the computational study we investigated the power of our algorithm on a rather specific set of benchmark instances. The fundamental approach, however, is in principle much more generally applicable to problems that require scheduling in high resolution. To verify this a more diversified set of benchmark instances, originating from different application areas, has to be considered. Of course this requires adjusted MILP formulations and adapted as well as novel bucket refinement strategies.

CHAPTER 6

Exact Approaches for the Directed Network Design Problem with Relays

In Chapter 3 we considered a column generation approach in conjunction with a preceding graph transformation for the network design problem with relays (NDPR). Here we consider its directed counterpart, the directed network design problem with relays (DNDPR). Similar to the NDPR, the DNDPR seeks to enable communication of a given set of commodity pairs within a network. To react to signal degradation, relays have to be placed before exceeding a certain distance limit. Apart from being based on a directed network, the main difference to the NDPR lies in the fact that commodities now have to be connected by simple paths instead of walks. The necessity of preventing repeated node visits makes the approach from Chapter 3 less appealing since detecting overlaps in the priced paths incurs substantial overhead. Therefore, we pursue a different modeling approach based on so-called *layered graphs (LGs).*

The idea of LGs is to extend a base graph along one or multiple dimension to make it easier to formulate certain constraints. In terms of the DNDPR this is done by creating for each node copies with respect to the distances at which it might be feasibly reached. Node copies beyond the distance limit are omitted. The placement of relays is modeled through additional arcs that connect a node copy at a higher layer to its copy at layer zero. Therefore, a path in such an LG implicitly meets the distance restrictions and specifies the nodes at which relays have to be placed. Visiting nodes more than once can be prevented by restricting the total in-degree of all copies of each node. Through the introduced node copies and the associated arcs, the LG is often substantially larger than the input graph. Whether such an approach may succeed, strongly depends on the number of node copies and the density of the input graph. Fractional distances impose an additional difficulty in this respect. To handle this case more efficiently we first round down the fractional values to restrict the problem to the integers. Potential infeasibilities with respect to the original problem that might be present in the resulting relaxation are addressed by cutting planes. In the following we investigate the effectiveness of our approach on diverse sets of benchmark instances.

This chapter is currently submitted to *Omega: The International Journal of Management Science*[1]:

> M. Leitner, I. Ljubić, M. Riedler, and M. Ruthmair. Exact approaches for the directed network design problem with relays. *Omega*, submitted

A preliminary version was presented at the *21st Conference of the International Federation of Operational Research Societies*[2] in Québec, Canada.

6.1 Introduction

The DNDPR was introduced by Li et al. [110] for modeling the design of networks when the maximum distance a commodity (i.e., signal) can travel is bounded from above by some threshold. This distance limit can be surpassed by locating special, commodity regenerating equipment (relays) at intermediate network nodes. Applications of this problem arise in the design of transportation and telecommunication networks [110]. In the latter, signals deteriorate after traveling a certain distance and thus there is the need to regenerate them before a predefined maximum distance is exceeded. Thus, comparably expensive regenerating devices (e.g., repeaters) need to be installed, see, e.g., Cabral et al. [30], Chen et al. [35], Yıldız and Karaşan [175], in order to avoid signal loss or falsification of the transmitted information.

In the design of optical telecommunication networks, for example, commodities correspond to node pairs that need to communicate with each other, but the quality of the optical signal degrades with the distance, so that after a certain distance the signal has to be amplified, which is done by deploying regenerator devices at some nodes of the network [36]. Edge costs are directly proportional to edge lengths (multiplied by some factor that corresponds to cable costs per unit of distance) whereas relay costs correspond to the installation and purchasing costs of regenerator devices. Such devices are usually very expensive (see, e.g., [126] for further details). In the design of fiber optic networks, wavelength division multiplexing (WDM) is used to divide the bandwidth of a single fiber into different wavelength channels so that there is no interference between transmissions on different wavelengths. A signal from a source node to its destination is sent using the wavelength routing through a *lightpath* which is an end-to-end connection over a dedicated communication channel (circuit) that traverses one or more links and uses one WDM channel per link. The circuit guarantees the full bandwidth of the channel and

---

[1] https://www.journals.elsevier.com/omega

[2] http://ifors2017.ca

allows for a data rate of 10 or even 40 gigabits per second, see, e.g., [129] for further details. When deploying regenerators in such a network, the signal is converted from optical to electric and back to optical, each time a regenerator is used in the routing path from a source to its destination (such paths are commonly referred to as *translucent lightpaths)*. When translucent lightpaths are not allowed to contain cycles (which can be due to the signal interference, or due to the fact that each lightpath has to be uniquely defined per source-destination pair), one has to explicitly impose *simple paths* for the wavelength routing.

The DNDPR is defined on a digraph $G = (V, A, c, w, d)$ with relay costs $c\colon V \to \mathbb{Q}_{\geq 0}$, arc costs $w\colon A \to \mathbb{Q}_{\geq 0}$, and arc distances $d\colon A \to \mathbb{Q}_{\geq 0}$. Moreover, a distance bound $\lambda_{\max}$ and a set of commodity pairs $\mathcal{K}$ are given. For each commodity $(u, v) \in \mathcal{K}$, nodes u and v are called its source and target, respectively. The goal of the DNDPR is to place relays on a subset of the nodes $V' \subseteq V$ and to select a subset of arcs $A' \subseteq A$ such that:

1. The subgraph induced by A' contains for each $(u, v) \in \mathcal{K}$ a directed (simple) path from u to v not exceeding the distance limit between u and the first relay, any two consecutive relays, and the last relay and v, and
2. the cost induced by installing relays and arcs, defined as

$$\sum_{v \in V'} c_v + \sum_{a \in A'} w_a$$

   is minimized.

A problem instance and its optimal solution are given in Figure 6.1.

The DNDPR is closely related to the previously introduced and well-studied NDPR, see Chapter 3. The major difference between the two problems is the way how routing paths are defined: whereas only simple paths are allowed in case of the DNDPR, solutions of the NDPR may contain cycles. This latter property renders NDPR solutions infeasible when it comes to the design of translucent optical networks. For an example see Figure 6.2.

To simplify notation, we will in the following use $S = \{u \mid (u, v) \in \mathcal{K}\}$ to denote the set of commodity sources and $T^u = \{v \mid (u, v) \in \mathcal{K}\}$ to denote all targets that need to be reached from source $u \in S$. Additionally, $\delta^-(W) = \{(j, i) \mid i \in W, (j, i) \in A\}$ and $\delta^+(W) = \{(i, j) \mid i \in W, (i, j) \in A\}$ will be used to denote the sets of incoming and outgoing arcs for node sets $W \subset V$. By slightly abusing notation, we write $\delta^-(i)$ and $\delta^+(i)$ instead of $\delta^-(\{i\})$ or $\delta^+(\{i\})$ for singletons $W = \{i\}$.

Related work. The DNDPR has been introduced in Li et al. [110] where a compact *node-arc formulation* and an *arc-path formulation* with an exponential number of variables have been proposed. Two branch-and-price (B&P) algorithms based on the latter

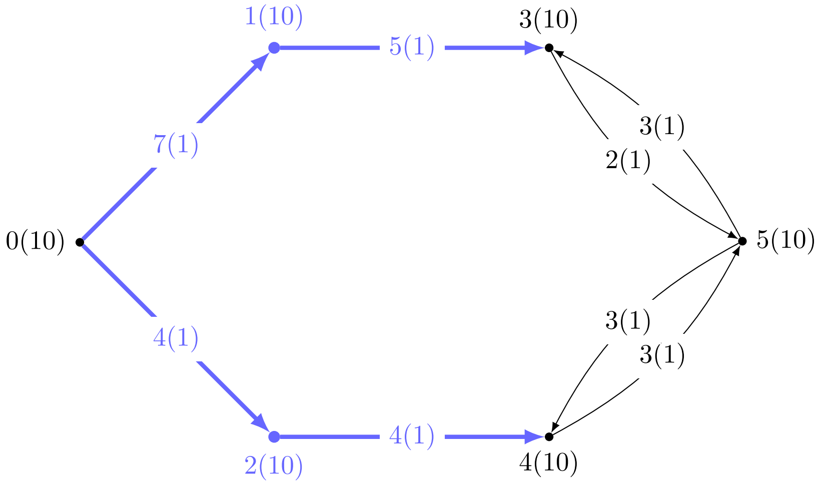


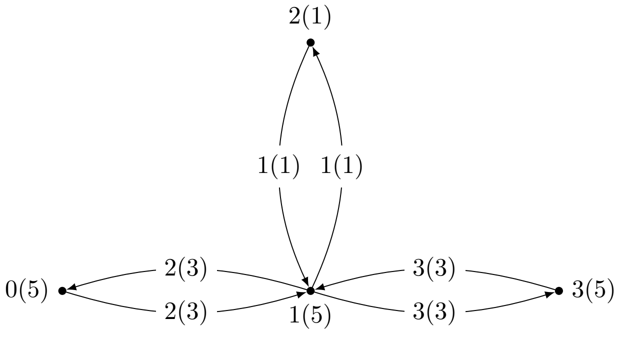
Figure 6.1: Example instance with two commodities $\mathcal{K} = \{(0,3),(0,4)\}$ and $\lambda_{\max} = 7$. Arc distances are provided next to the arcs, relay and arc costs are given in parentheses. Relays and arcs used in the optimal solution are marked bold and blue.

formulation have been developed that differ in the way the pricing subproblem is solved. A metaheuristic based on tabu search has been recently proposed in Li et al. [111].

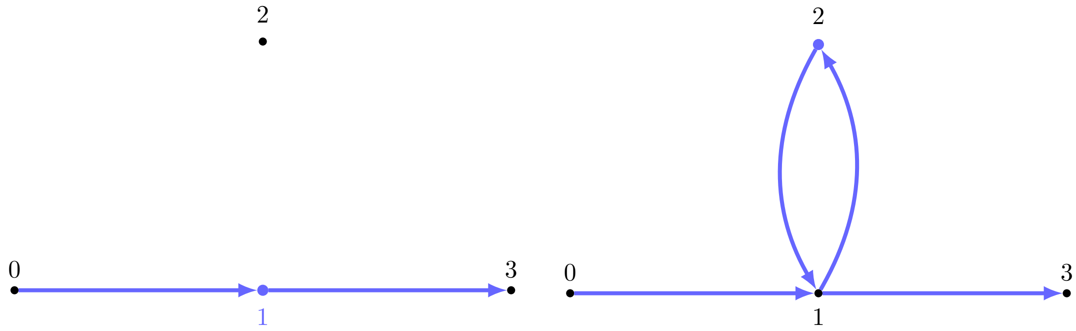
Several related studies consider the undirected variant of the problem, i.e., the NDPR. The NDPR has been introduced in Cabral et al. [30] where the proposed B&P approach turned out to be quite inefficient (even for small instances) due to the high complexity of the associated pricing subproblem. Therefore, Cabral et al. [30] have focused on construction heuristics that were able to tackle larger problem instances in comparably short time. More efficient B&P approaches for the NDPR have been given in in Chapter 3 and Yıldız et al. [177]. In addition to these exact approaches, several metaheuristics have been developed for approximately solving larger problem instances: genetic algorithms (Kulturel-Konak and Konak [102], Konak [100]), tabu search (Lin et al. [113]), and variable neighborhood search (Xiao and Konak [173]).

Existing methods for the NDPR cannot by applied in a straightforward way to the DNDPR, since NDPR solutions may contain cycles (or even traverse a single edge in both directions for one commodity). Besides, asymmetric arc costs and arcs existing in a single direction only would require some adaptations.

Node-arc formulation. The node-arc formulation (6.1) introduced in Li et al. [110] is used for comparison purposes in our computational study. Therefore, we briefly summarize it in the following. Its basic idea is to keep track of the distance from the last relay (or the source of the respective commodity) in order to forbid subpaths exceeding the distance bound. Four sets of variables are used. Binary arc and node variables x_a, $\forall a \in A$, and y_i, $\forall i \in V$, mark the selected arcs and relays, respectively. For each



Instance



Optimal acyclic solution (cost=11)

Optimal cyclic solution (cost=9)

Figure 6.2: Symmetric instance together with an acyclic and a general solution for $\lambda_{\max} = 4$ and $\mathcal{K} = \{(0,3)\}$. Arc distances are provided next to the arcs, relay and arc costs are given in parentheses. Relays and arcs used in the optimal solution are marked bold and blue. Note that in the acyclic solution we place a relay at node 1, while in the cheaper cyclic solution we place a relay at node 2.

commodity $k = (u,v) \in \mathcal{K}$ and node $i \in V$, continuous variable v_i^k tracks the distance of node i to the preceding relay or the source u of that commodity (in case the path from u to i does not contain relays). Finally, multi-commodity flow variables f_a^k, $\forall k \in \mathcal{K}$, $\forall a \in A$, are used to enforce connectivity of each commodity pair. Formulation (NA) reads as follows:

$$\min \sum_{i \in V} c_i y_i + \sum_{a \in A} w_a x_a \tag{6.1a}$$

$$\text{s.t.} \sum_{a \in \delta^+(i)} f_a^k - \sum_{a \in \delta^-(i)} f_a^k = \begin{cases} 1 & \text{if } k = (i,j) \\ -1 & \text{if } k = (j,i) \\ 0 & \text{otherwise} \end{cases} \qquad \forall k \in \mathcal{K}, \forall i \in V, \tag{6.1b}$$

$$f_{ij}^k \leq x_{ij} \qquad \forall k \in \mathcal{K}, \forall (i,j) \in A, \tag{6.1c}$$

$$v_i^k + d_{ij} f_{ij}^k - \lambda_{\max}(1 - f_{ij}^k + y_j) \leq v_j^k \qquad \forall k \in \mathcal{K}, \forall (i,j) \in A, \tag{6.1d}$$

$$v_i^k + d_{ij} f_{ij}^k \leq \lambda_{\max} \qquad \forall k \in \mathcal{K}, \forall (i,j) \in A, \qquad (6.1e)$$
$$0 \leq v_i^k \leq \lambda_{\max}(1 - y_i) \qquad \forall k \in \mathcal{K}, \forall i \in V, \qquad (6.1f)$$
$$v_u^{u,v} = 0 \qquad \forall (u,v) \in \mathcal{K}, \qquad (6.1g)$$
$$f_{ij}^k \in \{0,1\} \qquad \forall k \in \mathcal{K}, \forall (i,j) \in A, \qquad (6.1h)$$
$$y_i \in \{0,1\} \qquad \forall i \in V, \qquad (6.1i)$$
$$0 \leq x_{ij} \leq 1 \qquad \forall (i,j) \in A. \qquad (6.1j)$$

Flow conservation constraints (6.1b) together with linking constraints (6.1c) ensure the existence of a directed path from u to v for each commodity pair $(u,v) \in \mathcal{K}$. Constraints (6.1d) ensure that the value of variable v_j^k is at least the distance from the last relay or from the source, respectively, along the path connecting commodity $k \in \mathcal{K}$. The distance limit is enforced using inequalities (6.1e) and (6.1f). The latter inequalities also link distance and relay variables. Observe that binary (rather than continuous) flow variables are needed to prevent flow splittings which would yield incorrect values of distance variables v. The main weakness of this model is its relatively weak linear programming (LP) relaxation bound, resulting from the (potentially) large coefficient $\lambda_{\max}$ required in constraints (6.1d).

Overview and contributions. Two mixed integer linear programming (MILP) formulations that are based on considering one LG per source are introduced in Section 6.2. The first one is a flow-based formulation with a pseudo-polynomial number of variables and constraints, whereas the second one uses an exponential number of connectivity constraints. Fractional distance values are handled efficiently by augmenting both models with an exponentially-sized set of infeasible path constraints. Subsequently, different families of symmetry breaking constraints and valid inequalities are introduced. Section 6.3 describes components and variants of a branch-and-cut (B&C) algorithm based on the second formulation, introduces preprocessing routines, and details a heuristic used to obtain initial solutions. Benchmark instances used in our study are described in Section 6.4 where we also verify the effectiveness of our algorithms by extensive computational experiments. Finally, implications of our study to the practice of management are highlighted in Section 6.4.6.

6.2 Formulations

Properties of feasible solutions. In the DNDPR, the routing of each individual commodity is done following a simple path, see [110]. Hence, the in-degree of each node in the routing path is at most one. An optimal solution is a union of all routing paths over all commodity pairs, and hence, the in-degree of a node in this solution can be as large as the number of commodities. On the other hand, if all commodities share a common source node, then it is not difficult to see that there always exists an optimal solution in which the in-degree of each node is at most one, i.e., such that the set A' of selected arcs forms an arborescence.

Theorem 6.1. *If $S = \{u\}$, there exists an optimal DNDPR solution which corresponds to a Steiner arborescence rooted at u, whose leaves are a subset of the nodes from T^u.*

Theorem 6.1 enables the interpretation of an optimal solution as a union of several Steiner arborescences (one per source). It does not help, however, to handle the distance constraints and the installation of relays to respect the threshold $\lambda_{\max}$. To deal with these issues, we propose to exploit LGs introduced below.

The new formulations presented in this chapter use extended, so-called LGs. The basic idea of LGs is to introduce multiple copies for each node and arc of an original graph along one or multiple dimensions (e.g., time or distance) to implicitly model certain constraints and to obtain stronger mathematical models. In our case, LGs are used to encode the distances. If all distance values are integral, only feasible paths with respect to the distance bound $\lambda_{\max}$ are generated. On the contrary, paths (slightly) violating the distance bound may be contained in our LGs in the more general case of fractional distance values. These paths will be excluded from solutions via additional inequalities. Picard and Queyranne [137] were among the first to consider LGs and used them for solving the time-dependent traveling salesman problem. More recent successful applications of LGs are, e.g., given in Godinho et al. [75], Gouveia et al. [79, 80, 81], Gouveia and Ruthmair [77], Ljubić and Gollowitzer [114], Ruthmair and Raidl [155]; see Gouveia et al. [82] for a survey on this topic. Using a relaxed LG and handling infeasible paths by cutting planes is similar to the approach used in Dash et al. [48], however, we use a static relaxed LG instead of an iteratively derived one.

For the DNDPR, we construct layered digraph $G_{\mathrm{L}} = (V_{\mathrm{L}}, A_{\mathrm{L}})$ whose node set V_{L} is recursively defined by sets V_{L}^l, $l \in \{0, 1, \dots, \lambda_{\max}\}$. Thereby, $V_{\mathrm{L}} := V_{\mathrm{L}}^{\lambda_{\max}}$ and each subset V_{L}^l contains all nodes that can be reached with a total distance of at most l starting from a node at layer zero, i.e.,

$$\begin{aligned} V_{\mathrm{L}}^0 &= \{i_0 \mid i \in V\} \\ V_{\mathrm{L}}^l &= \{j_l \mid i_m \in V_{\mathrm{L}}^{l-1}, (i,j) \in \delta^+(i), m + \lfloor d(i,j) \rfloor = l\} \cup V_{\mathrm{L}}^{l-1}. \end{aligned}$$

Arc set A_{L} connects layered node copies $i_l, j_m \in V_{\mathrm{L}}$ for which $(i,j) \in A$ and the difference of the layers corresponds to the arc distance rounded down to the nearest integer, i.e., $m - l = \lfloor d(i,j) \rfloor$. Furthermore, arcs (i_l, i_0) are included for each node $i_l \in V_{\mathrm{L}}$ that is not at layer zero, i.e., when $l > 0$. As the latter arcs correspond to using a relay, we will call them *relay arcs*. Formally, arc set $A_{\mathrm{L}} = A_{\mathrm{L}}^{\mathrm{a}} \cup A_{\mathrm{L}}^{\mathrm{r}}$ where $A_{\mathrm{L}}^{\mathrm{r}}$ is the set of relay arcs and $A_{\mathrm{L}}^{\mathrm{a}}$ is the set of arcs derived from the original graph:

$$\begin{aligned} A_{\mathrm{L}}^{\mathrm{r}} &= \{(i_l, i_0) \mid i_l \in V_{\mathrm{L}}, l > 0\} \\ A_{\mathrm{L}}^{\mathrm{a}} &= \{(i_l, j_m) \mid i_l, j_m \in V_{\mathrm{L}}, (i,j) \in A, \lfloor d(i,j) \rfloor = m - l\}. \end{aligned}$$

Figure 6.3 shows the LG corresponding to the instance given in Figure 6.1 as well as the embedding of an optimal solution in the LG. Thereby, relay arcs $A_{\mathrm{L}}^{\mathrm{r}}$ are depicted in dashed lines and the remaining arcs in solid lines. Bold blue arcs indicate those that are included in the considered solution.

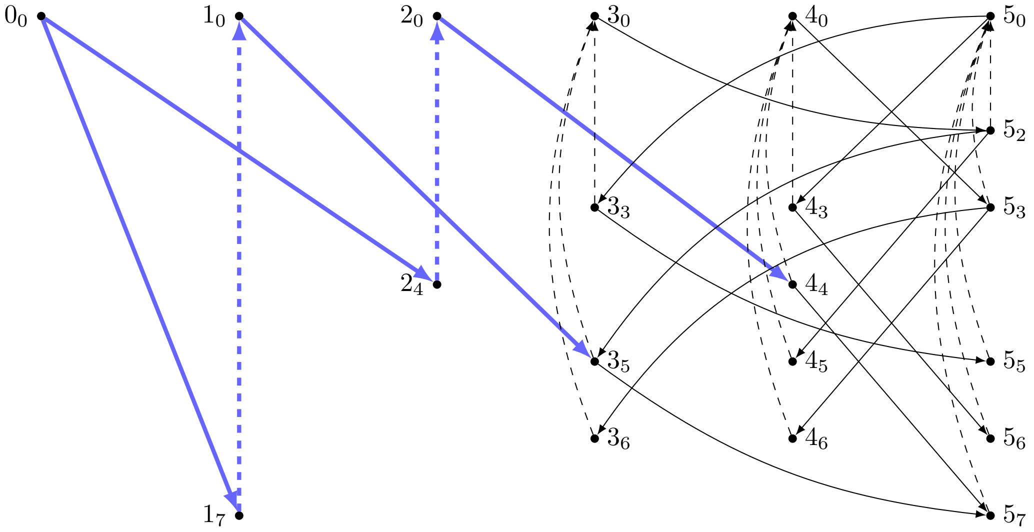


Figure 6.3: Layered graph $G_{\mathrm{L}} = (V_{\mathrm{L}}, A_{\mathrm{L}})$ for $\lambda_{\max} = 7$ corresponding to the instance in Figure 6.1. The optimal solution is marked bold and blue. Dashed lines indicate relay arcs.

6.2.1 Multi-commodity Flow Formulation

The *layered multi-commodity flow formulation* ($\mathrm{L_{MCF}}$) is based on flow variables $f_a^{uv} \geq 0$, $\forall (u,v) \in \mathcal{K}$, $\forall a \in A_{\mathrm{L}}$. As in the node-arc formulation, variables $y_i \in \{0,1\}$, $\forall i \in V$, indicate whether a relay is placed at some node and variables x_a, $\forall a \in A$, indicate whether an arc is included in the solution. Observe that for $(u,v) \in \mathcal{K}$, flow variables f^{uv} corresponding to arcs leaving any copy of target node v can be fixed to zero. Similarly, arcs incident to a copy of source node u on a non-zero layer can be set to zero. Instead of formulating the corresponding constraints we omit variables with respect to these arcs by using the notation

$$\hat{A}_{\mathrm{L}}^{uv} = \bigcup_{u_l \in V_{\mathrm{L}} : l > 0} \delta^+(u_l) \cup \bigcup_{u_l \in V_{\mathrm{L}}} \delta^-(u_l) \cup \bigcup_{v_l \in V_{\mathrm{L}}} \delta^+(v_l)$$

and deriving a formulation using flow variables f_a^{uv} for arcs from $A_{\mathrm{L}} \setminus \hat{A}_{\mathrm{L}}^{uv}$ only, given a commodity $(u,v) \in \mathcal{K}$. Formulation ($\mathrm{L_{MCF}}$) reads then as follows:

$$\min \sum_{i \in V} c_i y_i + \sum_{a \in A} w_a x_a \tag{6.2a}$$

$$\text{s.t.} \sum_{a \in \delta^+(u_0)} f_a^{uv} = 1 \qquad \forall (u,v) \in \mathcal{K}, \tag{6.2b}$$

$$\sum_{a\in\delta^-(i_l)} f_a^{uv} - \sum_{a\in\delta^+(i_l)} f_a^{uv} = 0 \qquad \forall (u,v)\in\mathcal{K}, \forall i_l \in V_{\mathrm{L}} : i \notin \{u,v\}, \tag{6.2c}$$

$$\sum_{v_l\in V_{\mathrm{L}}} \sum_{a\in\delta^-(v_l)\setminus A_{\mathrm{L}}^{\mathrm{r}}} f_a^{uv} = 1 \qquad \forall (u,v)\in\mathcal{K}, \tag{6.2d}$$

$$\sum_{i_l\in V_{\mathrm{L}}} \sum_{a\in\delta^-(i_l)\setminus A_{\mathrm{L}}^{\mathrm{r}}} f_a^{uv} \leq 1 \qquad \forall (u,v)\in\mathcal{K}, \forall i \in V\setminus\{u,v\}, \tag{6.2e}$$

$$\sum_{a=(i_l,i_0)\in A_{\mathrm{L}}^{\mathrm{r}}} f_a^{uv} \leq y_i \qquad \forall (u,v)\in\mathcal{K}, \forall i \in V, \tag{6.2f}$$

$$\sum_{a=(i_l,j_m)\in A_{\mathrm{L}}^{\mathrm{a}}} f_a^{uv} \leq x_{ij} \qquad \forall (u,v)\in\mathcal{K}, \forall (i,j) \in A, \tag{6.2g}$$

$$\sum_{a\in\sigma} f_a^{uv} \leq |\sigma| - 1 \qquad \forall (u,v)\in\mathcal{K}, \forall \sigma \in \mathcal{P}_{\mathrm{inf}}, \tag{6.2h}$$

$$y_i \in \{0,1\} \qquad \forall i \in V, \tag{6.2i}$$

$$x_a \in \{0,1\} \qquad \forall a \in A, \tag{6.2j}$$

$$f_a^{uv} \in \{0,1\} \qquad \forall (u,v)\in\mathcal{K}, \forall a \in A_{\mathrm{L}} \setminus \hat{A}_{\mathrm{L}}^{uv}. \tag{6.2k}$$

For each commodity, flow balance constraints (6.2b)–(6.2d) together with linking constraints (6.2g) ensure connectivity between the source and exactly one copy of the target node. Inequalities (6.2e) ensure that this connection contains at most one copy of each intermediate node, i.e., that the associated path in the original graph is simple. Constraints (6.2f) link the relay arcs to the relay variables. Recall that all fractional distances are rounded down in the construction of G_{L}. As a consequence G_{L} may contain paths whose length exceeds the distance limit $\lambda_{\max}$. Such paths are clearly infeasible, as they do not contain any relay arc. Let $\mathcal{P}_{\mathrm{inf}}$ denote this set of infeasible paths that may occur in the LG. To forbid the usage of paths from $\mathcal{P}_{\mathrm{inf}}$, we introduce *infeasible path constraints* (6.2h), cf. Ascheuer et al. [6]. These constraints, which are only considered if arcs with fractional distance values exist, are separated dynamically, see Section 6.3.3 for details.

6.2.2 Cut Formulation

In contrast to formulation ($\mathrm{L_{MCF}}$) which considers one variable for each commodity pair and LG arc, the *layered cut formulation* ($\mathrm{L_{CUT}}$) uses one LG variable $z_a^u \in \{0,1\}$ for each source $u \in S$ and LG arc $a \in A_{\mathrm{L}}$. By means of an exponential number of connectivity constraints, each set of variables $\mathbf{z}^u$ will model an arborescence rooted at $u \in S$ that reaches all targets $v \in T^u$, cf. Gouveia et al. [80] where a similar idea has been used in the context of a Steiner tree problem with multiple root nodes.

Similar to formulation ($\mathrm{L_{MCF}}$), for $u \in S$, we can eliminate z^u variables associated to arcs incident to a copy of source node u on a non-zero layer. In other words, for a given $u \in S$, we define

$$\hat{A}_{\mathrm{L}}^u = \bigcup_{u_l\in V_{\mathrm{L}}:l>0} \delta^+(u_l) \cup \bigcup_{u_l\in V_{\mathrm{L}}} \delta^-(u_l)$$

and work only with z^u_a variables from $A_{\mathrm{L}} \setminus \hat{A}^u_{\mathrm{L}}$. Notice that, in contrast to the formulation ($\mathrm{L}_{\mathrm{MCF}}$), the arcs leaving target nodes cannot be removed from this model. The ($\mathrm{L}_{\mathrm{CUT}}$) formulation reads as follows:

$$\min \sum_{i \in V} c_i y_i + \sum_{a \in A} w_a x_a \tag{6.3a}$$

$$\text{s.t.} \sum_{a \in \delta^-(W)} z^u_a \geq 1 \qquad \forall u \in S, \forall W \subseteq V_{\mathrm{L}} \setminus \{u_0\}, \exists v \in T^u : \{v_l \in V_{\mathrm{L}}\} \subseteq W, \tag{6.3b}$$

$$\sum_{i_l \in V_{\mathrm{L}}} \sum_{a \in \delta^-(i_l) \setminus A^{\mathrm{r}}_{\mathrm{L}}} z^u_a = 1 \qquad \forall u \in S, \forall i \in T^u, \tag{6.3c}$$

$$\sum_{i_l \in V_{\mathrm{L}}} \sum_{a \in \delta^-(i_l) \setminus A^{\mathrm{r}}_{\mathrm{L}}} z^u_a \leq 1 \qquad \forall u \in S, \forall i \in V \setminus (T^u \cup \{u\}), \tag{6.3d}$$

$$\sum_{a=(i_l,i_0) \in A^{\mathrm{r}}_{\mathrm{L}}} z^u_a \leq y_i \qquad \forall u \in S, \forall i \in V, \tag{6.3e}$$

$$\sum_{a=(i_l,j_m) \in A^{\mathrm{a}}_{\mathrm{L}}} z^u_a \leq x_{ij} \qquad \forall u \in S, \forall (i,j) \in A, \tag{6.3f}$$

$$\sum_{a \in \sigma} z^u_a \leq |\sigma| - 1 \qquad \forall u \in S, \forall \sigma \in \mathcal{P}_{\mathrm{inf}}, \tag{6.3g}$$

$$y_i \in \{0,1\} \qquad \forall i \in V, \tag{6.3h}$$

$$x_a \in \{0,1\} \qquad \forall a \in A, \tag{6.3i}$$

$$z^u_a \in \{0,1\} \qquad \forall u \in S, \forall a \in A_{\mathrm{L}} \setminus \hat{A}^u_{\mathrm{L}}. \tag{6.3j}$$

Connectivity constraints (6.3b) state that every subset of nodes containing all copies of some target node must be connected to the corresponding source. As there exist exponentially many of these constraints, we will add them on the fly in a cutting plane approach, see Section 6.3.3 for details. Constraints (6.3c) and (6.3d) prevent nodes from being visited more than once, i.e., each target node is visited exactly once and each non-target node is visited at most once. Thus, together with constraints (6.3b) they ensure that, for every source $u \in S$, the subgraph induced by all arcs $a \in A_{\mathrm{L}}$ such that $z^u_a = 1$ is an arborescence rooted at u_0 that contains exactly one copy of each node $v \in T^u$. The LG variables are linked to the relay node and arc variables on the original graph by inequalities (6.3e) and (6.3f), respectively. Infeasible path constraints (6.3g) are considered in the case of fractional distances to ensure that paths violating the distance constraint are not used, see Section 6.3.3 for their separation.

In the upcoming polyhedral comparison, we compare the strength of the two proposed formulations, concerning the quality of their LP relaxation bounds. In doing so, we ignore infeasible path constraints (6.2h) and (6.3g), as these valid inequalities are only used for cutting off infeasible integer solutions in case of fractional distances and not for strengthening the LP relaxation of our models. Additionally, since the two sets of inequalities are defined in different variable spaces it is not obvious how they relate to each other in this context.

Theorem 6.2. *Formulations (L_{MCF}) and (L_{CUT}) without infeasible path constraints* (6.2h) *and* (6.3g)*, respectively, are equally strong, i.e., the LP relaxation values of the two models coincide.*

Proof. Let $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{f}^*)$ be an optimal LP solution of the ($\mathrm{L}_{\mathrm{MCF}}$) model. We show how to construct a feasible solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{z}})$ of the ($\mathrm{L}_{\mathrm{CUT}}$) model with the same solution value. We set $\tilde{\mathbf{x}} = \mathbf{x}^*$, $\tilde{\mathbf{y}} = \mathbf{y}^*$, and

$$\tilde{z}^u_a = \max_{(u,v)\in\mathcal{K}} f^{uv}_a \qquad \forall u \in S, \forall a \in A_{\mathrm{L}}.$$

Following this definition, it is not difficult to see that flow-based capacity constraints (6.2f) and (6.2g) imply constraints (6.3e) and (6.3f), respectively. Consider a node $u \in S$. The flow-balance constraints (6.2d) are slightly different from the classical ones, due to the aggregation of the incoming flow at the target node $v \in T^u$. In this constraint the incoming flow is aggregated over all copies $v_l \in V_{\mathrm{L}}$ of the target node $v \in T^u$. This can be interpreted as a flow-balance constraint in a modified LG, say G^{uv}_{L}, in which a target node t_v is introduced for each node $v \in T^u$, and arcs (v_l, t_v) with infinite capacity are added to this graph. Hence, in such a modified graph, the flow-balance constraints (6.2b)–(6.2d) guarantee existence of a path from u_0 to t_v, for each $t \in T^u$. By the max-flow min-cut theorem, this implies that cut-set inequalities (6.3b) are satisfied. Degree-constraints (6.3c) and (6.3d) are not satisfied by an arbitrary flow $\mathbf{f}^*$, but the flow can be rerouted (without changing the capacities given by $\mathbf{x}^*$ and $\mathbf{y}^*$) so that these constraints are always satisfied.

Consider now an optimal LP solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{z}})$ of the ($\mathrm{L}_{\mathrm{CUT}}$) model. For each commodity pair $(u, v) \in \mathcal{K}$, we consider the graph G^{uv}_{L} described above, with arc capacities cap_a defined as:

$$\mathrm{cap}_a = \tilde{z}^u_a, \forall a \in A_{\mathrm{L}} \qquad \mathrm{cap}_a = \infty, \forall a = (v_l, t_v), v_l \in V_{\mathrm{L}}.$$

By the max-flow min-cut theorem applied to G^{uv}_{L}, it follows that for each $(u, v) \in \mathcal{K}$, one can send one unit of flow from u_0 to t_v in G^{uv}_{L} using $\tilde{\mathbf{z}}$ (and hence $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{x}}$) as capacities. Since $f^{uv}_a \leq \tilde{z}^u_a$ holds for each $a \in A_{\mathrm{L}}$ and $(u, v) \in \mathcal{K}$, constraints (6.2e)–(6.2g) are implied by (6.3d)–(6.3f) which concludes the proof. ☐

We also propose to extend the ($\mathrm{L}_{\mathrm{CUT}}$) model by considering *flow-balance constraints* (6.4). For each source $u \in S$, they ensure that an outgoing arc of LG node i_l, $i \notin T^u \cup \{u\}$, has to be used in the arborescence associated to source u if at least one incoming arc is chosen as well:

$$\sum_{a\in\delta^-(i_l)} z^u_a \leq \sum_{a\in\delta^+(i_l)} z^u_a \qquad \forall u \in S, \forall i_l \in V_{\mathrm{L}} : i \notin T^u \cup \{u\}. \tag{6.4}$$

While the flow-balance constraints are not necessary to ensure validity of ($\mathrm{L}_{\mathrm{CUT}}$), there exist cases in which they strengthen the associated LP relaxation. Figure 6.4 shows one of the rather typical situations in which this happens due to involved relay arcs. By adding flow-balance inequalities, the LP solution becomes integral and corresponds to the optimal solution shown in Figure 6.3. Observe that the two incoming arcs to node 5_7 belong to different variable sets in the corresponding LP solution to ($\mathrm{L_{MCF}}$) for which reason similar constraints are not strengthening there. Besides strengthening the LP relaxation of ($\mathrm{L_{CUT}}$), the flow-balance constraints also help to improve convergence by reducing the number of violated connectivity cuts (6.3b).

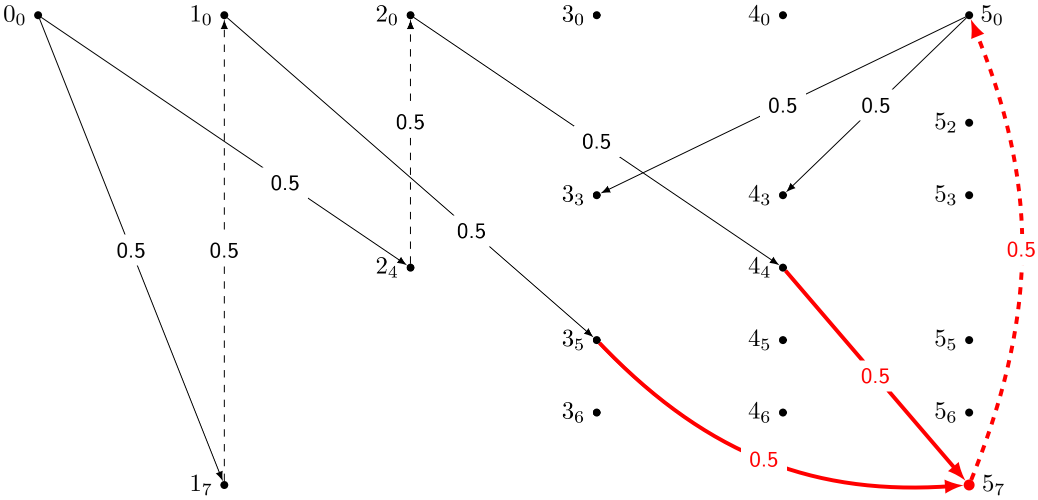


Figure 6.4: Optimal LP solution of ($\mathrm{L_{CUT}}$) on the LG for the input in Figure 6.1 without flow-balance constraints. Only arcs with associated non-zero variables are shown, labeled with the respective LP values. The violation of flow-balance constraints at node 5_7 is marked bold and red.

6.2.3 Symmetry Breaking Constraints

By construction of the LG, it can sometimes happen that multiple feasible embeddings of rooted arborescences, one for each $u \in S$, exist. Each such embedding results in the same solution in the original graph, and hence, symmetries may be introduced in our ($\mathrm{L_{MCF}}$) and ($\mathrm{L_{CUT}}$) models. Since these symmetries may deteriorate the performance of branch-and-bound (B&B) based approaches, we next introduce two families of symmetry breaking constraints. One typical situation arises if the routing paths of two different commodities contain a common node that needs to be used as a relay by only one of them. Let $i \in V$ be a node at which a relay has to be installed and let $(u, v) \in \mathcal{K}$ be a commodity that does not need to use i as a relay along its routing path. Assume that the distance to the previous relay (or the commodity source) of i along the path connecting u and v is equal to l. Furthermore, let (i, j) be the outgoing arc of node i on this path. Then, two feasible routing paths in G_L exist:

1. If the relay at i is not used, then LG arc $(i_l, j_{l+d_{ij}})$ belongs to the arborescence rooted at u, and is used to connect u_0 to some copy of v in V_L;
2. Alternatively, if the relay at i is used, the subpath given by $\{(i_l, i_0), (i_0, j_{d_{ij}})\}$ is used instead.

To get rid of symmetries implied by such ambiguities, we force that in every feasible routing path installed relays are used whenever possible.

In case of (L_CUT), this is enforced by constraints (6.5) that forbid the use of non-relay arcs emanating from some node i_l, $l > 0$, if a relay is installed at node i:

$$\sum_{a=(i_l,j_m)\in A_\mathrm{L}\setminus A_\mathrm{L}^\mathrm{r}:l>0} z_a^u \le M_i^u \cdot (1 - y_i) \qquad \forall u \in S, \forall i \in V, i \ne u. \tag{6.5}$$

Thereby, M_i^u is a (tight) upper bound on the out-degree of node i in the arborescence rooted at $u \in S$ which is defined as follows:

$$M_i^u = \begin{cases} \min(|T^u|, |\delta^+(i)|) & \text{if } i \notin T^u \\ \min(|T^u| - 1, |\delta^+(i)|) & \text{otherwise.} \end{cases}$$

For (L_MCF) we use the stronger variant of the above symmetry breaking constraints

$$\sum_{a=(i_l,j_m)\in A_\mathrm{L}\setminus A_\mathrm{L}^\mathrm{r}:l>0} f_a^{uv} \le 1 - y_i \qquad \forall (u,v) \in \mathcal{K}, \forall i \in V, i \ne u \tag{6.6}$$

that exploit the fact that the binary flow variables are disaggregated per commodity. Thus, the outflow of each node is at most one.

6.3 Algorithmic Framework

This section describes all implementation details that are relevant to ensure a good performance of our approaches. These include: (1) preprocessing techniques that aim to reduce the number of variables that have to be considered, (2) further valid inequalities, (3) the separation routines of all families of inequalities that are added dynamically, (4) customized branching priorities, and (5) a heuristic to obtain initial solutions.

6.3.1 Preprocessing

Reductions may be possible for nodes $i \in V$ whose in-degree or out-degree is equal to one at some layer. If $\delta^-(i_l) = \{(j_p, i_l)\}$ for a node $i_l \in V_\mathrm{L}$, $i \notin S$, we can remove a possibly existing outgoing arc (i_l, j_m) and all associated variables since using it would induce a cycle of length two in the original graph. Similarly, incoming arcs (j_p, i_l) can be eliminated if $\delta^+(i_l) = \{(i_l, j_m)\}$ in case $i \notin T$. In case a non-source node becomes unreachable (i.e., all incoming arcs are removed), this node and all its outgoing arcs can be removed as well (cf. *simple path reductions* introduced in De Boeck and Fortz [49]). Similarly, a non-target node without outgoing arcs can be removed together with all its incoming arcs. Additional reductions can be made if nodes are unreachable from a particular target, or have no remaining flow or layered arc variables associated to incoming (outgoing, respectively) arcs for a particular commodity or source node. In these cases, we eliminate the flow variables associated with that commodity or the layered arc variables associated with some source, respectively. These procedures are iteratively applied in several elimination rounds until no further reductions occur.

6.3.2 Valid Inequalities

In this section, we describe two further families of valid inequalities for formulation ($\mathrm{L}_{\mathrm{CUT}}$). Though both are implied by the LG connectivity constraints (6.3b) considering them before separating the latter inequalities typically turns out to be beneficial for the performance of our B&C approaches.

Connectivity constraints on G. Connectivity constraints (6.7) on the original graph are analogous to inequalities (6.3b) on the LG:

$$\sum_{a \in \delta^-(W)} x_a \geq 1 \qquad \forall W \subset V : \exists (u,v) \in \mathcal{K}, u \notin W, v \in W. \tag{6.7}$$

They ensure that each node set that separates source and target of a commodity must have at least one incoming arc. From the max-flow min-cut theorem, one can easily conclude that any solution satisfying constraints (6.7) contains a path from u to v for every commodity $(u,v) \in \mathcal{K}$. This path may, however, contain relay-free subpaths whose distance exceeds $\lambda_{\max}$. Thus, they are not sufficient to guarantee a feasible solution. A main advantage compared to the LG connectivity constraints (6.3b) is that they are specified on the arc design variables of the original graph, thus each such cut influences all commodities. Since the number of connectivity constraints (6.7) is exponential, we separate them dynamically; see Section 6.3.3 for details.

Two-cycle inequalities. Constraints (6.8) ensure that an outgoing arc of some LG node can only be used if at least one incoming arc whose source is different from the outgoing arc's target is used as well:

$$\sum_{a'=(p_r,i_l) \in \delta^-(i_l): p \neq j} z^u_{a'} \geq z^u_a \qquad \forall u \in S, \forall a = (i_l, j_m) \in A_{\mathrm{L}} : i \neq u. \tag{6.8}$$

Two-cycle inequalities (6.8) are implied by LG connectivity constraints (6.3b) and do not strengthen the formulation [76]. Similar to cut constraints (6.7), they are, however, beneficial for reducing the number of dynamically separated cut-set inequalities (6.3b).

Since the number of flow-balance constraints (6.4) and two-cycle inequalities (6.8) is pseudo-polynomial, two implementation variants are considered in our computations: (1)

adding them exhaustively to the initial formulation, and (2) separating them dynamically. Details of the used separation procedures are given below in Section 6.3.3.

6.3.3 Separation

In this section we describe the separation procedure used in our B&C approach for formulation ($\mathrm{L_{CUT}}$) that dynamically adds layered connectivity constraints (6.3b), flow balance constraints (6.4), connectivity constraints on the original graph (6.7), and two-cycle elimination constraints (6.8). Two variants of the separation are considered in this chapter. In the following, the version to which we refer as $\mathrm{L_{CUT}}$-d is described in detail, and minor modifications for the other variant, denoted by $\mathrm{L_{CUT}}$-s, are provided below. The overall separation procedure for $\mathrm{L_{CUT}}$-d is outlined in Algorithm 6.1.

Algorithm 6.1: Separation procedure for $\mathrm{L_{CUT}}$-d.

```
separate cut-set inequalities (6.7) on the original graph
separate flow-balance constraints (6.4)
separate two-cycle inequalities (6.8)
if no flow-balance constraints and two-cycle inequalities added then
    separate cut-set inequalities (6.3b) on the LG
end
```

First, possibly violated cut-set constraints on the original graph are identified using the maximum flow algorithm by Cherkassy and Goldberg [37] (cf. Step 1 of Algorithm 6.1). Thereby, so-called nested cuts (see, e.g., Ljubić et al. [115]) are considered which means that the capacities of all arcs included in just added cuts are set to one and the flow computation is subsequently repeated to possibly find further violated inequalities. This procedure is applied for each commodity pair until no further violated inequalities are found. Before proceeding with the next commodity pair, all arc capacities are reset (to the original values induced by the current LP solution). Since this procedure may yield identical cuts for different commodity pairs, we employ duplicate detection and stop separating cuts for the current commodity as soon as a duplicate is identified. To keep track of the already added cuts and check for duplicates we use hash sets. The order in which the commodities are separated is perturbed in each separation call, i.e., we consider the commodities in a random order based on a fixed seed value.

Once this first separation routine terminates, we add violated flow-balance (6.4) and two-cycle inequalities (6.8) (cf. Steps 2 and 3 of Algorithm 6.1). Separation of these constraints is performed by inspecting the LP values of relevant variables for all not yet added inequalities. Complete separation of the two-cycles turned out to be too inefficient. Therefore, we resort to a slightly simpler approach that adds those inequalities if the following condition is violated:

$$\sum_{a' \in \delta^-(i_l)} z^u_{a'} \geq z^u_a \qquad\qquad \forall u \in S, \forall a = (i_l, j_m) \in A_\mathrm{L} : i \neq u.$$

Finally, we separate LG connectivity cuts (6.3b) in case neither flow-balance nor two-cycle inequalities have been added in the previous step (cf. Step 5 of Algorithm 6.1). Such a conditional separation is beneficial for avoiding too many (possibly redundant) constraints in the model. As before, violated cuts are identified by maximum flow computations using the algorithm from [37]. To detect a violated inequality of type (6.3b), for each $(u, v) \in \mathcal{K}$, we construct the LG G_{L}^{uv} as described in the proof of Theorem 6.2, and calculate the maximum flow between u_0 and the target node t_v, using z_a^u values as arc capacities for the arcs from A_{L} and ∞ for the arcs adjacent to t_v. If the obtained flow is less than one, the violated cut is added to the model. Again, we consider nested cuts, duplicate handling, and fixed-seed randomization for the order in which the commodity pairs are processed.

In the second implementation variant of our B&C approach, denoted by $\mathrm{L}_{\mathrm{CUT}}$-s, flow-balance constraints and two-cycle inequalities are not separated. Instead, since there is only a pseudo-polynomial number of them, these cuts are added a priori to the model. In addition, the layered cut-set inequalities are separated unconditionally, i.e., the overall separation procedure for $\mathrm{L}_{\mathrm{CUT}}$-s consists of Steps 1 and 5 of Algorithm 6.1 performed sequentially.

To avoid separating too many inequalities, we only add cut-set inequalities if they are violated by a value of at least 0.5. Flow-balance constraints and two-cycle inequalities, however, are separated without such a threshold.

Infeasible path cuts. Constraints (6.2h) and (6.3g) are only considered if the input instance contains fractional distance values, and separated if the current candidate solution vector is integral and satisfies all other types of dynamically separated inequalities. In this case, violated constraints corresponding to (inclusion-wise) minimal infeasible paths starting at source or relay nodes are identified by breadth-first search.

6.3.4 Branching

Several properties of feasible solutions are enforced on the LG. The objective function, however, depends solely on the variables corresponding to the original graph. Moreover, decisions concerning the arcs available in the original graph directly influence the LG variables. Hence, it is reasonable to focus on the former for branching decisions. Regarding the two types of variables for the original graph—edge and relay variables—it is natural to prioritize the relay variables since placing a relay is usually much more expensive than installing a connection along an arc. To stay consistent with the literature we do not use custom branching priorities for the node-arc formulation.

6.3.5 Initial Heuristic

Before starting the B&C algorithm, we compute a feasible solution and hand it over to the MILP solver as initial primal bound. The heuristic resembles Prim's algorithm for spanning trees: it computes optimal paths for one commodity at a time and sets the costs of used arcs and relays to zero before it proceeds with the next source-target pair. When only a single commodity pair is given, the DNDPR is known as the minimum cost path problem with relays (MCPPR). This latter problem can be solved exactly using an efficient dynamic programming (DP) algorithm proposed by Laporte and Pascoal [105]. However, different to the DNDPR, the MCPPR also allows connecting commodities by non-simple paths. We therefore adjust the DP algorithm from Laporte and Pascoal [105] by keeping track of already visited nodes in each state, in order to disallow extensions that form cycles.

To improve the basic algorithm, we consider some extensions. Observe that the design of the heuristic entails a strong influence of the order in which the commodities are processed. We attempt to reduce this influence by considering ten different permutations based on fixed-seed randomization and then keep the best solution.

Additionally, within the DP algorithm for the MCPPR we not only order the labels by non-decreasing cost but also break ties by favoring paths that reach the considered node at smaller distance. Once the best solution among the ten runs has been identified, we run the DP algorithm once again with the costs of all used relays and arcs set to zero. In certain cases this helps to avoid redundancies resulting in a smaller cost. Similar heuristics based on sequential upgrades of a partial solution have been used in Chapter 3 and [110].

6.4 Computational Study

In this section we present computational results for the considered algorithms and variants. We start by giving details on the computational environment as well as the used test instances and the motivation for their selection. Finally, we present the obtained results.

Our algorithms are implemented in C++ using CPLEX 12.7.1 as a general-purpose MILP solver. All experiments have been performed in single thread mode with default parameter settings. Experiments have been executed on an Intel Xeon E5-2670v2 machine with 2.5 GHz. The computation time limit has been set to 7200 seconds.

In the following we compare the four solution approaches described in Table 6.1. Note that in both $\mathrm{L_{CUT}}$-s and $\mathrm{L_{CUT}}$-d, cut-set inequalities (6.3b) and (6.7) are separated dynamically. In $\mathrm{L_{CUT}}$-s, flow-balance constraints (6.4) and two-cycle inequalities (6.8) are added initially to the model while in $\mathrm{L_{CUT}}$-d these two sets are separated dynamically as described in Section 6.3.3.

6.4.1 Instances

Benchmark instances used by the authors of [110] are no longer available (personal communication with X. Li). Due to the lack of other existing benchmark instances for the DNDPR we constructed new ones based on existing instances for the NDPR. We consider three sets of benchmark instances: (1) asymmetric instances derived from Cabral et al.

Table 6.1: Overview of the tested algorithms with their abbreviations. Column “base” denotes the inequalities of the core model, column “static” provides valid inequalities that are added to the initial formulation, and column “separation” provides valid inequalities as well as inequalities of the base formulation that are separated dynamically.

		Inequalities		
Abbreviation	Model	Base	Static	Separation
NA	(NA) from [110]	(6.1b)–(6.1j)	-	-
$\mathrm{L}_{\mathrm{MCF}}$	($\mathrm{L}_{\mathrm{MCF}}$)	(6.2b)–(6.2k)	(6.6)	(6.2h)
$\mathrm{L}_{\mathrm{CUT}}$-s	($\mathrm{L}_{\mathrm{CUT}}$)	(6.3b)–(6.3j)	(6.4), (6.5), (6.8)	(6.3b), (6.3g), (6.7)
$\mathrm{L}_{\mathrm{CUT}}$-d	($\mathrm{L}_{\mathrm{CUT}}$)	(6.3b)–(6.3j)	(6.5)	(6.3b), (6.3g), (6.4), (6.7), (6.8)

[30], (2) symmetric instances derived from Konak [100], and (3) a set of newly generated symmetric instances. In the following we shortly outline the construction procedures for the original instances which involve undirected graphs, and then discuss our adjustments to obtain directed graphs.

Cabral instances. Cabral et al. [30] generated instances based on square grid graphs (i.e., each node is connected to its direct vertical and horizontal neighbors). Integral edge costs and distances are chosen uniformly at random from the interval $[10, 30]$ and the distance limit $\lambda_{\max}$ is equal to 70. The relay costs are selected uniformly at random from $\{\lambda_{\max}, \lambda_{\max} + 1, \dots, 2\lambda_{\max}\}$. All instances are based on grid graphs with a rows and b columns (i.e., with $|V| = ab$ nodes and $|E| = 2ab - a - b$ edges). The small instance set consists of nine such graphs with $(a, b) \in \{(4, 5), (5, 5), (6, 5), (7, 5), (8, 5), (9, 5), (10, 5), (11, 5), (12, 5)\}$. For the large set (a, b) is chosen from $\{10, 20, 30, 40, 50\} \times \{5, 10, 15, 20\}$. Each set contains 10 instances for each graph and each considered number of commodities $|\mathcal{K}| \in \{5, 10\}$ (by random sampling of the commodities). Observe that the graph with $(a, b) = (10, 5)$ is contained in both sets but the sampled instances are not identical. In particular, all commodities of each instance have the same source node, i.e., $|S| = 1$. We remark that duplicate commodities exist in some of the instances from Cabral et al. [30] which can be removed in a preprocessing step.

We obtained directed instances by replacing each edge by two directed arcs. Distance and cost of the first arc are equivalent to those of the edge. The values for the second arc are chosen uniformly at random from the interval $[10, 30]$. Instances for which the specified number of commodities does not match the actual number have been corrected by inserting sufficiently many new commodities while preserving the property that $|S| = 1$. Note that Li et al. [110] rely on the same approach to construct their instances although they use new base graphs instead of those from [30]. Thus, our new instance set is comparable in size and structure. The basic instance properties ($|V|$, $|A|$, $|\mathcal{K}|$, and $\lambda_{\max}$) are shown in Tables 6.3 and 6.4. We consider all of the 180 small instances and all of the large instances except for those with $b = 20$ as those turned out to be too challenging to provide meaningful insights. Instead, we included further instances with fifteen commodities whose base graphs were also generated by Cabral et al. [30] but remained unpublished. Hence, the large set consists in total of 450 instances. In our result tables we mark the subsets of large instances considered by Li et al. [110] with the set indices used in their paper.

Konak instances. Konak [100] generated instances by first placing $|V| \in \{40, 50, 60, 80, 160\}$ nodes at random integer coordinates $(x, y) \in [0, 100] \times [0, 100]$. Initially all node pairs $i, j \in V$ are connected by arcs with lengths d_{ij} set to the Euclidean distance, while the costs are either equal to the arc length (type I) or equal to $\lambda_{\max} - d_{ij}$ (type II). Edges with length beyond the distance limit are omitted. This leads to instance sizes ranging from $|V| = 40$ and $|E| = 198$ up to $|V| = 160$ and $|E| = 3624$. Relay costs are selected uniformly at random from $\{0, 1, \ldots, 100\}$. Using $\lambda_{\max} \in \{30, 35\}$ and $|\mathcal{K}| \in \{5, 10\}$, 20 instances have been generated for each of the two types. Each of these instances typically contains multiple sources and targets.

Directed instances are obtained by replacing each edge by two directed arcs. However, this time both arcs have the cost and distance of the original edge. This is done to keep the instance Euclidean and also to preserve the direct or indirect correlation of edge costs and distances. See Table 6.5 for an overview.

The newly generated third instance group uses a similar construction principle as the instances by Konak [100] and is specifically designed to reflect a practical application from telecommunications. Further details are given in Section 6.4.6 below.

6.4.2 Comparison to the State of the Art

As indicated above, we could not obtain the instances used in the previous literature. However, the Cabral instances are comparable in structure and size to those tested in [110]. This allows us to obtain at least an intuition on how our exact algorithms compare to those presented in [110]. As reference point we employ the node-arc formulation. We compare speedups between NA and the best B&P approach from the literature as well as the algorithms based on our LG formulations. The respective values are computed by t_{NA}/t_{alg} for $alg \in \{\text{B\&P}, \text{L}_{\text{MCF}}, \text{L}_{\text{CUT}}\text{-s}, \text{L}_{\text{CUT}}\text{-d}\}$. Since the Cabral instances feature 10 instances of each type, results have been aggregated by computing averages. The results are shown in Table 6.2. Observe that we provide this comparison only for the small Cabral instances because there are no NA results in Li et al. [110] for the large set. The speedup values of our algorithms are slightly worse than those from the literature for the smallest instances but they are considerably better for $a \geq 6$. Also note that on our instances the node-arc formulation sometimes terminated prematurely due to the time limit. Allowing the algorithm to finish—as done in [110]—would have resulted in even larger speedups.

Without the original benchmark set, a precise comparison is impossible. Yet these results indicate that our algorithms are at least as fast as those from the existing literature and most likely outperform them significantly.

Table 6.2: Speedup ratio to the node-arc formulation. Values have been obtained by dividing the computation time of the node-arc formulation through the computation time of the respective algorithm. The first column has been obtained by extracting the respective results from [110]. The highest speedup per column is marked bold.

	Speedup ratio			
Instance	B&P2 (Li et al.)	L_{MCF}	L_{CUT}-s	L_{CUT}-d
04A05B70L05K	**8.4**	1.3	3.1	3.2
04A05B70L10K	**55.9**	9.6	35.7	19.8
05A05B70L05K	**20.8**	8.2	15.7	14.1
05A05B70L10K	**107.3**	14.4	53.6	44.9
06A05B70L05K	7.1	14.0	**25.4**	22.3
06A05B70L10K	61.1	67.8	**164.4**	98.7
07A05B70L05K	**31.8**	15.0	19.2	16.2
07A05B70L10K	34.6	224.1	**476.3**	289.8
08A05B70L05K	9.3	194.9	**216.3**	81.7
08A05B70L10K	92.0	286.1	**543.7**	218.6
09A05B70L05K	9.9	97.1	**110.9**	69.0
09A05B70L10K	40.5	217.8	**391.6**	237.8
10A05B70L05K	40.9	305.6	**319.8**	122.1
10A05B70L10K	33.6	467.7	**1337.3**	683.9
11A05B70L05K	25.1	266.0	**306.5**	123.0
11A05B70L10K	45.4	731.0	**1500.4**	555.7
12A05B70L05K	5.2	**597.4**	528.4	215.0
12A05B70L10K	110.1	538.7	**1164.3**	405.9

6.4.3 LP Relaxation Bounds

In the following, we compare the quality of lower bounds that can be obtained by the three algorithms from Table 6.1: L_{MCF}, L_{CUT} (L_{CUT}-s and L_{CUT}-d provide the same lower bounds), and NA. Note that we ignore infeasible path constraints in this comparison since we only use them to cut off infeasible integer solutions in cases of fractional distances and not to strengthen the LP bounds. When computing LP bounds we deactivate CPLEX presolving, general purpose heuristics, and general purpose cuts. In addition, no threshold value is set for the separation of cut-set inequalities (6.3b) and (6.7).

The presented LP gaps are computed as $(UB^* - LB)/UB^*$ where UB^* is the best known upper bound and LB is the lower bound obtained by the LP relaxation. Tables 6.3 and 6.4 report results obtained on the Cabral instances, and Table 6.5 provides results for the Konak instances.

Cabral Instances. We observe that the algorithms based on (L_{CUT}) yield the strongest bounds. L_{MCF} follows closely behind but mostly delivers strictly weaker bounds. The reason for this is the fact that the Cabral instances consider only a single source node. This means that algorithms based on (L_{CUT}) use precisely one set of variables with respect to the LG on which they model an arborescence. The multi-commodity flow formulation, on the other hand, uses one set of variables per commodity pair. In this situation the cut model benefits from aggregating per source which enables it to obtain a

Table 6.3: LP gaps for the small directed Cabral instances. Each line represents the average across ten instances. The strongest bounds per row are marked bold.

	Properties				LP gap [%]		
Instance	$\lvert V\rvert$	$\lvert A\rvert$	$\lambda_{\max}$	$\lvert\mathcal{K}\rvert$	$\mathrm{L_{MCF}}$	$\mathrm{L_{CUT}}$	NA
04A05B70L05K	20	62	70	5	0.2	**0.0**	27.6
04A05B70L10K	20	62	70	10	0.2	**0.0**	35.0
05A05B70L05K	25	80	70	5	0.8	**0.0**	31.4
05A05B70L10K	25	80	70	10	0.1	**0.0**	34.4
06A05B70L05K	30	98	70	5	0.5	**0.0**	36.8
06A05B70L10K	30	98	70	10	0.6	**0.0**	34.9
07A05B70L05K	35	116	70	5	0.1	**0.0**	40.5
07A05B70L10K	35	116	70	10	0.7	**0.1**	40.6
08A05B70L05K	40	134	70	5	0.1	**0.0**	45.1
08A05B70L10K	40	134	70	10	1.0	**0.1**	40.2
09A05B70L05K	45	152	70	5	0.1	**0.0**	42.9
09A05B70L10K	45	152	70	10	0.7	**0.0**	39.8
10A05B70L05K	50	170	70	5	0.1	**0.0**	46.2
10A05B70L10K	50	170	70	10	0.9	**0.0**	43.9
11A05B70L05K	55	188	70	5	0.5	**0.0**	46.2
11A05B70L10K	55	188	70	10	0.2	**0.1**	42.5
12A05B70L05K	60	206	70	5	0.5	**0.1**	43.3
12A05B70L10K	60	206	70	10	0.8	**0.1**	42.6

stronger bound by means of flow-balance constraints (6.4). Moreover, the cut formulation yields a much smaller model here with respect to the number of variables. In general, both algorithms based on LG models deliver excellent bounds well below 5 %. The only exception are the larger instances with fifteen commodities where the bounds are slightly larger or even missing due to hitting the time limit. For the reasons mentioned above we observe that the performance of the multi-commodity flow formulation is much more susceptible to an increasing number of commodities than the cut formulation. As expected, the node-arc formulation is the weakest model with significantly worse bounds than the LG models. On the other hand, we obtain lower bounds for all instances in relatively short CPU times due to the small size of the model.

Konak instances. Compared to the Cabral instances we face much denser graphs here. Moreover, we are now dealing with multiple source nodes instead of just a single one. This means that now also the ($\mathrm{L_{CUT}}$) formulation requires multiple sets of LG variables. Under these circumstances we still obtain strong LP bounds but not as strong as on the Cabral instances, see Table 6.5. For the larger instances of type I it becomes challenging to solve the LP relaxation to optimality indicated by dashes in the table. The instances with indirectly correlated costs (type II) turned out to be much easier to solve. Here $\mathrm{L_{MCF}}$ as well as $\mathrm{L_{CUT}}$ provide results for all instances before exceeding the time limit. The results indicate that the bound strength is excellent if the computations can be completed. As before, we observe that the bounds provided by $\mathrm{L_{CUT}}$ are at least as strong as those of $\mathrm{L_{MCF}}$. The node-arc formulation (NA) yields much weaker bounds but also terminates significantly faster. Therefore, NA gives the only bounds for the

Table 6.4: LP gaps for the large directed Cabral instances. Each line considers ten instances. Averages for the gaps are computed only with respect to the instances for which all algorithms terminated within the time limit. Column #tl denotes the number of instances that terminated due to the time limit. The strongest bounds per row are marked bold. Superscripts next to the instance names refer to the comparable instance group in Li et al. [110].

	Properties				Gap [%]			#tl		
Instance	$\|V\|$	$\|A\|$	$\lambda_{\max}$	$\|\mathcal{K}\|$	$\mathrm{L}_{\mathrm{MCF}}$	$\mathrm{L}_{\mathrm{CUT}}$	NA	$\mathrm{L}_{\mathrm{MCF}}$	$\mathrm{L}_{\mathrm{CUT}}$	NA
10A05B70L05K	50	170	70	5	0.4	**0.0**	47.5	**0**	**0**	**0**
10A05B70L10K	50	170	70	10	1.0	**0.0**	44.8	**0**	**0**	**0**
10A05B70L15K	50	170	70	15	1.3	**0.0**	42.2	**0**	**0**	**0**
10A10B70L05K^{19}	100	360	70	5	1.6	**0.0**	48.4	**0**	**0**	**0**
10A10B70L10K^{25}	100	360	70	10	0.8	**0.0**	45.1	**0**	**0**	**0**
10A10B70L15K	100	360	70	15	2.6	**0.0**	45.6	**0**	**0**	**0**
10A15B70L05K^{20}	150	550	70	5	2.0	**0.3**	52.9	**0**	**0**	**0**
10A15B70L10K	150	550	70	10	3.3	**0.0**	50.1	**0**	**0**	**0**
10A15B70L15K	150	550	70	15	3.9	**0.1**	46.7	**0**	**0**	**0**
20A05B70L05K^{21}	100	350	70	5	0.3	**0.0**	53.2	**0**	**0**	**0**
20A05B70L10K^{26}	100	350	70	10	0.4	**0.1**	48.6	**0**	**0**	**0**
20A05B70L15K	100	350	70	15	0.1	**0.0**	48.7	**0**	**0**	**0**
20A10B70L05K	200	740	70	5	0.7	**0.0**	51.8	**0**	**0**	**0**
20A10B70L10K	200	740	70	10	2.0	**0.0**	50.6	**0**	**0**	**0**
20A10B70L15K	200	740	70	15	3.6	**0.1**	49.9	**0**	**0**	**0**
20A15B70L05K	300	1130	70	5	1.4	**0.0**	52.4	**0**	**0**	**0**
20A15B70L10K	300	1130	70	10	3.7	**0.1**	52.2	**0**	**0**	**0**
20A15B70L15K	300	1130	70	15	6.6	**2.5**	53.2	2	2	**0**
30A05B70L05K^{22}	150	530	70	5	**0.0**	**0.0**	55.3	**0**	**0**	**0**
30A05B70L10K^{27}	150	530	70	10	0.1	**0.0**	53.2	**0**	**0**	**0**
30A05B70L15K	150	530	70	15	0.3	**0.0**	51.3	**0**	**0**	**0**
30A10B70L05K	300	1120	70	5	1.0	**0.1**	54.2	**0**	**0**	**0**
30A10B70L10K	300	1120	70	10	1.4	**0.0**	54.2	**0**	**0**	**0**
30A10B70L15K	300	1120	70	15	1.9	**0.1**	53.1	4	**0**	**0**
30A15B70L05K	450	1710	70	5	0.7	**0.0**	56.0	**0**	**0**	**0**
30A15B70L10K	450	1710	70	10	1.6	**0.5**	53.6	5	**0**	**0**
30A15B70L15K	450	1710	70	15	-	-	-	10	3	**0**
40A05B70L05K^{23}	200	710	70	5	0.1	**0.0**	56.6	**0**	**0**	**0**
40A05B70L10K^{28}	200	710	70	10	0.3	**0.0**	55.7	**0**	**0**	**0**
40A05B70L15K	200	710	70	15	0.1	**0.0**	53.0	**0**	**0**	**0**
40A10B70L05K	400	1500	70	5	0.4	**0.0**	56.4	**0**	**0**	**0**
40A10B70L10K	400	1500	70	10	1.6	**0.4**	55.1	2	**0**	**0**
40A10B70L15K	400	1500	70	15	5.9	**4.7**	54.8	7	**0**	**0**
40A15B70L05K	600	2290	70	5	0.5	**0.0**	56.1	**0**	**0**	**0**
40A15B70L10K	600	2290	70	10	3.3	**1.9**	54.8	7	3	**0**
40A15B70L15K	600	2290	70	15	-	-	-	10	10	**0**
50A05B70L05K^{24}	250	890	70	5	0.1	**0.0**	58.1	**0**	**0**	**0**
50A05B70L10K^{29}	250	890	70	10	0.2	**0.0**	56.6	**0**	**0**	**0**
50A05B70L15K	250	890	70	15	0.1	**0.0**	54.9	**0**	**0**	**0**
50A10B70L05K	500	1880	70	5	0.2	**0.0**	56.6	**0**	**0**	**0**
50A10B70L10K	500	1880	70	10	1.0	**0.3**	55.2	3	**0**	**0**
50A10B70L15K	500	1880	70	15	-	-	-	10	**0**	**0**
50A15B70L05K	750	2870	70	5	0.5	**0.0**	56.4	**0**	1	**0**
50A15B70L10K	750	2870	70	10	**0.0**	**0.0**	54.4	9	7	**0**
50A15B70L15K	750	2870	70	15	-	-	-	10	10	**0**

Table 6.5: LP gaps for the directed Konak instances. Missing gap values correspond to runs that did not complete within the time limit. Bold values indicate the tightest bounds per type and instance.

					LP gap [%]					
	Properties				Type I			Type II		
Instance	$\lvert V\rvert$	$\lvert A\rvert$	$\lambda_{\max}$	$\lvert\mathcal{K}\rvert$	$\mathrm{L}_{\mathrm{MCF}}$	$\mathrm{L}_{\mathrm{CUT}}$	NA	$\mathrm{L}_{\mathrm{MCF}}$	$\mathrm{L}_{\mathrm{CUT}}$	NA
040N_05K_30L	40	396	30	5	**21.2**	**21.2**	39.2	**37.9**	**37.9**	77.3
040N_05K_35L	40	544	35	5	**4.7**	**4.7**	25.1	**0.6**	**0.6**	75.2
040N_10K_30L	40	396	30	10	22.9	**21.4**	41.3	**31.3**	**31.3**	76.6
040N_10K_35L	40	544	35	10	7.2	**6.3**	26.5	6.9	**4.9**	72.3
050N_05K_30L	50	558	30	5	**0.8**	**0.8**	31.6	**0.0**	**0.0**	76.5
050N_05K_35L	50	744	35	5	**0.0**	**0.0**	29.6	**0.0**	**0.0**	83.6
050N_10K_30L	50	558	30	10	20.1	**16.1**	48.8	**0.4**	**0.4**	79.6
050N_10K_35L	50	744	35	10	12.4	**9.4**	36.5	**0.0**	**0.0**	83.0
060N_05K_30L	60	610	30	5	**8.8**	**8.8**	51.7	**5.4**	**5.4**	84.9
060N_05K_35L	60	824	35	5	**2.6**	**2.6**	36.9	**0.0**	**0.0**	79.7
060N_10K_30L	60	610	30	10	**13.4**	**13.4**	51.1	**7.2**	**7.2**	82.1
060N_10K_35L	60	824	35	10	**4.8**	**4.8**	36.7	**0.0**	**0.0**	79.3
080N_05K_30L	80	1282	30	5	**1.7**	**1.7**	17.9	**1.2**	**1.2**	71.5
080N_05K_35L	80	1706	35	5	**0.0**	**0.0**	14.0	**0.2**	**0.2**	75.2
080N_10K_30L	80	1282	30	10	**4.3**	-	25.4	**0.6**	**0.6**	66.9
080N_10K_35L	80	1706	35	10	**4.4**	-	21.3	**0.0**	**0.0**	75.1
160N_05K_30L	160	5546	30	5	**0.3**	-	21.1	**2.1**	**2.1**	85.2
160N_05K_35L	160	7248	35	5	**6.2**	-	21.3	**3.2**	**3.2**	83.0
160N_10K_30L	160	5546	30	10	-	-	**32.1**	**2.4**	**2.4**	81.1
160N_10K_35L	160	7248	35	10	-	-	**29.5**	**0.2**	**0.2**	78.7

two largest instances of type I where the time limit is reached for the LG models. In contrast to the LG models, the node-arc formulation seems to work much better on type I instances than on type-II instances where the bounds are much worse, roughly by a factor of two.

6.4.4 Overall Performance

We continue by evaluating the performance of the MILP runs on the instances by Cabral et al. [30] and Konak [100].

Cabral instances. Our LG models are able to solve all 180 small instances to proven optimality and 334 of the 450 large instances. The MILP runs are consistent with the LP results, however, $\mathrm{L}_{\mathrm{MCF}}$ is now much closer to the cut model despite its worse bounds—at least for the small instances, see Table 6.6. Optimality gaps are computed by $(UB^* - LB)/UB^*$ where UB^* is the best known upper bound and LB is the lower bound obtained by the investigated algorithm.

On the small instances we observe a clear difference between the static and the dynamic variant of the cut formulation $\mathrm{L}_{\mathrm{CUT}}$-s and $\mathrm{L}_{\mathrm{CUT}}$-d, respectively. The reasons for the advantage of the static approach are the sparseness and the size of the input graphs.

Table 6.6: Results for the small directed Cabral instances. Each line represents the average across ten instances. Column #opt provides the number of optimally solved instances.

	Gap [%]				CPU time [s]				#opt			
Instance	L_{MCF}	L_{CUT}-s	L_{CUT}-d	NA	L_{MCF}	L_{CUT}-s	L_{CUT}-d	NA	L_{MCF}	L_{CUT}-s	L_{CUT}-d	NA
04A05B70L05K	**0.0**	**0.0**	**0.0**	**0.0**	**<1**	**<1**	**<1**	1	**10**	**10**	**10**	**10**
04A05B70L10K	**0.0**	**0.0**	**0.0**	**0.0**	1	**<1**	**<1**	8	**10**	**10**	**10**	**10**
05A05B70L05K	**0.0**	**0.0**	**0.0**	**0.0**	1	**<1**	**<1**	4	**10**	**10**	**10**	**10**
05A05B70L10K	**0.0**	**0.0**	**0.0**	**0.0**	1	**<1**	**<1**	16	**10**	**10**	**10**	**10**
06A05B70L05K	**0.0**	**0.0**	**0.0**	**0.0**	1	**<1**	1	11	**10**	**10**	**10**	**10**
06A05B70L10K	**0.0**	**0.0**	**0.0**	**0.0**	2	**1**	**1**	111	**10**	**10**	**10**	**10**
07A05B70L05K	**0.0**	**0.0**	**0.0**	**0.0**	**1**	**1**	**1**	10	**10**	**10**	**10**	**10**
07A05B70L10K	**0.0**	**0.0**	**0.0**	**0.0**	3	**1**	2	639	**10**	**10**	**10**	**10**
08A05B70L05K	**0.0**	**0.0**	**0.0**	**0.0**	**1**	**1**	2	167	**10**	**10**	**10**	**10**
08A05B70L10K	**0.0**	**0.0**	**0.0**	**0.0**	3	**2**	5	991	**10**	**10**	**10**	**10**
09A05B70L05K	**0.0**	**0.0**	**0.0**	**0.0**	**1**	**1**	**1**	78	**10**	**10**	**10**	**10**
09A05B70L10K	**0.0**	**0.0**	**0.0**	**0.0**	4	**2**	4	891	**10**	**10**	**10**	**10**
10A05B70L05K	**0.0**	**0.0**	**0.0**	**0.0**	**1**	**1**	2	302	**10**	**10**	**10**	**10**
10A05B70L10K	**0.0**	**0.0**	**0.0**	3.2	9	**3**	6	4126	**10**	**10**	**10**	8
11A05B70L05K	**0.0**	**0.0**	**0.0**	**0.0**	**1**	**1**	3	382	**10**	**10**	**10**	**10**
11A05B70L10K	**0.0**	**0.0**	**0.0**	6.8	6	**3**	8	4168	**10**	**10**	**10**	6
12A05B70L05K	**0.0**	**0.0**	**0.0**	1.9	**2**	**2**	4	906	**10**	**10**	**10**	9
12A05B70L10K	**0.0**	**0.0**	**0.0**	6.6	9	**4**	11	4666	**10**	**10**	**10**	6

Both lead to rather small models and the overhead for adding the valid inequalities in advance is manageable. Therefore, the slowdown for solving the LP relaxations is negligible but we can reduce the number of cut iterations in each node of the B&C tree significantly. Similarly, we also observe that much fewer B&B nodes—about 17 % on average—are needed until optimality can be proven. L_{MCF}, however, performs better in this respect: It solves the majority of instances already at the root node and the two "outliers" with 19 and 23 B&B nodes, respectively. The cut formulation, albeit being stronger, solves 52 instances fewer at the root node and requires up to 67 B&B nodes when adding flow-balance and two-cycle inequalities statically. The reasons for this seem to be that the fractional solutions of L_{MCF} are closer to being feasible and that L_{MCF} interacts better with the solver since no further inequalities are added in the solution process. Although the computation times of L_{MCF} and L_{CUT}-s are quite similar, we still observe that the former is considerably more sensitive to changes in the number of commodities due to the resulting increase in model size, see also Table 6.2. The node-arc formulation is significantly outperformed and cannot even solve all instances to optimality within the time limit.

Most of the observations with respect to the small instance set directly transfer to the large one, see Table 6.7. Considering the two algorithms based on (L_{CUT}) we now observe an advantage for the dynamic variant. Due to the larger graph sizes it is no longer beneficial to add all the strengthening inequalities to the initial formulation. In total the cut formulations solve the highest number of instances to optimality: 318 in the static and 334 in the dynamic variant. The multi-commodity flow formulations follows closely behind with 303 instances solved to proven optimality. In general, we observe that the flow formulation works quite well for instances with only five commodities, frequently even outperforming the cut formulation. However, as the number of commodities increases the performance starts to deteriorate. We observe significant advantages for the cut formulations in these cases. In particular, for the largest instances the algorithm based on ($\mathrm{L_{MCF}}$) often delivers no bounds while both algorithms based on ($\mathrm{L_{CUT}}$) still terminate with comparatively tight gaps. This appears to be a natural consequence of the larger model size of ($\mathrm{L_{MCF}}$) that depends to a higher degree on the number of commodities. We omit results for NA for reasons of space and since the results are by far not competitive to our LG approaches.

Konak instances. The results of solving the MILP formulations are provided in Tables 6.8 and 6.9. In accordance with the experiments on the LP bounds, type II instances are again easier to solve than type I instances. $\mathrm{L_{MCF}}$ and $\mathrm{L_{CUT}}$-d solve all instances of type II to optimality. $\mathrm{L_{CUT}}$-s, on the other hand, cannot solve the largest two instances of this set to optimality. Similarly, $\mathrm{L_{CUT}}$-s solves fewer instances of type I to optimality than the dynamic variant and leaves larger gaps whenever both terminate prematurely due to the time limit. As the graphs are denser here than the 4-grid graphs of the Cabral instances, it is no longer beneficial to add all valid inequalities in advance. Thus, dynamic separation helps to reduce the size of the LP relaxations. While being slightly slower on the largest type II instances, $\mathrm{L_{MCF}}$ outperforms the $\mathrm{L_{CUT}}$ approaches on the type I instances. There it solves seven more instances to optimality and features considerably smaller computation times on the remaining ones. However, for the largest two instances it fails to provide any non-trivial bounds. $\mathrm{L_{CUT}}$-d terminates still in the root note, but at least provides reasonable bounds. It is noticeable that $\mathrm{L_{MCF}}$ proves optimality for 17 out of 20 type II instances and 6 out of 20 type I instances already at the root note, mostly with non-zero LP gap. $\mathrm{L_{CUT}}$-s performs quite similar in this respect and solves 2 type I and 12 type II instances at the root node. $\mathrm{L_{CUT}}$-d, on the other hand, achieves this only for a single type II instance. Again, we presume that $\mathrm{L_{MCF}}$ interacts better with the solver due to having most information available from the beginning. Since these instances feature fractional distances, infeasible path constraints have to be separated to ensure feasibility. The indirectly correlated instances of type II require such cuts only in rare cases. $\mathrm{L_{CUT}}$-s and $\mathrm{L_{MCF}}$ solve all but two instances without infeasible path constraints and those two instances with just one cut each. $\mathrm{L_{CUT}}$-d requires cuts for one additional instance and uses no more than three such cuts. The directly correlated instances of type I require a higher number of infeasible path cuts. This can be explained by the fact that cost reductions can be achieved by exhausting the distance limit. Among the optimally solved instances $\mathrm{L_{MCF}}$ requires no more than 27 infeasible path cuts while five instances can be solved without them. $\mathrm{L_{CUT}}$-s solves only few of the type I instances to optimality, three of them without infeasible path cuts and the remaining five with at most 9. $\mathrm{L_{CUT}}$-d solves four instances without infeasible path cuts and the remaining ones with at most 18. With respect to the instances that terminated due to the time limit the maximum number of added infeasible path cuts is 34 for $\mathrm{L_{CUT}}$-d, the highest

Table 6.7: Results for the large directed Cabral instances. Each line represents the average across ten instances. Column #opt provides the number of optimally solved instances. Superscripts next to the instance names refer to the comparable instance group in Li et al. [110].

	Gap [%]			CPU time [s]			#opt		
Instance	L_{MCF}	L_{CUT}-s	L_{CUT}-d	L_{MCF}	L_{CUT}-s	L_{CUT}-d	L_{MCF}	L_{CUT}-s	L_{CUT}-d
10A05B70L05K	**0.0**	**0.0**	**0.0**	**1**	**1**	2	**10**	**10**	**10**
10A05B70L10K	**0.0**	**0.0**	**0.0**	8	**3**	8	**10**	**10**	**10**
10A05B70L15K	**0.0**	**0.0**	**0.0**	48	**7**	15	**10**	**10**	**10**
10A10B70L05K^{19}	**0.0**	**0.0**	**0.0**	**15**	20	37	**10**	**10**	**10**
10A10B70L10K^{25}	**0.0**	**0.0**	**0.0**	71	**38**	79	**10**	**10**	**10**
10A10B70L15K	**0.0**	**0.0**	**0.0**	1200	**108**	212	**10**	**10**	**10**
10A15B70L05K^{20}	**0.0**	**0.0**	**0.0**	**70**	125	184	**10**	**10**	**10**
10A15B70L10K	1.0	**0.0**	**0.0**	2505	**638**	732	8	**10**	**10**
10A15B70L15K	2.9	**0.0**	**0.0**	4606	**1961**	2701	5	**10**	**10**
20A05B70L05K^{21}	**0.0**	**0.0**	**0.0**	**5**	6	11	**10**	**10**	**10**
20A05B70L10K^{26}	**0.0**	**0.0**	**0.0**	62	**28**	40	**10**	**10**	**10**
20A05B70L15K	**0.0**	**0.0**	**0.0**	88	**32**	74	**10**	**10**	**10**
20A10B70L05K	**0.0**	**0.0**	**0.0**	**46**	96	172	**10**	**10**	**10**
20A10B70L10K	1.1	**0.0**	**0.0**	4101	**1332**	1731	5	**10**	**10**
20A10B70L15K	2.8	2.1	**1.7**	4886	3170	**3146**	4	**7**	**7**
20A15B70L05K	**0.0**	**0.0**	**0.0**	**423**	1589	1179	**10**	**10**	**10**
20A15B70L10K	3.1	4.1	**1.8**	6159	6086	**5231**	2	2	**7**
20A15B70L15K	16.7	8.2	**8.0**	7200	**6877**	6937	0	**1**	**1**
30A05B70L05K^{22}	**0.0**	**0.0**	**0.0**	**13**	14	32	**10**	**10**	**10**
30A05B70L10K^{27}	**0.0**	**0.0**	**0.0**	149	**77**	155	**10**	**10**	**10**
30A05B70L15K	**0.0**	**0.0**	**0.0**	567	**222**	303	**10**	**10**	**10**
30A10B70L05K	**0.0**	**0.0**	**0.0**	398	**380**	512	**10**	**10**	**10**
30A10B70L10K	0.5	0.6	**0.4**	4574	3444	**2506**	6	8	**9**
30A10B70L15K	31.4	2.7	**1.8**	7200	5811	**5548**	0	4	**5**
30A15B70L05K	**0.1**	2.1	0.8	**1192**	2829	2092	**9**	7	**9**
30A15B70L10K	21.6	5.5	**4.4**	6028	6414	**6008**	**3**	2	**3**
30A15B70L15K	90.1	9.9	**9.7**	7200	7200	7200	0	0	0
40A05B70L05K^{23}	**0.0**	**0.0**	**0.0**	32	**26**	45	**10**	**10**	**10**
40A05B70L10K^{28}	**0.0**	**0.0**	**0.0**	468	**195**	221	**10**	**10**	**10**
40A05B70L15K	**0.0**	**0.0**	**0.0**	841	**271**	374	**10**	**10**	**10**
40A10B70L05K	**0.0**	0.5	**0.0**	**407**	2184	1474	**10**	8	**10**
40A10B70L10K	11.2	3.6	**2.7**	6079	5982	**5455**	3	4	**5**
40A10B70L15K	90.0	8.0	**7.2**	6736	**6254**	6329	1	**2**	**2**
40A15B70L05K	**0.1**	1.8	1.1	**2061**	4232	4063	**9**	6	6
40A15B70L10K	80.5	10.0	**9.1**	**6949**	7200	7200	**1**	0	0
40A15B70L15K	100.0	**10.6**	11.9	7200	7200	7200	0	0	0
50A05B70L05K^{24}	**0.0**	**0.0**	**0.0**	57	**51**	86	**10**	**10**	**10**
50A05B70L10K^{29}	**0.0**	**0.0**	**0.0**	525	**254**	359	**10**	**10**	**10**
50A05B70L15K	**0.0**	**0.0**	**0.0**	1583	**884**	939	**10**	**10**	**10**
50A10B70L05K	**0.0**	1.2	0.5	**628**	2453	2379	**10**	8	8
50A10B70L10K	40.4	4.2	**3.1**	6496	6241	**5799**	**3**	2	**3**
50A10B70L15K	100.0	8.9	**8.3**	7200	7025	**6958**	0	**1**	**1**
50A15B70L05K	**0.3**	2.3	1.5	**2971**	4108	3820	**9**	6	7
50A15B70L10K	90.0	**8.1**	12.1	**6690**	7200	7115	**1**	0	**1**
50A15B70L15K	100.0	**13.9**	16.1	7200	7200	7200	0	0	0

Table 6.8: Results for the directed Konak instances of type I.

	Gap [%]				CPU time [s]			
Instance	L_{MCF}	L_{CUT}-s	L_{CUT}-d	NA	L_{MCF}	L_{CUT}-s	L_{CUT}-d	NA
040N_05K_30L	**0.0**	**0.0**	**0.0**	**0.0**	**6**	103	120	1204
040N_05K_35L	**0.0**	**0.0**	**0.0**	3.4	**23**	575	329	7200
040N_10K_30L	**0.0**	**0.0**	**0.0**	11.3	**45**	1217	1357	7200
040N_10K_35L	**0.0**	5.2	**0.0**	16.3	**153**	7200	4202	7200
050N_05K_30L	**0.0**	**0.0**	**0.0**	7.3	**1**	23	15	7200
050N_05K_35L	**0.0**	**0.0**	**0.0**	**0.0**	**3**	234	71	5417
050N_10K_30L	**0.0**	7.7	1.9	42.6	**690**	7200	7200	7200
050N_10K_35L	**0.0**	13.5	7.4	31.3	**638**	7200	7200	7200
060N_05K_30L	**0.0**	**0.0**	**0.0**	38.0	**25**	282	213	7200
060N_05K_35L	**0.0**	**0.0**	**0.0**	23.1	**3**	211	280	7200
060N_10K_30L	**0.0**	8.3	7.8	46.7	**392**	7200	7200	7200
060N_10K_35L	**0.0**	7.8	7.9	31.3	**469**	7200	7200	7200
080N_05K_30L	**0.0**	**0.0**	**0.0**	10.9	**12**	2664	1040	7200
080N_05K_35L	**0.0**	5.4	**0.0**	6.3	**24**	7200	5500	7200
080N_10K_30L	**0.6**	6.9	4.9	19.3	7200	7200	7200	7200
080N_10K_35L	**3.4**	65.4	32.5	17.5	7200	7200	7200	7200
160N_05K_30L	**0.0**	76.4	7.7	18.5	**1123**	7200	7200	7200
160N_05K_35L	**6.0**	76.7	10.4	19.8	7200	7200	7200	7200
160N_10K_30L	100.0	78.1	**26.8**	31.7	7200	7200	7200	7200
160N_10K_35L	100.0	77.7	**26.6**	29.0	7200	7200	7200	7200

among all algorithms. The node-arc formulation is not competitive and features large gaps even on the smaller instances. In contrast to the results of the LP runs it provides better results on the type II instances, like the LG algorithms.

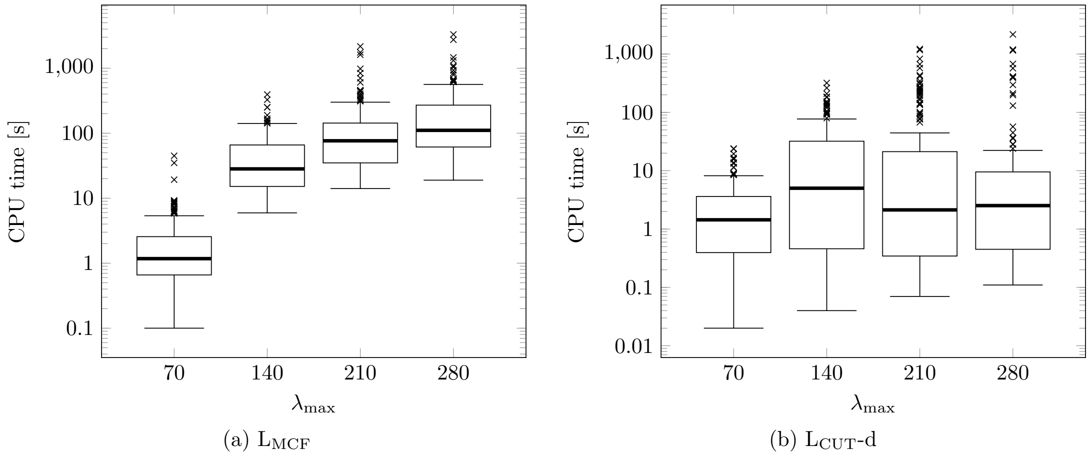
6.4.5 Evaluation of Algorithm Properties

In the following we evaluate the impact of specific instance properties and algorithmic components on the performance of the introduced approaches.

We start by evaluating the sensitivity of our LG algorithms to the distance limit. We decided to use the small Cabral instances which could all be solved to optimality with the original distance limit of $\lambda_{\max} = 70$. For our experiments we consider increased distance limits of 140, 210, and 280, while leaving the remaining instance characteristics the same. Increasing the limit further does seem relevant because with $\lambda_{\max} \geq 280$ the solutions no longer contain any relays. Figure 6.5 shows box plots for the respective computation times. We observe a moderate slowdown, leveling off as we converge towards the relay-free scenario. In general L_{CUT}-d appears to be slightly more resilient to the parameter change due to the smaller base model. The observed slowdown is to be expected since increasing the distance limit allows connecting the commodities with more complex paths consisting of a higher number of arcs. We conjecture that paths with a higher number of arcs impact most kinds of algorithms alike. In our case the number of layers increases, which leads to larger models. Column generation based approaches, on the other hand, suffer from the additional computational effort for solving the subproblems and an increased

Table 6.9: Results for the directed Konak instances of type II.

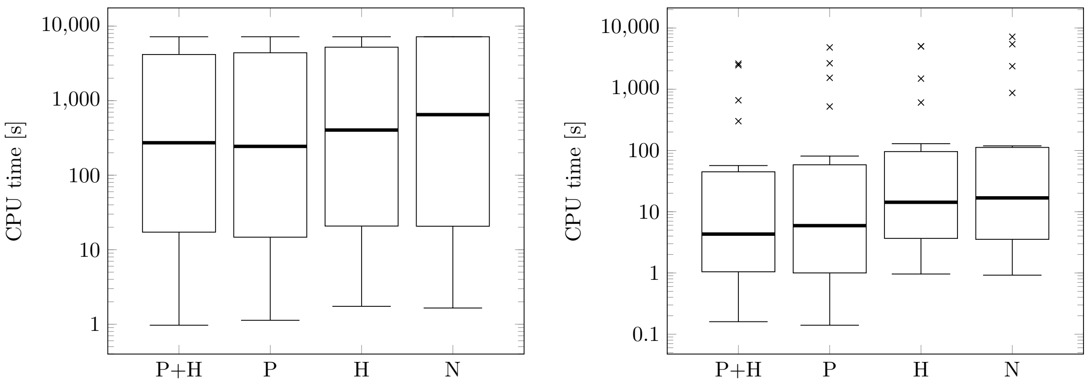
	Gap [%]				CPU time [s]			
Instance	L_{MCF}	L_{CUT}-s	L_{CUT}-d	NA	L_{MCF}	L_{CUT}-s	L_{CUT}-d	NA
040N_05K_30L	**0.0**	**0.0**	**0.0**	**0.0**	**< 1**	2	13	7
040N_05K_35L	**0.0**	**0.0**	**0.0**	**0.0**	**1**	2	2	51
040N_10K_30L	**0.0**	**0.0**	**0.0**	**0.0**	**1**	16	36	352
040N_10K_35L	**0.0**	**0.0**	**0.0**	**0.0**	**5**	30	13	3420
050N_05K_30L	**0.0**	**0.0**	**0.0**	**0.0**	**< 1**	1	1	19
050N_05K_35L	**0.0**	**0.0**	**0.0**	**0.0**	**1**	3	2	4679
050N_10K_30L	**0.0**	**0.0**	**0.0**	49.9	**1**	8	42	7200
050N_10K_35L	**0.0**	**0.0**	**0.0**	62.5	**4**	27	18	7200
060N_05K_30L	**0.0**	**0.0**	**0.0**	44.1	**3**	73	88	7200
060N_05K_35L	**0.0**	**0.0**	**0.0**	**0.0**	**1**	3	3	1039
060N_10K_30L	**0.0**	**0.0**	**0.0**	70.7	**57**	1903	621	7200
060N_10K_35L	**0.0**	**0.0**	**0.0**	62.9	**3**	11	8	7200
080N_05K_30L	**0.0**	**0.0**	**0.0**	34.3	**8**	18	14	7200
080N_05K_35L	**0.0**	**0.0**	**0.0**	44.2	**13**	27	20	7200
080N_10K_30L	**0.0**	**0.0**	**0.0**	42.7	**26**	48	39	7200
080N_10K_35L	**0.0**	**0.0**	**0.0**	59.2	**33**	334	56	7200
160N_05K_30L	**0.0**	**0.0**	**0.0**	73.5	301	1428	**246**	7200
160N_05K_35L	**0.0**	**0.0**	**0.0**	73.4	661	1681	**153**	7200
160N_10K_30L	**0.0**	45.0	**0.0**	71.8	2616	7200	**2041**	7200
160N_10K_35L	**0.0**	50.8	**0.0**	69.3	2466	7200	**620**	7200



(a) L_{MCF}

(b) L_{CUT}-d

Figure 6.5: Sensitivity of L_{MCF} and L_{CUT}-d to increasing values of λ_{max} on the Cabral instances. Both box plots use a logarithmic scale.



(a) Type I

(b) Type II

Figure 6.6: Impact of preprocessing and initial heuristic for L_{MCF} on the Konak instances. Both box plots use a logarithmic scale. The four boxes are labeled as follows: "P+H" uses both preprocessing and initial heuristic, "P" uses only preprocessing, "H" uses only an initial heuristic, "N" uses neither preprocessing nor the initial heuristic.

number of pricing iterations (cf. Chapter 3, where we conduct a similar experiment for a B&P algorithm for the NDPR).

In Figure 6.6 we evaluate the impact of the introduced preprocessing techniques and the initial heuristic on the performance of L_{MCF}. We omit the results for the algorithms based on (L_{CUT}) for brevity as they lead to similar conclusions. The box plots indicate that both techniques are beneficial though preprocessing appears to be a little more important due to providing larger speedups. However, when looking at the detailed results we observe that the initial heuristic is significant for proving optimality. Compared to L_{MCF} with preprocessing and initial heuristic we can solve 5 fewer instances of type I when not using the heuristic, 4 fewer instances of type I when not using preprocessing, and 6 fewer instances of type I as well as 1 fewer instance of type II when using neither preprocessing nor the initial heuristic.

6.4.6 Managerial Insights

In this section we provide more insights into managerial implications of our study. We focus on the design of translucent optical WDM networks (cf. Section 6.1) and analyze some of their major key performance indicators.

To this end, we consider a set of instances in which cost parameters are set to mimic a realistic setting in which commodities correspond to node pairs that need to communicate with each other, and arc costs are directly proportional to the arc lengths (multiplied by some factor that corresponds to cable costs per unit of distance). The cost parameters in our study are chosen so as to reflect this difference between arc and regenerator costs, the latter ones being often significantly more expensive than the arc costs (cf., e.g. [126]). The Konak instances of type I seem to be particularly relevant from this practical perspective due to their direct correlation between arc distances and arc costs. Because of the limited size of the Konak data set, we have generated further instances using the method proposed by Konak [100].

We sampled a set of five base instances for each $|V| \in \{20, 25, 30, 35, 40\}$ and $|K| \in \{5, 7, 10\}$ with $\lambda_{\max} = 30$. The relay costs were selected according to a normal distribution with $\mu = 200$ and a standard deviation of 25 to model a base cost for purchasing the relay and a slightly varying construction cost. Then, we varied the distance limit $\lambda_{\max} \in \{30, 35, 40, 45\}$ with arc costs set to $c_{ij} = d_{ij} \cdot (1 + (\lambda_{\max} - 30)/100)$, i.e., a cost increase of 1 % per unit increase in the distance limit. This is intended to model a scenario in which transmission over increased distances is only possible when relying on material of higher quality which is in turn more expensive. In total, we consider 300 new instances ranging from 60 to 696 arcs.

In what follows, we provide a cost-effectiveness analysis by comparing the costs for installing regenerators and arcs in the network. In addition, one of the major concerns when designing WDM optical networks is the power consumption. The power consumption is directly correlated with the number of deployed regenerators. We therefore analyze the energy efficiency of obtained solutions in function of the size of the network, the input demand, and the operating range of regenerators.

Cost-effectiveness analysis. The major goal of this analysis is to find out what are the potential benefits of investing into regenerators with a higher operating range. We measure these benefits in terms of the savings of the overall solution costs, when compared to the nominal solution, in which the regenerators of the minimum range ($\lambda_{\max} = 30$ in our setting) are purchased. Figure 6.7 provides a detailed overview of the overall costs of optimal DNDPR solutions. The results are sorted according to increasing value of $\lambda_{\max}$ (starting with $\lambda_{\max} = 30$ and ranging up to $\lambda_{\max} = 45$). In addition, we separately show the costs needed to install regenerators and those for installing the links in the network. For each fixed value of $\lambda_{\max}$, the number of commodities grows from 5 to 10.

We observe that the overall solution cost significantly decreases when regenerators with a higher range are deployed (i.e., increasing the value of $\lambda_{\max}$ from 30 to 45 allows for a reduction of the overall cost of up to 50 %). This can be explained by the fact that a larger distance limit allows to reduce the number of relays. Through the increased freedom in placing the relays they can be shared to a higher degree by using more direct connections. Consequently, also fewer arcs have to be installed but the impact of savings due to economizing regenerators dominates as a consequence of their higher cost. The portion of the costs spent on the regenerators decreases between 20 (5K) and 23 % (10K) when increasing the distance limit from 30 to 45. We also observe that the increase of the total costs is less sensitive to the increase of the number of commodities when regenerators with higher range are deployed. This can be particularly important in realistic scenarios in which the future demand is uncertain but it is expected to increase. In such a scenario, networks with regenerators of higher range are expected to be more resilient to the increasing demand, so that fewer upgrades might be needed in a later stage, once the uncertain demand is revealed.

Hence, our analysis clearly indicates that the decision makers need to carefully explore the cost structure of the underlying solutions and to consider different available technologies, before deciding on the type of regenerators to be deployed in the network. Sometimes, investing in more expensive hardware may result in an overall reduction of the capital expenditures (CapEx), as is shown in Figure 6.7.

Energy efficiency analysis. From the environmental and operational perspective, one of the major concerns when designing WDM networks is the power consumption which is directly correlated with the number of deployed regenerator devices [178]. In the following we study the dependency between the number of deployed regenerators and their operational range.

Figure 6.8 reports the average number of arcs and regenerators in an optimal solution. The results are grouped according to the values of $\lambda_{\max}$, ranging between 30 and 45, and according to the number of commodities, ranging between 5 and 10.

The obtained results indicate that increasing the number of commodities has a stronger effect on the total size of the network (in terms of the number of arcs) than on the number of deployed regenerators. Furthermore, this effect does not depend on the range of regenerators. For example, by increasing the number of commodities by 100 % (i.e., from 5 to 10), the number of links in the network raises by around 50 %. On the contrary, the number of deployed regenerators remains relatively stable and raises only by a single unit. A much stronger effect on the number of deployed regenerators comes from their range. So, for example, the number of regenerators can be reduced by 50 %, by deploying regenerators of range $\lambda_{\max} = 45$, when compared to the number of regenerators needed with $\lambda_{\max} = 30$.

Overall, when it comes to operational expenditures (OpEx) associated to energy costs, our result shows that significant savings in OpEx can be achieved by purchasing regenerators of higher range. Hence, there is a clear trade-off between CapEx and potential savings in OpEx that has to be carefully examined before final decisions are made.

6.5 Conclusion

We introduced two exact solution approaches for the DNDPR based on LGs. The first approach relies on a multi-commodity flow formulation ($\mathrm{L_{MCF}}$) which is pseudo-polynomial in size, and the second is a B&C approach based on the ($\mathrm{L_{CUT}}$) model with an exponential number of constraints. Both models provide extremely tight LP bounds on the considered benchmark instances. We proposed additional valid inequalities for strengthening the ($\mathrm{L_{CUT}}$) formulation and also for breaking symmetries induced by the LG structure. We investigated two approaches for adding the strengthening inequalities to the ($\mathrm{L_{CUT}}$) formulation: a static and a dynamic one. For small instances from Cabral

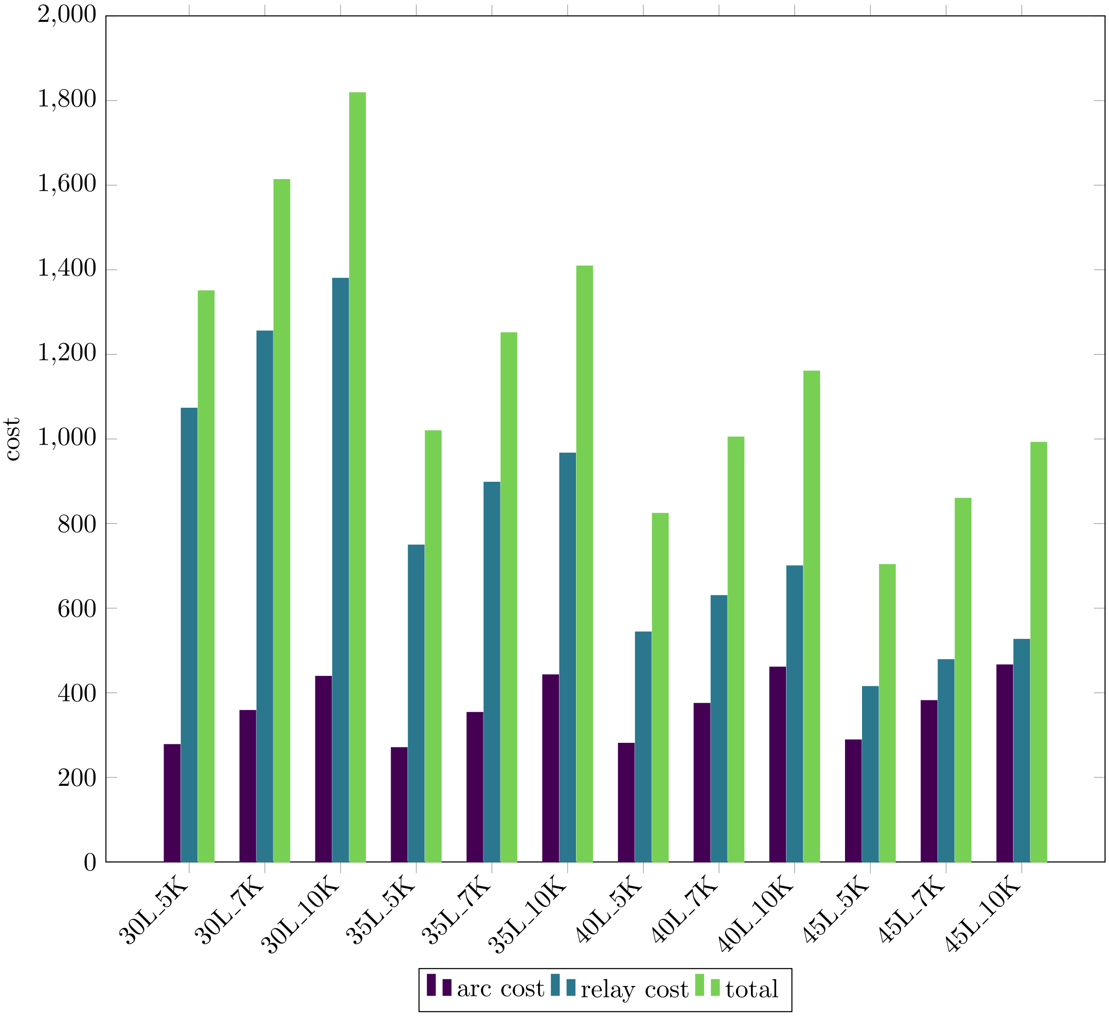


Figure 6.7: Cost-effectiveness analysis: distribution of the solution costs, separated by the cost for building the network (arc costs) and the cost for deploying relays. Each group of bars considers the mean of 25 instances, 5 for each $|V| \in \{20, 25, 30, 35, 40\}$.

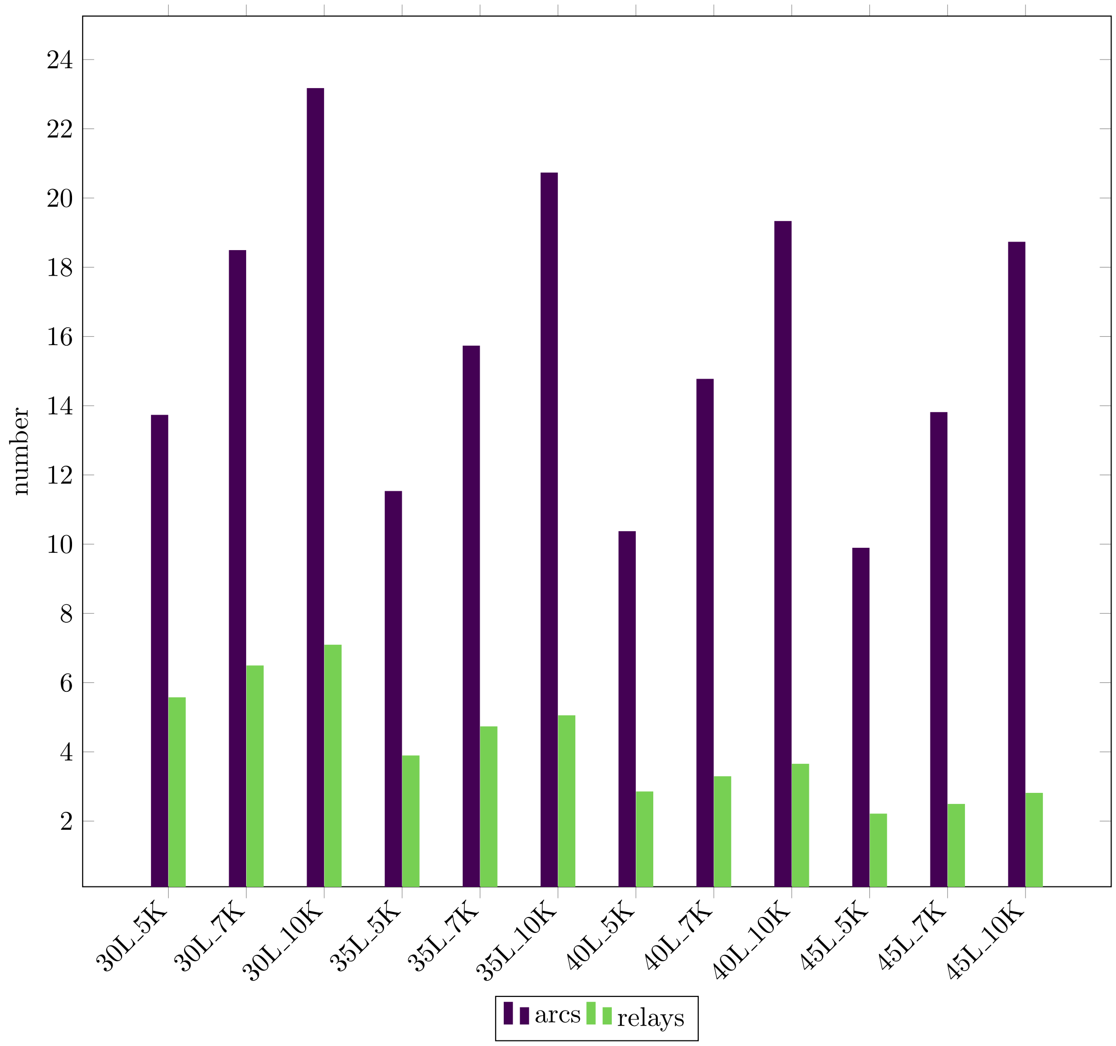


Figure 6.8: Solution structure of the newly generated Euclidean instances. Each group of bars considers the mean of 25 instances, 5 for each $|V| \in \{20, 25, 30, 35, 40\}$.

et al. [30], representing sparse 4-grid graphs with very few commodities, it turns out that the overhead of adding more inequalities a priori to the model is negligible. On the contrary, for larger and denser instances from Cabral et al. [30] and Konak [100], this overhead does not pay off, and dynamic separation of strengthening inequalities is recommended.

The overall performance of the proposed LG approaches based on ($\mathrm{L}_{\mathrm{MCF}}$) and ($\mathrm{L}_{\mathrm{CUT}}$) is comparable. In general, we observed that the former performs slightly better on sparse graphs with very few commodities, whereas the latter can be used as an alternative for larger and denser graphs and when dealing with a larger number of commodities.

Using the existing node-arc formulation as base line for a comparison with the existing approaches, we showed that our exact approaches are significantly faster than the state of the art from [110].

Our managerial study sends a strong signal to decision makers that, even though the capital expenditures might be higher when acquiring regenerators of a higher range, there are significant long term savings in terms of operating expenditures that need to be taken into account. These savings are notably related to the energy consumption, network maintenance, and the network upgrade costs caused by an increasing future demand.

CHAPTER 7

Strategies for Iteratively Refining Layered Graph Models

In the previous chapter we already considered an application of layered graphs (LGs). Their basic idea is to extend a graph along one or multiple dimensions to facilitate improved mixed integer linear programming (MILP) models, e.g., through enforcing certain constraints implicitly. Formulations on such extended graphs are known to frequently provide excellent linear programming (LP) bounds, but are often difficult to solve due to their excessive size. In Chapter 6 we avoided expensive scaling in case of fractional distances by a combination of rounding and cutting planes. Separation turned out to work well due to the small number of remaining infeasibilities. This allowed us keeping the graph size sufficiently small to obtain an effective approach. However, in many cases the size of LGs already becomes prohibitive when considering integral input data. Coarser rounding, e.g., considering only every i^{th} integer value, might be an option but often leads to many infeasibilities and therefore makes separation ineffective in practice.

In the following we investigate a framework that attempts to avoid the full size of LGs by approximating them. This is done by means of an iteratively refined relaxation that gives rise to a sequence of converging primal and dual bounds. A similar principle was investigated in Chapter 5 for a scheduling problem. The network design problems considered here differ in the fact that LGs based on graph problems convey more structural information. We use this knowledge to develop specialized methods for implementing the refinement step, i.e., the strategy according to which the LG is expanded in each step to obtain a more precise relaxation. In particular, we develop algorithms based on path computations with respect to potentially infeasible solutions obtained from MILP models formulated on intermediate LGs. Moreover, we investigate the impact of a strong heuristic component within the algorithm, both for improving convergence speed and for improving the potential of an employed reduced cost fixing step. The effectiveness of our novel refinement strategies is evaluated on two benchmark problems: the traveling salesman problem with time windows (TSPTW) and the rooted distance-constrained minimum spanning tree problem (DCMST).

This chapter has been accepted for presentation at the *11th International Workshop on Hybrid Metaheuristics*[1] in Concepción, Chile. The accompanying paper has been accepted for publication in the workshop proceedings as a full paper in the *Springer Lecture Notes in Computer Science*:

> M. Riedler, M. Ruthmair, and G. R. Raidl. Strategies for iteratively refining layered graph models. In *Hybrid Metaheuristics: 11th International Workshop, HM 2019*, Lecture Notes in Computer Science. Springer International Publishing, to appear

7.1 Introduction

In mathematical programming LGs are a well-known technique to deal with specific constraints and restrictions in problems expressed on graphs. The basic idea is to construct an extended model that considers some problem dimension explicitly to make it easier to express certain constraints or even impose them implicitly. Picard and Queyranne [137] were among the first to consider such an approach. They modeled the time-dependent traveling salesman problem by means of an extended graph that contains for each original node copies for all sequence positions at which it might be feasibly reached. Another typical application is related to distance restrictions in graphs. In such cases one can create node copies with respect to the feasible distances at which the original nodes can be reached. By omitting copies beyond the distance limit it is implicitly ensured that all paths in the extended graph adhere to the limit. If the dimension among which the original graph is extended corresponds to time, resulting LGs are sometimes called *time-expanded networks*. Such approaches are frequently considered for scheduling problems in which time is discretized to obtain so-called *time-indexed models*. For further details on LGs and associated MILP formulations see the extensive survey by Gouveia et al. [82].

The main advantage of LG formulations is that they provide a convenient modeling option while usually leading to strong LP bounds. In many cases the LG is even acyclic and allows pseudo-polynomial formulations. However, there is also an important drawback involved: LGs and the associated models are typically much larger than simpler formulations on the original input. Frequently, this leads to models which are computationally impractical for reasonable problem sizes. However, often it is the case that already a subgraph of the full LG would suffice to encode an optimal solution. Several researchers used this observation to construct iterative algorithms that successively approximate the full LG until an optimal solution is found. This is usually done by omitting node copies and redirecting

---

[1] http://hm2019.ing.udec.cl

arcs. Among the first were Wand and Regan [167] who consider LG formulations for a pickup and delivery problem with time windows. In particular, they propose a relaxed formulation and a heuristic component that considers a subset of the feasible solutions. Those two formulations are successively extended until their bounds match, proving optimality. Ruthmair and Raidl [155] suggested such an iterative approach for the rooted DCMST. Another successful application of a similar algorithm was proposed by Dash et al. [48] for solving the TSPTW. Their approach differs slightly from the former two as it refines the reduced LG only based on solving LP relaxations in a first stage. The final reduced LG is then used for solving an MILP in which the remaining infeasibilities are tackled by cutting planes. Further iterative refinement approaches in the network design area were considered by Macedo et al. [118], Boland et al. [27, 26], and Clautiaux et al. [39].

Algorithms of this type that contain a component for obtaining heuristic solutions provide an eventually converging sequence of primal and dual bounds. Therefore, such an algorithm can also be terminated prematurely to obtain a high-quality primal solution together with a dual bound.

All previous works in this area have in common that they consider only a single strategy for extending the reduced LG in each iteration without evaluating alternatives. The employed techniques reach from rather simple approaches to more complex algorithms. In Chapter 5 we presented an extensive evaluation of different refinement techniques for a resource-constrained project scheduling problem (RCPSP). However, scheduling problems are somewhat special when it comes to the refinement step. In an aggregated set of time instants it is usually not clear what is the best/most promising option to reveal infeasibilities. Network design problems appear to be more accessible in this respect. An LG relaxation is typically obtained by redirecting arcs due to omitted layers which causes the arcs to no longer represent correct lengths. This makes it straightforward how an arc can be corrected to the full extent. Moreover, we can use the knowledge regarding the true length of the arcs to get further insight from intermediate solutions. This is a crucial part of the algorithm and evaluating promising alternatives seems to be worthwhile.

In the following we start by introducing the necessary terminology of LGs in terms of an example problem: the TSPTW. In particular, we discuss how reduced LGs can be obtained whose associated MILP models provide either primal or dual bounds. Then, we propose a generic refinement algorithm including several enhancements. Afterwards, we explain the specific refinement strategies and evaluate them in our computational experiments. In addition to existing strategies from the literature, we suggest new ones that aim at extracting more information from intermediate solutions. In the computational study we evaluate the discussed refinement strategies on several benchmark sets for the TSPTW. To show how the strategies behave on a structurally different problem we also conduct experiments for the rooted DCMST.

7.2 Mathematical Formalization

In the following we describe the construction of an LG and an associated MILP model. The process is exemplified in terms of the TSPTW. Afterwards, we characterize reduced LGs that serve as basis for the refinement algorithm introduced in the next section.

Notational remarks. For graph $G = (V, A)$ and node subset $S \subseteq V$ let $\delta^+(S) = \{(i, j) \in A \mid i \in S,\, j \notin S\}$ be the set of outgoing and $\delta^-(S) = \{(j, i) \in A \mid i \in S,\, j \notin S\}$ be the set of incoming arcs. To simplify notation we omit the set braces for singletons S. Solution vectors are indicated by a superscript "$*$".

7.2.1 Traveling Salesman Problem with Time Windows

The TSPTW is defined on a directed graph $G = (V, A)$ with node set $V = \{\alpha, 1, \ldots, n, \omega\}$, associated arc costs $c\colon A \to \mathbb{Z}_{\geq 0}$, and travel times $t\colon A \to \mathbb{Z}_{>0}$. As in [48], we represent the depot by two distinct nodes α and ω in order to model a tour starting and ending at the depot as path. Each node $i \in V$ is associated with a time window $[r_i, d_i]$ with $r_i \leq d_i$. Service times for the nodes can be incorporated into the arc distances and are therefore not considered separately. The goal is to find a least cost Hamiltonian path through V starting at α and ending at ω such that all nodes are visited within their time windows. Waiting at nodes is allowed in case of early arrival.

7.2.2 Layered Graph Model

We consider the layered digraph $G_{\mathrm{L}} = (V_{\mathrm{L}}, A_{\mathrm{L}})$. Initially, node set $V_{\mathrm{L}} = \{i_l \mid i \in V,\, l \in [r_i, d_i]\}$ contains all node copies that are feasible with respect to the time windows. Thereby, node copy $i_l \in V_{\mathrm{L}}$ represents a copy of node $i \in V$ reached at distance l. To get an abstraction for connecting the layered node copies we introduce distance function $\theta(i_l, j) := \max(r_j, l + t_{(i,j)})$ that provides the layer at which node j is reached when starting at node i at layer (distance) l. The obtained arc set is $A_{\mathrm{L}} = \{(i_l, j_m) \mid i_l, j_m \in V_{\mathrm{L}},\, (i, j) \in A,\, \theta(i_l, j) = m\}$. To obtain a smaller graph we remove all unnecessary node copies—and their incident arcs—that cannot be reached from depot copy α_{r_α}[2]. An example in which node 3_1 is not reachable from α_0 is given in Figure 7.1.

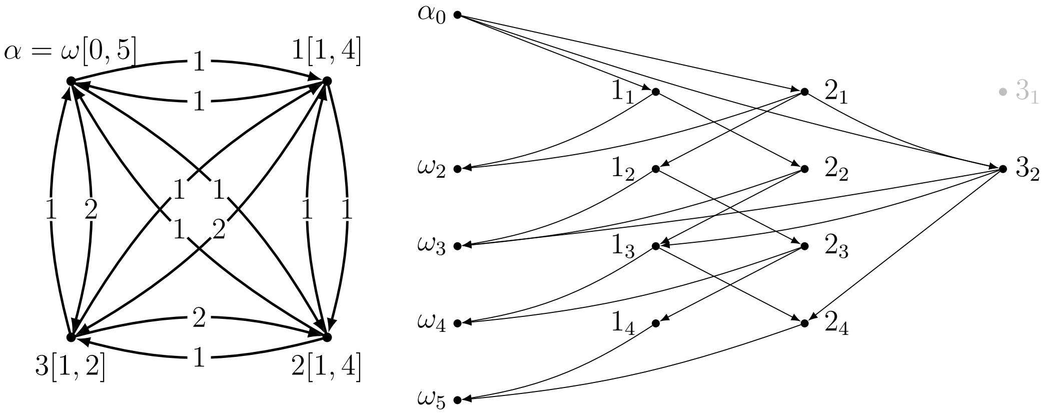
We model the TSPTW in terms of binary arc variables x_a, for $a \in A$, indicating which arcs of the original graph are part of the tour, and non-negative arc variables z_a for the LG. Observe that this problem could also be modeled without the original graph variables. However, these variables are convenient for imposing certain strengthening inequalities and beneficial for the reduced cost fixing explained in Section 7.3.

$$\text{(TSPTW-L)} \quad \min \sum_{a \in A} c_a x_a \tag{7.1}$$

$$\text{subject to} \quad \sum_{i_l \in V_{\mathrm{L}}} \sum_{a \in \delta^-(i_l)} z_a = 1 \qquad \forall i \in V \setminus \{\alpha\}, \tag{7.2}$$

---

[2] In the following we assume this step to be completed when referring to the full LG G_{L}.



(a) Input graph (b) Full layered graph

Figure 7.1: Example of a layered graph for a TSPTW instance with $d = c$.

$$\sum_{a \in \delta^+(i_l)} z_a = \sum_{a \in \delta^-(i_l)} z_a \qquad \forall i_l \in V_\mathrm{L}, \tag{7.3}$$

$$\sum_{(i_l, j_m) \in A_\mathrm{L}} z_{(i_l, j_m)} = x_{(i,j)} \qquad \forall (i,j) \in A, \tag{7.4}$$

$$\boldsymbol{x} \in \{0,1\}^{|A|},\ \boldsymbol{z} \in \mathbb{R}_{\geq 0}^{|A_\mathrm{L}|}. \tag{7.5}$$

The model presented above can be strengthened[3] by well-known cut-set inequalities of the following form:

$$\sum_{a \in \delta^-(W)} x_a \geq 1 \qquad \forall W \subseteq V \setminus \{\alpha\},\ W \neq \emptyset. \tag{7.6}$$

Stronger cut-set inequalities can be specified with respect to the z variables (see [82]):

$$\sum_{a \in \delta^-(W)} z_a \geq 1 \qquad \forall W \subseteq V_\mathrm{L} \setminus \{\alpha_{r_\alpha}\},\ W \neq \emptyset,\ \exists v \in V : \{v_l \in V_\mathrm{L}\} \subseteq W. \tag{7.7}$$

Both sets of cut-set inequalities are of exponential size and require dynamic separation in practice. Although the original graph cut-set inequalities are known to be weaker than their LG counterpart, they are still worth considering due to faster convergence as a result of the smaller size of the original graph.

7.2.3 Reduced Layered Graphs

The full LG defined above guarantees that the associated MILP model, denoted by TSPTW-L(G_L), contains all feasible solutions that can be realized in the original graph.

---

[3] In case of zero-distance cycles, these inequalities become mandatory.

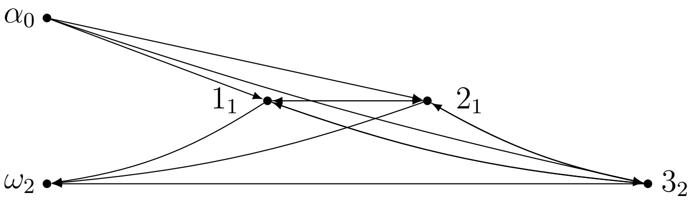


Figure 7.2: Example of a dual layered graph with respect to the TSPTW instance provided in Figure 7.1.

However, depending on the number of layers and the density of the original graph we often end up with a problematic model size. Smaller graphs can be extracted from the full LG by either giving up optimality or feasibility. For pragmatic reasons we require each reduced LG $G'_{\mathrm{L}} = (V'_{\mathrm{L}}, A'_{\mathrm{L}})$ to contain at least one copy for each node of the original graph, i.e., $V = \{i \mid i_l \in V'_{\mathrm{L}}\}$.

Due to the omitted node copies, arcs are redirected. We say that an arc (i_l, j_m) is *shortened* (*lengthened*) if there exists an arc (i_l, j_k) in the full LG with $m < k$ $(m > k)$, otherwise it has the *correct* length.

Dual Layered Graphs

A *dual* LG $G_{\mathrm{dL}} = (V_{\mathrm{dL}}, A_{\mathrm{dL}})$ is obtained by considering only a subset of the layered node copies $V_{\mathrm{dL}} \subseteq V_{\mathrm{L}}$ inducing the reduced arc set

$$\begin{aligned}A_{\mathrm{dL}} = \{(i_l, j_m) \mid i_l, j_m \in V_{\mathrm{dL}},\, (i,j) \in A,\, m \leq \theta(i_l, j),\\ \nexists m' \; (m < m' \leq \theta(i_l, j) \wedge j_{m'} \in V_{\mathrm{dL}})\}.\end{aligned}$$

In short, this means that if a layered node is present in V_{dL} but the target of its outgoing arc according to V_{L} is not, then we use the copy of the target node at the maximum layer no larger than the originally used copy and omit the arc if such a copy does not exist. An example is provided in Figure 7.2.

In order to guarantee that the associated MILP model is a relaxation we only consider node subsets $V_{\mathrm{dL}} \subseteq V_{\mathrm{L}}$ such that $i_l \in V_{\mathrm{dL}} \wedge (i_l, j_m) \in A_{\mathrm{L}} \implies \exists m' \; (m' \leq m \wedge (i_l, j_{m'}) \in A_{\mathrm{dL}})$. This ensures that we loose no connections that were present in the original graph. By using only shortened and correct arcs we never arrive at a node on a higher layer than in the full LG. As a result TSPTW-L(G_{dL}) is a relaxation with respect to the original problem. Consequently, the LP relaxation of TSPTW-L(G_{dL}) yields a dual bound. Moreover, if an optimal integral solution to TSPTW-L(G_{dL}) is feasible (on the x variables) with respect to the original problem, then it is guaranteed to be optimal. Observe that the dual LG is—opposed to the full LG—usually not acyclic. Therefore, separating cut-set inequalities is necessary to obtain a connected solution.

Primal Layered Graphs

A primal LG $G_{\mathrm{pL}} = (V_{\mathrm{pL}}, A_{\mathrm{pL}})$ is obtained by considering only a subset of the layered node copies $V_{\mathrm{pL}} \subseteq V_{\mathrm{L}}$ and an associated induced arc set

$$A_{\mathrm{pL}} = \{(i_l, j_m) \mid i_l, j_m \in V_{\mathrm{pL}},\, (i,j) \in A,\, m \geq \theta(i_l, j),\\ \nexists m' \; (\theta(i_l, j) \leq m' < m \wedge j_{m'} \in V_{\mathrm{pL}})\}.$$

This time we redirect arcs to the node copy at the minimum layer at least as large as the original one. Therefore, the primal LG turns its associated MILP model into a heuristic as it may exclude feasible solutions—possibly to the extent that no solutions remain. A feasible solution to TSPTW-L(G_{pL}) provides a primal bound but cannot be shown to be optimal on its own—not even if all layered arcs associated with the selected z variables have the correct length.

7.2.4 Other Problems

The definitions provided above can be easily adjusted to other problems. To cover the rooted DCMST (see [78])—for which we also perform experiments in Section 7.4—it suffices to redefine the distance function to $\theta(i_l, j) := l + d_{(i,j)}$, i.e., waiting is not permitted/necessary. The problem's distance restriction can be imagined as time window for each node with a lower bound of zero and an upper bound equal to the global distance limit. A suitable MILP model can then be obtained by taking the model for the TSPTW and replacing constraints (7.3) by

$$z_{(i_l,j_m)} \leq \sum_{(k_h,i_l) \in \delta^-(i_l): k \neq j} z_{(k_h,i_l)} \quad \forall i_l \in V_{\mathrm{L}},\, \forall (i_l, j_m) \in \delta^+(i_l). \tag{7.8}$$

7.3 Algorithmic Framework

In this section we describe our iterative refinement algorithm. In particular, we consider different refinement strategies that are used to iteratively extend an initially small dual LG.

To simplify the description in what follows, we make some assumptions on the considered input problem. We focus on problems for which each node must be connected to a designated source node. This guarantees that we can perform the necessary path computations in the LG for some of the refinement strategies presented in Section 7.3.2. Consequently, suitable connectivity inequalities must be available (e.g., cut-set inequalities (7.7) for the TSPTW). This assumption might seem restrictive at first but actually covers a large variety of problems. Depot-based routing problems as well as most network design problems are compatible with this restriction. In addition, we assume that the model is specified as minimization problem and includes at least design variables $\boldsymbol{x}$ for the original graph arcs and design variables $\boldsymbol{z}$ for the LG variables—further auxiliary variables are of course possible. Note that these conditions are more strict than necessary but being more general would go beyond the scope of this work.

Algorithm 7.1: Iterative refinement algorithm (IRA)

```
while termination condition not met do
    solve LP; stop if gap closed
    refine LG
    if solution is integer and feasible then terminate    // optimal solution
    apply primal heuristic; stop if gap closed
    apply reduced cost fixing
    if graph could be refined then continue with next iteration
    solve IP
    refine LG
    if solution is feasible then terminate                // optimal solution
    apply primal heuristic; stop if gap closed
end
```

7.3.1 Iterative Refinement Algorithm

The main idea of our iterative refinement algorithm (IRA) is to start with a small dual graph and solve the associated MILP model or its corresponding LP relaxation, respectively. The result is then used to either prove optimality or—if this cannot be done—to obtain a larger dual graph (closer to the full LG) for repeating the procedure. This step of adding not-yet-present node copies of the full LG to the dual LG is called *refinement*. If the refinement adds at least one new node copy in each iteration, then it is guaranteed that the algorithm terminates with an optimal solution in finitely many iterations since the dual graph eventually converges to the full LG.

Algorithm 7.1 provides the detailed procedure. The mentioned gap refers to the absolute difference between the current dual (db) and primal (pb) bounds and is considered to be closed if $db \geq pb$. In the beginning we need an initial dual LG. This step depends on the problem at hand. For the TSPTW—and many other problems—a minimal starting graph that satisfies the restrictions imposed above can be obtained by considering for each original graph node the copy at the smallest feasible layer. Based on this initial dual LG we solve the LP relaxation of the associated MILP model. The obtained solution value is a dual bound and can be used to prove optimality if a primal bound is available. In order to get a more meaningful solution for the subsequent refinement process, we assume the LP to be extended by connectivity inequalities. If optimality could not be proven yet, we use a refinement algorithm to identify possible infeasibilities in the relaxed solution. If infeasibilities could be detected, we add further nodes to the graph to reveal them. Otherwise, we test whether the obtained LP solution is integral. An integral solution that is feasible must be optimal according to the construction of the dual LG. A fractional solution, on the other hand, might prevent the refinement algorithm from detecting remaining infeasibilities. Therefore, we solve the MILP model in the following. However, before doing this, we can apply a heuristic (guided by the current fractional solution) to obtain a primal bound to possibly close the gap and prove optimality. Furthermore, we can use the obtained primal bound to attempt *reduced cost fixing* with respect to the x variables of the MILP model. To this end let db be the current solution value of the LP relaxation, pb the current primal bound, $\boldsymbol{x}^*$ the solution vector of the original graph variables, and $\boldsymbol{x}^{\mathrm{r}}$ the vector of reduced costs of the x variables. For each arc $a \in A$ we consider two cases. If $x^*_a = 0 \land db + x^{\mathrm{r}}_a \geq pb$, we can remove arc a from the input graph and consequently all its copies in any LG. On the other hand, if $x^*_a = 1 \land db - x^{\mathrm{r}}_a \geq pb$, we know that arc a must be part of an optimal solution and its associated variable can therefore be fixed to one in subsequent iterations. When the algorithm is already close to convergence, reduced cost fixing might fix a sufficient number of variables to zero such that the model becomes infeasible, proving optimality of the solution that provided the current primal bound.

Unless one of the previous considerations allowed proving optimality, we are now in the situation that the (fractional) LP solution does not allow the refinement algorithm to identify the remaining infeasibilities. In this unfortunate case we have to take the additional computational burden and solve the MILP to optimality. If the integral solution is feasible, we proved optimality. Otherwise, we apply the primal heuristic once more before solving the LP relaxation according to the refined dual LG.

Primal graph heuristic. Steps 5 and 11 of Algorithm 7.1 can in principle be realized by any suitable heuristic. Problem-dependent algorithms typically provide better solution quality but are sometimes tedious to implement and often have to be replaced completely if a slightly different problem variant is considered. A more convenient problem-independent way to obtain heuristic solutions is to use the MILP formulation on the primal LG. We construct the primal LG by taking the node set of the dual LG and add for each node a copy at the maximum feasible layer. This enables us to benefit from the iterations made so far while reducing the risk of obtaining a graph that encodes no feasible solution. Primal graph heuristics of this type were considered in [39, 154, 155, 167].

7.3.2 Refinement Procedures

The perhaps most crucial part of IRA is the refinement step. Solutions with respect to the current dual LG typically contain multiple infeasibilities and it is usually not clear how they can be handled most efficiently. In this context one has to deal (among others) with the following important questions: (1) for which nodes should we add further copies, (2) how many copies should be added, and (3) on which layers should the copies be inserted. Answering those questions typically involves keeping a suitable balance between the growth of the dual LG and the number of iterations IRA has to complete before proving optimality. The latter is quite important as it determines how often the MILP solver has to be invoked which is usually the most time-consuming part of the algorithm. On the other hand, the time each invocation takes increases with the size of the associated dual LG.

Full Infeasible Arc Refinement (FAR)

The probably most straightforward refinement strategy simply refines all nodes that are part of the current solution. To this end, we take all layered arcs whose associated solution value is non-zero. All shortened arcs are fully corrected by adding the appropriate target node to the dual LG. To avoid refining already feasible solutions, we check if the solution with respect to the x variables is feasible before starting the refinement process. This refinement procedure was employed in [26, 118, 154, 155].

Infeasible Path Refinement (PR)

Instead of considering all arcs, we only consider those that are most relevant from the structural perspective. We start by constructing an auxiliary LG that is obtained by taking all arcs of the dual LG with associated non-zero z variable value in the current solution. Based on this graph we compute a shortest path, using distances weighted by $1 - z^*_a$, to each node and determine the effective distance at which the node would be reached. If this distance is incompatible with the node's time window, we compute a refinement. This is done by traversing the path backwards and refining each arc as done for FAR stopping once we reach a node for which the effective distance is equivalent to its layer.

Repeated Infeasible Path Refinement (RPR)

We start by performing PR. In a subsequent step, we check for each formerly infeasible path if it is still contained in the adjusted dual LG—traversing different node copies but still reaching the target node after its time window when considering the effective distance of the path—and repeat the refinement step until the path is no longer present.

Single-Copy Infeasible Path Refinement (SPR)

Especially in later iterations the dual LG contains multiple layered copies with respect to each node of the original graph. We perform the same approach as done in PR, however, for each node of the original graph we only compute a refinement for the path reaching the node at the highest effective distance, i.e., the most infeasible path.

Minimum Sum of Negative Waiting Times (DASH)

This strategy was developed by Dash et al. [48]. Similar to the other techniques they consider the subgraph G'_{dL} induced by non-zero z variable values. If a node copy v_l is reached by shortened arcs, a new node copy is added that minimizes the sum of *negative waiting times*. The negative waiting time of an arc is essentially the amount by which it was shortened, i.e., if $(i_l, j_m) \in G'_{\mathrm{dL}}$ and $(i_l, j_k) \in G_{\mathrm{L}}$, then the negative waiting time is $k - m$. A new layered copy of node v is added at the layer λ that achieves the minimum when computing the sum of negative waiting times weighted by the associated arcs' solution values. Let $\boldsymbol{z}^*$ be the current solution vector and let

$I_{v_l} = \{((j_m, v_l), l') \mid (j_m, v_l) \in A_{\mathrm{dL}}, (j_m, v_{l'}) \in A_{\mathrm{L}}\}$. Then we seek the layer λ that minimizes $\mu(v_l, \lambda) = \sum_{(a,l') \in I_{v_l}: l' < \lambda} (l' - l) z_a^* + \sum_{(a,l') \in I_{v_l}: l' \geq \lambda} (l' - \lambda) z_a^*$. Dash et al. do not indicate which value is used if there are multiple options for λ that achieve the minimum. Function μ is piece-wise linear for a fixed v_l and changes slope only at points at which at least one arc arrives at the correct distance. Therefore, it makes sense to restrict the procedure to such values, i.e., $l' - \lambda$ is zero for at least one element from I_{v_l}, as this guarantees to reduce the number of shortened arcs. However, this still might leave several options. In preliminary experiments we tested using either the smallest or the largest value of λ that achieves the minimum. The performance was roughly the same with a slight advantage for the latter.

Again we check feasibility with respect to the original graph variables to avoid superfluous refinements.

7.4 Computational Study

Our algorithms are implemented in C++ using CPLEX 12.8.0 as general-purpose MILP solver. All experiments have been performed in single thread mode with default parameter settings. For performance reasons the implicitly integral LG variables z are implemented as binary variables together with a cost-based branching priority to focus on the original graph variables. Experiments have been executed on an Intel Xeon E5540 machine with 2.53 GHz. The computation time limit has been set to 7200 seconds and the memory limit to 8 GB RAM. To test our framework we consider two benchmark sets for the TSPTW from http://lopez-ibanez.eu/tsptw-instances. The first set is from Ascheuer et al. [6] and contains 50 instances while the second one was proposed in Dumas et al. [56] and contains 135 instances. In addition, we also tested on instances for the rooted DCMST by Gouveia et al. [78]. Experiments were limited to the subset of 60 “TE” instances with distances ranging to 10, 100, and 1000 as the other instances turned out to be too easy. The TSPTW instances were preprocessed as described in [6] and the DCMST instances as described in [154].

We want to emphasize that our aim is to compare the different refinement strategies on a common basis and not to beat the state of the art. Achieving the latter would require further problem-specific tuning and incorporation of additional strengthening inequalities which is not the focus of this work.

7.4.1 Experiments

In the upcoming tables we present averages for gaps, computation times, the number of iterations in which the LP relaxation was solved (itr), the number of iterations in which the MILP was solved (itr-ip) as well as the number of nodes and arcs in the final LGs. Instances that terminated due to the memory limit are omitted when computing averages and those that ran into the time limit are considered with a value of 7200 seconds. Gaps are computed by $(pb^* - db)/pb^*$ where db is the dual bound of the respective run and

Table 7.1: Results on the TSPTW instances by Ascheuer at al. [6].

Algorithm	Gap [%]	t [s]	#itr	#itr-ip	$\|V\|$	$\|A\|$	#tl	#ml	#opt
MIP	20.24	3800	-	-	78437	1339490	21	5	24
IRA_FAR	0.09	2230	**17.9**	**0.0**	547	9820	14	0	36
IRA_DASH	0.08	2114	18.8	**0.0**	531	9546	13	0	37
IRA_PR	0.08	2090	31.4	3.5	**398**	**7663**	13	0	37
IRA_RPR	0.08	**2007**	22.1	2.5	417	8010	12	0	**38**
IRA_SPR	0.08	2080	32.6	3.6	399	7761	12	0	**38**
IRA_FAR_HS	0.08	2215	**15.1**	**0.0**	503	9248	14	0	36
IRA_DASH_HS	0.08	2064	15.5	**0.0**	480	8872	13	0	37
IRA_PR_HS	0.08	2038	27.9	3.2	**376**	**7379**	12	0	**38**
IRA_RPR_HS	0.08	**1937**	19.1	1.9	394	7710	12	0	**38**
IRA_SPR_HS	0.08	2068	29.2	3.1	377	7398	12	0	**38**
IRA_FAR_HS_RCF	0.08	2047	**15.3**	**0.0**	492	8340	13	0	37
IRA_DASH_HS_RCF	0.08	1935	15.4	**0.0**	475	7985	13	0	37
IRA_PR_HS_RCF	0.08	2096	29.2	3.6	**384**	**6876**	13	0	37
IRA_RPR_HS_RCF	0.08	**1933**	19.2	2.0	399	7190	12	0	**38**
IRA_SPR_HS_RCF	0.08	2001	30.6	3.3	**384**	6948	12	0	**38**

pb^* is the best primal bound known for the respective instance. The remaining columns report the number of runs that ran into the time limit (tl) or the memory limit (ml), respectively, and the number of instances solved to proven optimality (opt).

For the TSPTW we consider each refinement strategy in three variants: without a primal component, with an initially provided primal solution (“HS”), and with an initially provided primal solution and reduced cost fixing activated (“HS_RCF”). We do this in order to show two things: (1) the benefits of a strong primal component and (2) the potential of reduced cost fixing if a high-quality solution is available. High-quality heuristic solutions for the Ascheuer instances were obtained from http://lopez-ibanez.eu/tsptw-instances and optimal solutions for the instances by Dumas et al. were obtained from http://homepages.dcc.ufmg.br/~rfsilva/tsptw.

Table 7.1 reports our results on the instances by Ascheuer et al. [6]. The first observation is that directly solving the MILP on the full LG (MIP) is not effective. The size of the associated model leads either to problems with the memory limit or to long runs that can frequently not be completed within the time limit. Consequently, the remaining gap is quite large with more than 10 % on average. All variants of our refinement algorithm perform much better. The most striking difference is that we deal with considerably smaller graphs that help to avoid any memory issues. This enables us to solve significantly more instances to optimality. Among the various refinement strategies we observe that the naive approach (FAR) solves the fewest instances to optimality while leading to the largest graph sizes. The approach by Dash et al. [48] works noticeably better and solves one more instance to optimality but requires comparatively large graphs. Our new path-based strategies solve the largest number of instances to optimality. The drawback

Table 7.2: Results on the TSPTW instances by Dumas et al. [56].

Algorithm	Gap [%]	t [s]	#itr	#itr-ip	$\lvert V\rvert$	$\lvert A\rvert$	#tl	#ml	#opt
MIP	0.42	2000	-	-	3276	37069	29	0	106
IRA_FAR	0.00	250	10.5	0.2	285	3597	1	0	134
IRA_DASH	0.00	240	10.7	**0.1**	282	3549	1	0	134
IRA_PR	0.00	80	13.1	4.0	142	1719	0	0	**135**
IRA_RPR	0.00	**55**	**9.2**	3.0	145	1773	0	0	**135**
IRA_SPR	0.00	93	13.2	4.1	**141**	**1711**	0	0	**135**
IRA_FAR_HS	0.00	110	**8.2**	**0.1**	256	3207	0	0	**135**
IRA_DASH_HS	0.00	123	8.3	**0.1**	251	3135	0	0	**135**
IRA_PR_HS	0.00	65	12.3	3.5	139	1681	0	0	**135**
IRA_RPR_HS	0.00	**45**	8.4	2.5	139	1691	0	0	**135**
IRA_SPR_HS	0.00	73	12.4	3.5	**138**	**1674**	0	0	**135**
IRA_FAR_HS_RCF	0.00	44	**8.0**	**0.1**	255	2247	0	0	**135**
IRA_DASH_HS_RCF	0.00	45	8.3	**0.1**	250	2187	0	0	**135**
IRA_PR_HS_RCF	0.00	56	12.4	3.5	**138**	1395	0	0	**135**
IRA_RPR_HS_RCF	0.00	**42**	8.3	2.5	**138**	1401	0	0	**135**
IRA_SPR_HS_RCF	0.00	55	12.4	3.5	**138**	**1387**	0	0	**135**

of these rather careful and minimalist approaches is that they require a higher number of iterations to converge, even including some iterations where the MILP has to be solved, which is not necessary for FAR and DASH. Nevertheless, we observe the smallest average computation times for these strategies. The more slowly growing graphs outweigh the higher number of iterations through the smaller associated models. Among the three path-based approaches, we see that RPR performs best.

Providing high-quality initial solutions improves all variants of IRA alike. We observe a decrease in the number of iterations as well as the final graph sizes. This shows that a tight dual bound is sometimes obtained before feasibility can be established through further refinement steps. Enabling reduced cost fixing helps to improve the results further. For strategy PR we observe a minor slowdown compared to the variant in which only the initial solution is provided. The reason is that solution quality and refinement quality are not directly correlated. A weaker solution might lead to a very successful refinement in a subsequent iteration that is not reached by a better solution. The slowdown, however, is not dramatic and we achieve the smallest final graph size with this approach.

In Table 7.2 we provide the results on the instances by Dumas et al. [56]. Compared to the Ascheuer instances this set features much narrower time windows (100 at most). Therefore, the MILP on the full LG performs considerably better. Although it no longer faces problems with the memory limit, it is still not able to solve all instances to optimality within the time limit. The remaining gap is rather small but could also not be closed completely. In terms of computation times we again observe a clear advantage for IRA. The performance of the different refinement strategies is comparable to what we observed

Table 7.3: Results on the hard DCMST instances (TE) by Gouveia et al. [78].

Algorithm	Gap [%]	t [s]	#itr	#itr-ip	$\lvert V\rvert$	$\lvert A\rvert$	#tl	#ml	#opt
MIP	0.72	3063	-	-	23833	529314	13	13	34
IRA_FAR	0.08	2481	21.9	**0.0**	474	10827	9	0	51
IRA_DASH	0.05	2118	22.4	0.1	473	10816	6	0	54
IRA_PR	0.05	2555	23.6	0.2	418	9703	7	0	53
IRA_RPR	0.04	**1913**	**18.0**	0.1	481	10767	4	0	**56**
IRA_SPR	0.06	2601	24.3	0.2	**415**	**9640**	10	0	50
IRA_FAR_PHeu	0.07	2211	21.5	**0.0**	471	10769	6	0	54
IRA_DASH_PHeu	0.05	2058	21.9	**0.0**	472	10793	5	0	55
IRA_PR_PHeu	0.04	2256	23.0	0.1	417	9678	6	0	54
IRA_RPR_PHeu	0.04	**1680**	**17.8**	0.1	482	10781	4	0	**56**
IRA_SPR_PHeu	0.04	1987	23.6	0.2	**414**	**9632**	4	0	**56**
IRA_FAR_PG	0.08	2453	21.2	**0.0**	469	10709	9	0	51
IRA_DASH_PG	0.05	2185	21.5	**0.0**	467	10675	5	0	55
IRA_PR_PG	0.04	2269	23.3	0.2	416	9659	6	0	54
IRA_RPR_PG	0.04	**1816**	**17.6**	0.1	479	10702	4	0	**56**
IRA_SPR_PG	0.07	2553	23.9	0.2	**413**	**9606**	9	0	51

for the Ascheuer instances. Strategies FAR and DASH require larger graphs but converge within fewer iterations. The path-based approaches, on the other hand, lead to much smaller final graphs but also have to complete some iterations in which the MILP is solved. Providing an initial primal solution again improves the results significantly. This time reduced cost fixing provides a consistent improvement and does not suffer from side effects. It is even effective enough to almost improve the slower refinement strategies to the level of the better ones through variable fixes that significantly reduce the LG size.

Finding feasible solutions to the TSPTW is $\mathcal{NP}$-hard (see Ascheuer et al. [6]) but can be done in (pseudo-)polynomial time for the rooted DCMST. Therefore, we use this problem to show the performance of a simple problem-specific heuristic ("PHeu") in comparison to the general purpose heuristic based on the primal LG ("PG"). The considered problem-specific heuristic iteratively computes a resource constrained shortest path to a still unreached node farthest from the source. The costs of the thereby added arcs are set to 0 for the next iteration and the procedure is stopped once all nodes are connected to the source. We use the solution on the x variables as guidance by operating on adjusted costs weighted by $1 - x^*_a$. We do not use reduced cost fixing here to avoid side effects that could influence the results.

The MILP model on the full LG once more solves the fewest instances to optimality while being the slowest algorithm on average. The reason why the MILP is not that far off this time is that the considered instance set considers also small distance limits. For small and medium distance limits the MILP is competitive while it is clearly outperformed for the larger ones or cannot be solved due to the memory limit. Again, all variants of the iterative approach outperform the pure MILP approach for the larger distance restrictions. Strategies FAR and SPR do not work as well as the other strategies. The former appears to refine too unstructured while the latter does not make enough progress resulting in a comparatively high number of iterations. Among the remaining three variants we observe that RPR works best. Although it leads to larger graphs than the other path-based strategies, it turned out to be quite fast. Apparently, the repeated refinement helps to significantly reduce the number of iterations which compensates for the larger graph size.

Adding heuristics significantly decreases computation times and allows solving further instances to proven optimality. The primal LG heuristic turns out to be a valuable alternative to the problem-specific one. Nevertheless, we want to point out that it strongly depends on the problem whether the generic approach works well. Preliminary experiments for the TSPTW showed that it can be difficult to obtain feasible solutions if the underlying problem is challenging in this respect. Node copies corresponding to an initial heuristic solution might be inserted into the LG to resolve these issues.

Finally, we applied the one-tailed Wilcoxon signed-rank test for RPR and each other refinement strategy (without heuristics or reduced cost fixing). The alternative hypothesis that RPR is faster was assumed with a significance level of 0.05, except for the instances by Ascheuer where PR, SPR, and DASH performed too similar, mainly due to the comparatively high number of unsolved instances.

7.5 Conclusion and Future Work

In this chapter we considered a general framework for iteratively refining a reduced LG. Based on solutions to an associated MILP formulation and heuristically obtained primal bounds the approach converges towards proven optimality if given enough time. In particular, we focused on one of the crucial points of such algorithms which did not receive much attention in previous works: the refinement step that extends the LG in each iteration. We investigated strategies from the literature and suggested new path-based alternatives. Through our experiments on two benchmark problems—the TSPTW and the rooted DCMST—we could show that the previous approaches work reasonably well but still leave room for improvement. The path-based approaches are able to solve a higher number of instances to optimality while leading to smaller LGs in the final iteration. We also showed that a strong heuristic component is important for the algorithm to converge faster and to provide high-quality (intermediate) solutions. In addition, these solutions can be used to apply reduced cost fixing.

A problem-independent heuristic was shown to be competitive with a simple problem-specific one. Future work could put more effort into this component to improve the obtained results. In this work we focused on refinement strategies for the reduced LG that is used to compute dual bounds. However, one may also consider refinements based on the LG that is used to obtain heuristic solutions.

For brevity, we restricted the discussion to problems with a designated source node to which all other nodes must be connected. However, the method can in principle be applied to any network design problem for which feasibility of the relaxation may be checked through path computations. Interesting problems are those whose resource dependencies can be naturally modeled through LGs. This especially includes problems with resource-dependent (non-linear) costs, e.g., time-dependent travel times. If many layers are present of which only few are assumed to be traversed, the iterative algorithm is expected to work particularly well. In general, the approach does not work for applications represented by a cyclic LG because the described dual LGs not necessarily represent a relaxation for them. Typical examples are pickup and delivery problems with increasing and decreasing load along the route as well as the energy state in electric vehicle routing.

To be comparable to the state of the art further tuning would be necessary for both benchmark problems. In terms of the TSPTW our algorithms struggle in particular with some of the harder Ascheuer instances. A promising solution appears to be the inclusion of information related to node precedences as done, e.g., in [9, 48]. Concerning the rooted DCMST our results are already quite close to the state-of-the-art column generation approach in [107] with only four instances that could not be solved to optimality.

In preliminary experiments we tested a cleanup algorithm that removes nodes from the LG that were not used for a specified number of iterations. This approach showed potential to decrease the final graph sizes further. Unfortunately, we ran into problems with cycling that increased the number of iterations, negating the provided benefits. Future research could address these issues by more complex cleanup or cycle-prevention strategies. Having shown that even smaller final graph sizes can be achieved, we think that a more theoretical investigation could prove useful. Computing minimal or even minimum LGs that lead to tight dual bounds or even optimal solutions could serve as starting point to design more elaborate refinement strategies.

CHAPTER 8

Conclusions

In this thesis we considered several variants of decomposition approaches for mixed integer linear programming (MILP). Our aim was twofold. First, we wanted to show how decomposition approaches can be employed to tackle challenging optimization problems that could not be handled effectively with standard MILP approaches. Second, we used the insights gained from our experiments to improve the decomposition methods. To this end we focused on individual subcomponents and thoroughly investigated different alternatives, including novel strategies developed in this work. Some of the novel extensions and methods are problem-specific, but several concepts are more generic and therefore appear to be promising also for solving other problems.

In Chapter 3 we started by considering column generation for solving the network design problem with relays (NDPR). We improved upon a previous application of column generation by applying a graph transformation. Thereby we managed to reduce the complexity of the pricing subproblem leading to a more balanced decomposition. Through the incorporation of additional strengthening inequalities we obtained an effective approach. While this strategy is rather problem-dependent, we think that at least the idea can be generalized: Although some method might not be successful when applied in a straightforward way, it can still succeed after exploiting insights into structural aspects of the problem that can be used to develop a suitable problem transformation.

The next decomposition method we investigated was logic-based Benders decomposition (LBBD), which is a recent extension of the well-established classical Benders decomposition (BD). As benchmark problem we selected a request maximizing variant of the dial-a-ride problem (DARP). By means of the decomposition approach we separated the problem into an optimization and a feasibility component to facilitate the application of specialized algorithms. In particular, we used a MILP model to deal with the optimization aspect. To enhance performance we included additional valid inequalities derived from subproblem relaxations. Further speedups could be achieved through heuristic techniques applied in terms of the Benders algorithm. The feasibility component was tackled with a constraint programming (CP) approach. To compensate for significant outliers in terms of computation time we resorted to a hybrid approach. This was done by incorporating a MILP algorithm that was slower on average but much more consistent in terms of computation time. Finally, we investigated different techniques for generating Benders cuts. To this end we considered straightforward Benders cuts as reference point and compared them to heuristically strengthened ones as well as two variants of theoretically strongest Benders cuts. Our study showed that Benders cuts based on infeasible substructures of minimum cardinality are particularly successful. This result is quite interesting as our method incurred substantial overhead for deriving these substructures and yet performed similar or even better than the much faster heuristic approach. Developing algorithms to compute such substructures efficiently appears to be an interesting starting point for future work. We believe that similar Benders cuts can also be considered for other problems. Most contributions from the literature employ straightforward Benders cuts—sometimes strengthened by simple heuristics—but do not consider more elaborate techniques. Our result provides a lead on future research directions to develop improvements in this respect.

The next method we considered was based on relaxations. Our goal was to tackle a scheduling problem subject to a fine-grained time discretization. To reduce the impact of the large resulting time horizon we aggregated consecutive time instants into so-called time buckets. The resulting MILP formulation constitutes a relaxation with respect to the original problem, potentially leading to solutions with remaining infeasibilities. Accordingly, we developed an algorithm based on iteratively deriving relaxed solutions and according dual bounds, originating from successively refined aggregations. Once the solution to the relaxation becomes feasible for the original problem, we know that it must be optimal. To improve convergence speed we also developed heuristics to derive primal solutions guided by the relaxations. In our study we focused strongly one the refinement process, i.e., the strategy according to which the buckets are subdivided in each iteration to refine the relaxation. Our results indicate that it is most important to restrict the growth of the model as well as the number of iterations. A reasonable balance among this conflicting goals can only be achieved by applying just few of the most relevant refinements. The most successful strategy was based on exploiting additional information gained by observing the solution components that are responsible for the remaining infeasibilities. Compared to classical MILP approaches from the literature our algorithm performed much better. In this respect our technique constitutes a valuable alternative that might also be applied to other scheduling problems with similar characteristics. During our investigation we strongly focused on the refinement step, which is according to our experience one of the most important parts of the algorithm. However, to improve the performance of the approach further, we believe that additional efforts should be devoted also to other aspects of the algorithm. Along these lines it might be promising to explore more sophisticated algorithms for computing heuristic solutions. Suitable techniques might be able to provide additional information that can be exploited in terms of the refinement process. Another important observation is that although our relaxation is considerably smaller than the original problem, solving its associated MILP model is still the main bottleneck of the approach. On the one hand, this seems to be justified as tight dual bounds are necessary to prove optimality. On the other hand, we think that a slightly different relaxation or another method of solving it could lead to improvements. Considering CP techniques appears to be particularly promising.

For solving the directed network design problem with relays (DNDPR)—the directed counterpart of the NDPR—we used layered graphs (LGs). We did so in order to deal with a specific side constraint that could not be handled effectively in terms of the communication graphs used for the NDPR. The idea of LGs is to extend a base graph along one or multiple dimensions to facilitate MILP modeling options. In particular, associated formulations are known to provide strong linear programming (LP) bounds. The drawback, however, is that LGs are typically substantially larger than the underlying base graph. In terms of the considered problem the graph size strongly depends on an imposed distance restriction. This entails additional difficulties when dealing with fractional input values. To restrict the size of the resulting LG we rounded down fractional inputs to obtain a relaxation. Potential infeasibilities were then addressed by cutting planes. We improved the considered model further by developing additional strengthening inequalities and symmetry breaking constraints. With our approach we achieved new state-of-the-art results for the DNDPR. The main downside of our algorithm is that it struggles with larger instances that lead to prohibitive graph sizes despite our rounding strategy. A potential remedy might be the iterative approach discussed in Chapter 7 that we are going to summarize in the following.

As mentioned above, LGs are in general an appealing modeling approach due to leading to excellent LP bounds. However, often they are not applicable as a result of their excessive size. The aforementioned combination of rounding and cutting planes was beneficial to avoid issues with fractional input values, in particular, because only few violations occurred. Unfortunately, often already integral inputs lead to prohibitive graph sizes. A coarser aggregation could be an option but often leads to many infeasibilities, making separation ineffective in practice. Instead, we considered an iterative approach that avoids the full size of the LG based on the observation that typically only a (small) subgraph is required to obtain a provably optimal solution. The resulting algorithm is similar to the one we considered for high-resolution scheduling. Instead of aggregating time instants we aggregate node copies and redirect arcs accordingly. Part of the node copies that were omitted in this way is then reintroduced until optimality can be proven. Different from the scheduling domain, comparable approaches have been considered in the previous network design literature. A major shortcoming of these works is that they focus rather strongly on specific applications. This leads to an overemphasis of the (re-)development of the iterative algorithm while refraining to question the way in which certain components are implemented. Taking into account the lessons learned from our experiments on the scheduling problem we focused on developing alternatives for the refinement step. We took advantage of the structural information provided by the underlying graph problem to design path-based approaches. The conducted experiments indicate that our novel strategies can prove optimality with considerably smaller LGs than the approaches from the literature. Consequently, further instances could be solved to optimality within shorter computation times. Preliminary experiments indicated that it might be possible to prove optimality with even smaller LGs. We therefore think that a more theoretical investigation could be a starting point for future work. To this end it appears to be promising to compute minimal LGs that provide tight dual bounds or encode optimal solutions. The structure of these graphs might provide important insights from which even more successful refinement strategies can be derived.

The variety of decomposition algorithms investigated in this thesis underlines the importance of investing in the development of different strategies. While the considered principles are very general, their success strongly depends on the problem at hand. Consequently, one should always aim to explore a problems peculiarities and special structural features to decide which approach fits best. Finally, the selected algorithm should be adjusted according to the gained insights, possibly hybridizing several techniques to draw upon their individual strengths.

APPENDIX A

Additional Result Tables for the Selective Dial-a-Ride Problem

							Computation time [s]									LB								
	Instance properties							LBBD				BaC					LBBD				BaC			
Instance	$\|K\|$	n	T	Q	L	LB*	CM	simple	aIIS	mIIS	gIIS	simple	aIIS	mIIS	gIIS	CM	simple	aIIS	mIIS	gIIS	simple	aIIS	mIIS	gIIS
30N_4K_A	4	30	240	3	30	**30**	83	3	53	38	6	11	24	23	**2**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_4K_B	4	30	240	3	30	**29**	ML	595	15	12	4	11	12	3	**1**	ML	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**
30N_4K_C	4	30	240	3	30	**30**	ML	TL	355	202	47	1293	195	192	**23**	ML	29	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_A	5	30	240	3	30	**30**	79	< **1**	112	4	1	21	30	34	2	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_B	5	30	240	3	30	**30**	23	3	13	4	1	< **1**	13	11	1	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_C	5	30	240	3	30	**30**	6655	406	137	75	19	19	52	25	**3**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
40N_4K_A	4	40	240	3	30	**38**	TL	TL	TL	343	**71**	TL	6053	496	101	36	36	**38**	**38**	**38**	36	**38**	**38**	**38**
40N_4K_B	4	40	240	3	30	**38**	ML	TL	6798	375	282	TL	3132	540	**144**	ML	36	**38**	**38**	**38**	29	**38**	**38**	**38**
40N_4K_C	4	40	232	3	30	**37**	TL	TL	TL	798	802	TL	TL	454	**142**	29	34	36	**37**	**37**	28	**37**	**37**	**37**
40N_5K_A	5	40	240	3	30	**40**	TL	35	2669	68	5	568	77	33	**3**	31	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
40N_5K_B	5	40	240	3	30	**40**	TL	TL	1499	327	97	TL	430	172	**27**	39	38	**40**	**40**	**40**	39	**40**	**40**	**40**
40N_5K_C	5	40	240	3	30	**40**	1662	10	29	53	14	308	108	5	**2**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
44N_4K_A	4	44	240	3	30	40	TL	TL	TL	TL	TL	TL	TL	TL	TL	38	36	36	39	38	24	38	**40**	**40**
44N_4K_B	4	44	240	3	30	**42**	TL	TL	TL	1959	**1088**	TL	TL	4279	3191	30	38	40	**42**	**42**	34	41	**42**	**42**
44N_4K_C	4	44	240	3	30	**41**	TL	TL	TL	TL	TL	TL	TL	**4740**	5553	32	37	35	40	39	33	40	**41**	**41**
44N_5K_A	5	44	239	3	30	**44**	ML	TL	TL	323	131	TL	429	339	**61**	ML	42	43	**44**	**44**	30	**44**	**44**	**44**
44N_5K_B	5	44	240	3	30	**44**	TL	TL	4173	1821	99	TL	872	155	**27**	42	43	**44**	**44**	**44**	38	**44**	**44**	**44**
44N_5K_C	5	44	240	3	30	**44**	ML	TL	5491	938	106	TL	652	514	**50**	ML	42	**44**	**44**	**44**	40	**44**	**44**	**44**
50N_4K_A	4	50	240	3	30	41	TL	TL	TL	TL	TL	TL	TL	TL	TL	26	37	35	38	39	26	39	**40**	**40**
50N_4K_B	4	50	240	3	30	43	TL	TL	ML	TL	TL	TL	TL	TL	TL	35	38	ML	39	39	34	35	**42**	41
50N_4K_C	4	50	240	3	30	44	TL	TL	TL	TL	TL	TL	TL	TL	TL	30	39	35	41	40	31	41	**43**	**43**
50N_5K_A	5	50	240	3	30	48	TL	TL	TL	TL	TL	TL	TL	TL	TL	19	44	39	45	46	40	**47**	**47**	**47**
50N_5K_B	5	50	240	3	30	**49**	TL	TL	TL	1667	1008	TL	TL	1856	**621**	36	44	42	**49**	**49**	30	48	**49**	**49**
50N_5K_C	5	50	240	3	30	**50**	TL	TL	TL	5480	1426	TL	6481	892	**80**	36	47	48	**50**	**50**	43	**50**	**50**	**50**
60N_4K_A	4	60	240	3	30	44	TL	TL	TL	TL	TL	TL	TL	TL	TL	29	40	34	39	39	26	39	42	**43**
60N_4K_B	4	60	240	3	30	45	TL	TL	TL	TL	TL	TL	TL	TL	TL	-	42	0	44	41	25	-	**45**	44
60N_4K_C	4	60	240	3	30	44	TL	TL	TL	TL	TL	TL	TL	TL	TL	27	41	38	41	41	34	37	**43**	42
60N_5K_A	5	60	240	3	30	56	TL	TL	TL	TL	TL	TL	TL	TL	TL	37	52	0	54	54	35	-	52	**56**
60N_5K_B	5	60	240	3	30	50	TL	TL	TL	TL	TL	TL	TL	TL	TL	25	45	44	46	45	30	45	**50**	49
60N_5K_C	5	60	240	3	30	53	TL	TL	TL	TL	TL	TL	TL	TL	TL	36	47	43	49	47	39	48	51	**52**

Table A.1: Overview of the instance properties and the computation times of the unboosted algorithm variants. Column LB* shows the best known lower bounds. Bounds corresponding to provably optimal solution values are marked bold. Columns CM, LBBD, and BaC show the computation times and lower bounds for the compact model, the LBBD decomposition algorithm and the BaC decomposition algorithm, respectively. For the decomposition approaches four kinds of sets have been used to obtain Benders cuts: simple uses unrefined cuts, aIIS uses all IISs, mIIS uses all IISs of minimum cardinality, and gIIS uses two heuristically computed IISs. Instances that could not be solved within the time limit of 2 hours are marked with “TL” and test runs that terminated due to the memory limit are marked with “ML”. For each instance the computation times of the fastest algorithm(s) and best bounds obtained are marked bold.

		Computation time [s]								LB							
		LBBD				BaC				LBBD				BaC			
Instance	LB*	simple	aIIS	mIIS	gIIS	simple	aIIS	mIIS	gIIS	simple	aIIS	mIIS	gIIS	simple	aIIS	mIIS	gIIS
30N_4K_A	**30**	2	10	4	1	5	4	3	< **1**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_4K_B	**29**	467	2	2	1	7	2	< **1**	< **1**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**
30N_4K_C	**30**	TL	95	25	10	754	31	15	**3**	29	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_A	**30**	< **1**	20	< **1**	< **1**	8	4	4	< **1**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_B	**30**	1	2	1	< **1**	< **1**	2	1	< **1**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_C	**30**	349	24	11	3	6	6	2	< **1**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
40N_4K_A	**38**	TL	5391	111	**63**	TL	5670	99	68	36	**38**	**38**	**38**	36	**38**	**38**	**38**
40N_4K_B	**38**	TL	6181	90	209	TL	2908	97	**69**	36	**38**	**38**	**38**	29	**38**	**38**	**38**
40N_4K_C	**37**	TL	TL	589	760	TL	TL	**125**	135	34	36	**37**	**37**	28	**37**	**37**	**37**
40N_5K_A	**40**	23	2747	16	1	747	14	4	< **1**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
40N_5K_B	**40**	TL	1015	84	57	TL	129	23	**9**	38	**40**	**40**	**40**	39	**40**	**40**	**40**
40N_5K_C	**40**	5	5	9	7	259	19	1	< **1**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
44N_4K_A	40	TL	TL	TL	TL	TL	TL	TL	TL	36	36	39	38	24	39	**40**	**40**
44N_4K_B	**42**	TL	TL	**750**	1025	TL	TL	1550	2768	38	40	**42**	**42**	34	41	**42**	**42**
44N_4K_C	**41**	TL	TL	TL	TL	TL	TL	**3134**	5552	37	35	40	39	35	40	**41**	**41**
44N_5K_A	**44**	TL	6252	97	104	TL	126	65	**21**	42	**44**	**44**	**44**	27	**44**	**44**	**44**
44N_5K_B	**44**	TL	3141	520	47	TL	263	18	**12**	43	**44**	**44**	**44**	38	**44**	**44**	**44**
44N_5K_C	**44**	TL	4576	199	57	TL	252	64	**16**	42	**44**	**44**	**44**	40	**44**	**44**	**44**
50N_4K_A	41	TL	TL	TL	TL	TL	TL	TL	TL	37	35	38	39	22	39	**41**	**41**
50N_4K_B	43	TL	ML	TL	TL	TL	TL	TL	TL	38	ML	39	39	34	34	**42**	41
50N_4K_C	44	TL	TL	TL	TL	TL	TL	TL	TL	39	35	41	40	31	41	**43**	**43**
50N_5K_A	48	TL	TL	TL	TL	TL	TL	TL	TL	45	39	45	46	40	47	**48**	**48**
50N_5K_B	**49**	TL	ML	778	791	TL	TL	484	**447**	44	ML	**49**	**49**	30	48	**49**	**49**
50N_5K_C	**50**	TL	TL	1458	933	TL	4343	115	**19**	47	48	**50**	**50**	43	**50**	**50**	**50**
60N_4K_A	44	TL	TL	TL	TL	TL	TL	TL	TL	40	34	39	39	26	40	42	**43**
60N_4K_B	45	TL	TL	TL	TL	TL	TL	TL	TL	42	0	44	41	25	-	**45**	44
60N_4K_C	44	TL	TL	TL	TL	TL	TL	TL	TL	41	38	41	41	34	39	**43**	42
60N_5K_A	56	TL	TL	TL	TL	TL	ML	TL	TL	52	46	55	54	35	ML	54	**56**
60N_5K_B	50	TL	ML	TL	TL	TL	TL	TL	TL	45	ML	46	45	30	46	**50**	48
60N_5K_C	53	TL	TL	TL	TL	TL	TL	TL	TL	47	43	49	47	39	48	51	**52**

Table A.2: Overview of the computation times and lower bounds of the un-boosted LBBD and BaC algorithm variants using a CP-MIP combination for the subproblems. Column LB* shows the best known lower bounds (provably optimal solution values are marked bold). For the decomposition approaches four kinds of sets have been used to obtain Benders cuts: simple uses unrefined cuts, aIIS uses all IISs, mIIS uses all IISs of minimum cardinality, and gIIS uses two heuristically computed IISs. Instances that could not be solved within the time limit of 2 hours are marked with “TL” and test runs that terminated due to the memory limit are marked with “ML”. For each instance the computation times of the fastest algorithm(s) and best bounds obtained are marked bold.

		Computation time [s]										LB									
		standard		gap (rel) - it		gap (rel) - ud		time - it		time - ud		standard		gap (rel) - it		gap (rel) - ud		time - it		time - ud	
Instance	LB*	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS
30N_4K_A	**30**	4	**1**	**1**	**1**	**1**	**1**	4	**1**	4	**1**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_4K_B	**29**	2	**1**	2	**1**	2	**1**	2	**1**	2	**1**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**
30N_4K_C	**30**	25	**10**	36	19	42	22	26	**10**	25	**10**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_A	**30**	**< 1**	**< 1**	**< 1**	**< 1**	**< 1**	**< 1**	**< 1**	**< 1**	**< 1**	**< 1**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_B	**30**	1	**< 1**	1	**< 1**	1	**< 1**	1	**< 1**	1	**< 1**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
30N_5K_C	**30**	11	**3**	8	7	14	4	11	**3**	11	**3**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
40N_4K_A	**38**	111	**63**	145	81	233	107	113	69	113	65	**38**	**38**	**38**	**38**	**38**	**38**	**38**	**38**	**38**	**38**
40N_4K_B	**38**	90	209	108	**71**	108	79	91	146	92	134	**38**	**38**	**38**	**38**	**38**	**38**	**38**	**38**	**38**	**38**
40N_4K_C	**37**	589	760	210	**173**	248	265	352	671	353	460	**37**	**37**	**37**	**37**	**37**	**37**	**37**	**37**	**37**	**37**
40N_5K_A	**40**	16	**1**	33	**1**	33	**1**	16	**1**	16	**1**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
40N_5K_B	**40**	84	**57**	112	83	312	298	88	58	87	59	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
40N_5K_C	**40**	9	7	7	**6**	7	**6**	8	7	9	7	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
44N_4K_A	40	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	39	38	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
44N_4K_B	**42**	**750**	1025	1674	930	2850	2100	1254	1224	1203	1664	**42**	**42**	**42**	**42**	**42**	**42**	**42**	**42**	**42**	**42**
44N_4K_C	**41**	TL	TL	TL	TL	4428	2939	4439	**2721**	6534	3560	40	39	40	40	**41**	**41**	**41**	**41**	**41**	**41**
44N_5K_A	**44**	97	104	266	227	994	534	**95**	105	98	106	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**
44N_5K_B	**44**	520	47	368	**45**	461	255	517	48	522	49	**44**	**44**	**44**	**44**	**44**	43	**44**	**44**	**44**	**44**
44N_5K_C	**44**	199	**57**	388	178	1335	1354	195	58	205	59	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**
50N_4K_A	41	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	38	39	39	39	39	39	**41**	40	**41**	**41**
50N_4K_B	43	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	39	39	40	40	40	40	**43**	42	**43**	42
50N_4K_C	44	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	41	40	42	43	42	43	**44**	**44**	**44**	**44**
50N_5K_A	48	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	45	46	**48**	47	47	**48**	47	47	**48**	47
50N_5K_B	**49**	**778**	791	873	878	2785	2234	1079	918	882	1137	**49**	**49**	**49**	**49**	**49**	**49**	**49**	**49**	**49**	**49**
50N_5K_C	**50**	1458	933	1670	693	1593	TL	965	1449	987	**644**	**50**	**50**	**50**	**50**	**50**	49	**50**	**50**	**50**	**50**
60N_4K_A	44	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	39	39	40	42	40	42	**44**	43	**44**	43
60N_4K_B	45	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	44	41	42	44	42	44	**45**	44	**45**	44
60N_4K_C	44	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	41	41	41	40	41	40	**44**	43	**44**	43
60N_5K_A	56	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	**55**	54	54	54	54	54	**55**	54	**55**	**55**
60N_5K_B	50	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	46	45	48	48	48	48	**50**	49	**50**	49
60N_5K_C	53	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	49	47	50	51	50	51	**53**	52	**53**	52

Table A.3: Results of the heuristic boosting techniques. Column computation time reports the time consumed and column LB provides the value of the lower bound. Column standard shows results of the algorithms without boosting, “gap (rel) - it” shows results for boosting with purely iterative adjustments whereas “gap (rel) - ud” adapts the threshold in both directions. Similarly, “time - it” and “time - ud” report results for boosting with reduced time limit. Smallest computation times and best bounds per instance are marked bold.

	Iterations								Master-Sub ratio								Cuts							
	LBBD								LBBD								LBBD							
	un-boosted		time - it		time - ud		BaC		un-boosted		time - it		time - ud		BaC		un-boosted		time - it		time - ud		BaC	
Instance	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS
30N_4K_A	24	19	24	19	24	19	23	14	0.07	0.17	0.06	0.19	0.06	0.13	0.02	0.07	400	280	400	280	400	280	232	**184**
30N_4K_B	14	16	14	16	14	16	5	19	0.36	1.63	0.39	1.79	0.40	1.66	0.28	0.57	240	208	240	208	240	208	**148**	152
30N_4K_C	86	92	86	92	86	92	69	80	0.28	1.28	0.28	1.29	0.29	1.38	0.01	0.08	1624	1668	1624	1668	1624	1668	**1036**	1060
30N_5K_A	3	3	3	3	3	3	33	17	0.07	0.43	0.02	0.33	0.02	0.43	0.01	0.15	165	**45**	165	**45**	165	**45**	485	220
30N_5K_B	4	7	4	7	4	7	10	17	0.09	0.45	0.09	0.43	0.07	0.36	0.02	0.12	230	**110**	230	**110**	230	**110**	260	265
30N_5K_C	46	48	46	48	46	48	31	25	0.15	0.97	0.17	1.01	0.16	1.06	0.01	0.34	975	960	975	960	975	960	520	**425**
40N_4K_A	78	97	78	97	78	97	97	142	0.61	1.21	0.64	1.37	0.69	1.21	0.18	2.94	**1832**	2032	**1832**	2032	**1832**	2032	1888	2160
40N_4K_B	76	127	76	127	76	127	128	209	0.92	15.26	0.94	9.73	1.02	8.75	0.50	3.80	**2096**	2604	**2096**	2604	**2096**	2604	2408	3060
40N_4K_C	86	145	131	195	131	169	155	227	12.60	17.72	3.73	9.37	3.74	8.57	1.03	1.95	**2260**	3128	2860	3556	2860	3288	3012	3688
40N_5K_A	34	15	34	15	34	15	19	18	0.06	0.22	0.07	0.30	0.07	0.32	0.02	0.07	1075	385	1075	385	1075	385	260	**240**
40N_5K_B	124	135	124	135	124	135	100	95	0.54	0.78	0.59	0.85	0.58	0.83	0.03	0.06	3510	3710	3510	3710	3510	3710	2145	**2045**
40N_5K_C	27	37	27	37	27	37	8	11	0.07	0.17	0.08	0.20	0.07	0.20	0.08	0.07	985	1045	985	1045	985	1045	**150**	170
44N_4K_A	83	151	187	299	317	431	297	393	116.49	215.85	42.49	124.35	28.81	100.03	32.64	153.95	**3616**	4084	5440	6592	6716	8056	6544	7292
44N_4K_B	153	225	251	304	244	374	429	391	1.77	14.27	1.84	13.65	1.94	16.67	1.91	29.12	**4016**	5324	5280	6468	5248	7244	7184	7268
44N_4K_C	168	212	239	274	430	383	310	343	28.33	108.24	14.04	30.19	11.92	28.59	10.66	76.03	**4988**	5340	5632	6020	7216	7084	5716	6200
44N_5K_A	94	136	94	136	94	136	112	162	0.22	0.30	0.23	0.29	0.23	0.30	0.02	0.07	3515	4005	3515	4005	3515	4005	**2655**	3125
44N_5K_B	389	144	389	144	389	144	51	103	0.39	0.50	0.38	0.49	0.39	0.51	0.01	0.03	8645	3890	8645	3890	8645	3890	**1175**	1885
44N_5K_C	214	151	214	151	214	151	136	207	0.58	0.69	0.61	0.67	0.62	0.71	0.02	0.22	6905	4870	6905	4870	6905	4870	**3045**	4265
50N_4K_A	67	192	413	632	604	636	738	1183	64.76	38.11	6.45	13.69	4.29	13.87	4.48	13.09	**4812**	5992	12316	16392	14672	15752	16104	23948
50N_4K_B	52	144	407	483	598	433	570	812	87.73	155.20	4.36	32.39	2.74	36.75	4.28	25.16	**3292**	4416	10796	12548	13576	11364	12188	16860
50N_4K_C	46	96	196	288	309	383	499	609	112.90	130.09	16.46	53.28	10.35	43.14	6.65	36.94	**2472**	2920	5808	7332	7440	8764	9164	11632
50N_5K_A	192	298	683	774	959	1391	1221	1563	27.13	112.20	4.30	35.56	3.09	17.11	2.91	18.35	**9865**	11540	23755	25050	28520	37945	34120	42425
50N_5K_B	154	217	231	273	206	310	296	353	3.69	20.97	2.84	20.25	2.97	23.24	1.08	11.24	**6085**	7425	7895	8745	7335	9355	7820	9245
50N_5K_C	513	568	390	687	390	481	173	167	1.00	10.23	0.79	13.36	0.82	8.63	0.02	0.11	14380	16005	12040	18335	12040	14260	**4260**	4490
60N_4K_A	17	93	609	864	586	892	818	915	385.39	67.87	1.25	5.09	1.12	4.41	1.77	12.19	**2504**	2916	17976	23140	17336	23424	19428	20368
60N_4K_B	50	243	528	887	527	921	777	1650	45.90	16.46	0.83	2.73	0.84	3.00	1.14	3.18	**5396**	7708	21196	26776	21424	27680	24700	39880
60N_4K_C	36	123	790	1120	801	1159	908	1378	117.79	89.90	1.65	11.66	1.74	10.45	3.05	15.33	**3644**	3896	22648	28900	22572	29036	22684	31824
60N_5K_A	289	541	619	1183	778	1240	1148	1571	4.49	11.38	1.41	5.59	1.05	5.14	0.83	5.34	**17390**	24160	28320	45580	33010	46330	33125	50675
60N_5K_B	46	150	760	1196	1018	1346	813	1404	181.78	269.00	4.93	24.13	3.10	23.99	6.57	35.34	**5820**	7170	32975	43680	39225	45890	28525	43365
60N_5K_C	50	126	226	791	495	1260	776	742	120.93	320.61	17.22	43.90	6.58	30.95	4.94	59.57	**5075**	5870	12860	27230	19935	35905	25720	24320

Table A.4: Characteristics of the decomposition approaches. Column iterations states the number of iterations the algorithm completed. For the LBBD approaches this corresponds to the number of times the master problem has been solved. For the BaC approaches it is equal to the number of times the separation routine has been called. Column master-sub ratio provides the relative ratio of time spent in the master problem compared to those spent in the subproblems ($t_{\text{master}}/t_{\text{sub}}$). The last column (cuts) shows the total number of Benders cuts that have been added.

			LB gap [%]									UB gap [%]								
				LBBD									LBBD							
				un-boosted		time - it		time - ud		BaC			un-boosted		time - it		time - ud		BaC	
Instance	LB*	UB*	CM	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	CM	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS	mIIS	gIIS
30N_4K_A	**30**	**30**	**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**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
30N_4K_B	**29**	**29**	-	**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**	**0.0**	**0.0**	**0.0**
30N_4K_C	**30**	**30**	-	**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**	**0.0**	**0.0**	**0.0**
30N_5K_A	**30**	**30**	**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**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
30N_5K_B	**30**	**30**	**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**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
30N_5K_C	**30**	**30**	**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**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
40N_4K_A	**38**	**38**	5.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	5.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
40N_4K_B	**38**	**38**	-	**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**	**0.0**	**0.0**	**0.0**
40N_4K_C	**37**	**37**	21.6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	8.1	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
40N_5K_A	**40**	**40**	22.5	**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**	**0.0**	**0.0**	**0.0**	**0.0**
40N_5K_B	**40**	**40**	2.5	**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**	**0.0**	**0.0**	**0.0**	**0.0**
40N_5K_C	**40**	**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**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
44N_4K_A	40	41	5.0	2.5	5.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	7.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	3.3	3.9
44N_4K_B	**42**	**42**	28.6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	4.8	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
44N_4K_C	**41**	**41**	22.0	2.4	4.9	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	7.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
44N_5K_A	**44**	**44**	-	**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**	**0.0**	**0.0**	**0.0**
44N_5K_B	**44**	**44**	4.5	**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**	**0.0**	**0.0**	**0.0**	**0.0**
44N_5K_C	**44**	**44**	-	**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**	**0.0**	**0.0**	**0.0**
50N_4K_A	41	44	36.6	7.3	4.9	**0.0**	2.4	**0.0**	**0.0**	**0.0**	**0.0**	13.6	**0.0**	4.5	**0.0**	6.8	**0.0**	**0.0**	9.1	9.1
50N_4K_B	43	47	18.6	9.3	9.3	**0.0**	2.3	**0.0**	2.3	2.3	4.7	6.4	**0.0**	**0.0**	**0.0**	2.1	2.1	**0.0**	4.3	4.3
50N_4K_C	44	46	31.8	6.8	9.1	**0.0**	**0.0**	**0.0**	**0.0**	2.3	2.3	8.7	2.2	2.2	2.2	**0.0**	**0.0**	2.2	6.2	6.5
50N_5K_A	48	49	60.4	6.2	4.2	2.1	2.1	**0.0**	2.1	**0.0**	**0.0**	2.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	2.0	2.0	2.0
50N_5K_B	**49**	**49**	26.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	2.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
50N_5K_C	**50**	**50**	28.0	**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**	**0.0**	**0.0**	**0.0**	**0.0**
60N_4K_A	44	56	34.1	11.4	11.4	**0.0**	2.3	**0.0**	2.3	4.5	2.3	7.1	**0.0**	3.6	3.6	5.4	3.6	5.4	5.7	5.4
60N_4K_B	45	56	-	2.2	8.9	**0.0**	2.2	**0.0**	2.2	**0.0**	2.2	7.1	**0.0**	1.8	5.4	5.4	5.4	5.4	6.9	6.5
60N_4K_C	44	53	38.6	6.8	6.8	**0.0**	2.3	**0.0**	2.3	2.3	4.5	13.2	**0.0**	1.9	7.5	7.5	7.5	7.5	8.5	9.4
60N_5K_A	56	60	33.9	1.8	3.6	1.8	3.6	1.8	1.8	3.6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
60N_5K_B	50	59	50.0	8.0	10.0	**0.0**	2.0	**0.0**	2.0	**0.0**	4.0	1.7	**0.0**	**0.0**	**0.0**	1.7	**0.0**	1.7	1.7	1.7
60N_5K_C	53	58	32.1	7.5	11.3	**0.0**	1.9	**0.0**	1.9	3.8	1.9	3.4	**0.0**	**0.0**	**0.0**	1.7	**0.0**	1.7	3.4	3.4

Table A.5: Overview of relative gaps to the best known lower (column LB gap) and upper (column UB gap) bounds, respectively. Columns LB* and UB* report the best known lower and upper bounds obtained across all algorithms. Entries in column LB* are marked bold if the corresponding solution is provably optimal. Per instance smallest gaps are marked bold.

Acronyms

ABCH activity block construction heuristic.

B&B branch-and-bound.

B&C branch-and-cut.

B&P branch-and-price.

BaC Branch-and-Check.

BD Benders decomposition.

BP&C branch-price-and-cut.

CapEx capital expenditure.

CP constraint programming.

CSP constraint satisfaction problem.

CTSNDP countinuous time service network design problem.

DARP dial-a-ride problem.

DCMST distance-constrained minimum spanning tree problem.

DEF discrete-event formulation.

DNDPR directed network design problem with relays.

DP dynamic programming.

EV electric vehicle.

GCH gap closing heuristic.

GRASP greedy randomized adaptive search procedure.

GRLP generalized regenerator location problem.

IIS irreducible infeasible set.

ILP integer linear programming.

IRA iterative refinement algorithm.

ITBRA iterative time-bucket refinement algorithm.

LBBD logic-based Benders decomposition.

LG layered graph.

LP linear programming.

MCDSP minimum connected dominating set problem.

MCPPR minimum cost path problem with relays.

MILP mixed integer linear programming.

MLSTP maximum leaf spanning tree problem.

MRCMPSP multi-mode resource-constrained multi-project scheduling problem.

MVRPTW vehicle routing problem with time windows and multiple routes.

NDPR network design problem with relays.

OpEx operational expenditure.

PDPTW pickup and delivery problem with time windows.

PHEV plug-in hybrid electric vehicle.

RCPSP resource-constrained project scheduling problem.

RLP regenerator location problem.

SI-PTPSP simplified intraday particle therapy patient scheduling problem.

TBR time-bucket relaxation.

TIF time-indexed formulation.

TSP traveling salesman problem.

TSPTW traveling salesman problem with time windows.

VRP vehicle routing problem.

WCSPP weight constrained shortest path problem.

WDM wavelength division multiplexing.

Bibliography

[1] O. Arslan, B. Yıldız, and O. E. Karaşan. Minimum cost path problem for plug-in hybrid electric vehicles. *Transportation Research Part E: Logistics and Transportation Review*, 80:123–141, 2015.

[2] C. Artigues. On the strength of time-indexed formulations for the resource-constrained project scheduling problem. *Operations Research Letters*, 45(2):154 – 159, 2017.

[3] C. Artigues and E. Hebrard. MIP relaxation and large neighborhood search for a multi-mode resource-constrained multi-project scheduling problem. In G. Kendall, B. McCollum, and G. Venden Berghe, editors, *In proceedings of the 6th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2013), 27–30 Aug 2013, Ghent, Belgium*, pages 815–819, 2013.

[4] C. Artigues, S. Demassey, and E. Neron. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications.* Wiley-ISTE, 2008.

[5] C. Artigues, P. Brucker, S. Knust, O. Koné, and P. Lopez. A note on "event-based MILP models for resource-constrained project scheduling problems". *Computers & Operations Research*, 40(4):1060–1063, 2013.

[6] 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.

[7] D. G. Baklagis, G. Dikas, and I. Minis. The team orienteering pick-up and delivery problem with time windows and its applications in fleet sizing. *RAIRO-Operations Research*, 50(3):503–517, 2016.

[8] R. Baldacci, E. Bartolini, and A. Mingozzi. An exact algorithm for the pickup and delivery problem with time windows. *Operations Research*, 59(2):414–426, 2011.

[9] R. Baldacci, A. Mingozzi, and R. Roberti. New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS J. Comput.*, 24(3): 356–371, 2012.

[10] P. Baptiste and R. Sadykov. On scheduling a single machine to minimize a piecewise linear objective function: A compact mip formulation. *Naval Research Logistics*, 56 (6):487–502, 2009.

[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*, 46:316–329, 1998.

[12] J. W. Baugh, jr., G. K. R. Kakivaya, and J. R. Stone. Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. *Engineering Optimization*, 30(2):91–123, 1998.

[13] G. Baydoun, A. Haït, R. Pellerin, B. Clément, and G. Bouvignies. A rough-cut capacity planning model with overlapping. *OR Spectrum*, 38(2):335–364, 2016.

[14] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[15] G. Berbeglia, G. Pesant, and L.-M. Rousseau. Checking the feasibility of dial-a-ride instances using constraint programming. *Transportation Science*, 45(3):399–412, 2011.

[16] T. Berthold, S. Heinz, M. E. Lübbecke, R. H. Möhring, and J. Schulz. A constraint integer programming approach for resource-constrained project scheduling. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, volume 6140 of *Lecture Notes in Computer Science*, pages 313–317. Springer, 2010.

[17] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1st edition, 1997.

[18] BGL. The boost graph library (BGL). http://www.boost.org/doc/libs/1_63_0/libs/graph/doc/index.html, 2016. accessed 2017-09-17.

[19] L. Bianco and M. Caramia. A new lower bound for the resource-constrained project scheduling problem with generalized precedence relations. *Computers & Operations Research*, 38(1):14–20, 2011.

[20] L. Bianco and M. Caramia. Minimizing the completion time of a project under resource constraints and feeding precedence relations: a lagrangian relaxation based lower bound. *4OR*, 9(4):371–389, 2011.

[21] L. Bianco and M. Caramia. An exact algorithm to minimize the makespan in project scheduling with scarce resources and generalized precedence relations. *European Journal of Operational Research*, 219(1):73–85, 2012.

[22] L. P. Bigras, M. Gamache, and G. Savard. Time-indexed formulations and the total weighted tardiness problem. *INFORMS Journal on Computing*, 20(1):133–142, 2008.

[23] C. Blum and G. R. Raidl. *Hybrid Metaheuristics: Powerful Tools for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer International Publishing, 2016.

[24] L. D. Bodin and T. Sexton. The multi-vehicle subscriber dial-a-ride problem. *TIMS studies in Management Science*, 2:73–86, 1986.

[25] N. Boland, R. Clement, and H. Waterer. A bucket indexed formulation for nonpreemptive single machine scheduling problems. *INFORMS Journal on Computing*, 28(1):14–30, 2016.

[26] N. Boland, M. Hewitt, D. M. Vu, and M. Savelsbergh. Solving the traveling salesman problem with time windows using time-expanded networks. In *9th Triennial Symposium on Transportation Analysis - TRISTAN 2016*, 2016.

[27] N. Boland, M. Hewitt, L. Marshall, and M. Savelsbergh. The continuous-time service network design problem. *Operations Research*, 65(5):1303–1321, 2017.

[28] C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[29] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European journal of operational research*, 112(1):3–41, 1999.

[30] E. A. Cabral, E. Erkut, G. Laporte, and R. A. Patterson. The network design problem with relays. *European Journal of Operational Research*, 180(2):834–844, 2007.

[31] J. F. Campbell and M. E. O'Kelly. Twenty-five years of hub location research. *Transportation Science*, 46(2):153–169, 2012.

[32] I. Capar, M. Kuby, V. J. Leon, and Y.-J. Tsai. An arc cover–path-cover formulation and strategic analysis of alternative-fuel station locations. *European Journal of Operational Research*, 227(1):142–151, 2013.

[33] J. Carlier, A. Moukrim, and A. Sahli. Lower bounds for the event scheduling problem with consumption and production of resources. *Discrete Applied Mathematics*, to appear. available at https://doi.org/10.1016/j.dam.2016.05.021.

[34] F. Cazals and C. Karande. A note on the problem of reporting maximal cliques. *Theoretical Computer Science*, 407(1–3):564 – 568, 2008.

[35] S. Chen, I. Ljubić, and S. Raghavan. The regenerator location problem. *Networks*, 55(3):205–220, 2010.

[36] S. Chen, I. Ljubić, and S. Raghavan. The generalized regenerator location problem. *INFORMS Journal on Computing*, 27(2):204–220, 2015.

[37] B. V. Cherkassy and A. V. Goldberg. On implementing push-relabel method for the maximum flow problem. In *Proceedings of the 4^{th} International IPCO Conference on Integer Programming and Combinatorial Optimization*, volume 920 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 1995.

[38] A. A. Cire and J. N. Hooker. A heuristic logic-based Benders method for the home health care problem. Technical report, Tepper School of Business, Carnegie Mellon University, 2012.

[39] F. Clautiaux, S. Hanafi, R. Macedo, M.-É. Voge, and C. Alves. Iterative aggregation and disaggregation algorithm for pseudo-polynomial network flow models with side constraints. *European Journal of Operational Research*, 258(2):467–477, 2017.

[40] G. Codato and M. Fischetti. Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006.

[41] J.-F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.

[42] J.-F. Cordeau and G. Laporte. The dial-a-ride problem (DARP): Variants, modeling issues and algorithms. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(2):89–101, 2003.

[43] J.-F. Cordeau and G. Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594, 2003.

[44] J.-F. Cordeau and G. Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.

[45] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4): 393–410, 1954.

[46] G. B. Dantzig. Origins of the simplex method. In S. G. Nash, editor, *A History of Scientific Computing*, pages 141–151. ACM, 1990.

[47] G. B. Dantzig and P. Wolfe. The decomposition algorithm for linear programs. *Econometrica: Journal of the Econometric Society*, pages 767–778, 1961.

[48] 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, 2012.

[49] J. De Boeck and B. Fortz. Extended formulation for hop constrained distribution network configuration problems. *European Journal of Operational Research*, 265 (2):488–502, 2018.

[50] S. Demassey, C. Artigues, and P. Michelon. Constraint-propagation-based cutting planes: An application to the resource-constrained project scheduling problem. *INFORMS Journal on Computing*, 17(1):52–65, 2005.

[51] E. L. Demeulemeester and W. S. Herroelen. A branch-and-bound procedure for the generalized resource-constrained project scheduling problem. *Operations Research*, 45(2):201–212, 1997.

[52] J. Desrosiers and M. E. Lübbecke. Branch-Price-and-Cut algorithms. In *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., 2011.

[53] J. Desrosiers, Y. Dumas, F. Soumis, S. Taillefer, and D. Villeneuve. An algorithm for mini-clustering in handicapped transport. Technical Report G-91-02, GERAD, HEC Montréal, Canada, 1991.

[54] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[55] Y. Dumas, J. Desrosiers, and F. Soumis. Large scale multi-vehicle dial-a-ride problems. Technical Report G-89-30, GERAD, HEC Montréal, Canada, 1989.

[56] Y. Dumas, J. Desrosiers, E. Gelinas, and M. M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, 43(2): 367–371, 1995.

[57] N. Dupin and E. G. Talbi. Dual heuristics and new lower bounds for the challenge EURO/OADEF 2010. In *Matheuristics 2016 - Proceedings of the Sixth International Workshop on Model-based Metaheuristics, 4–7 Sep 2016, Brussels, Belgium*, pages 60–71, 2016.

[58] EPASS24. EPASS24: The swedish road toll system. https://www.epass24.com/, 2018. accessed 2018-09-13.

[59] M. M. Fazel-Zarandi and J. C. Beck. Using logic-based Benders decomposition to solve the capacity- and distance-constrained plant location problem. *INFORMS Journal on Computing*, 24(3):387–398, 2012.

[60] T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.

[61] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[62] M. L. Fisher. Optimal solution of scheduling problems using lagrange multipliers: Part I. *Operations Research*, 21(5):1114–1127, 1973.

[63] G. Gamrath. Generic Branch-Cut-and-Price. Master's thesis, Technical University Berlin, Germany, 2010.

[64] G. Gamrath, T. Fischer, T. Gally, A. M. Gleixner, G. Hendel, T. Koch, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, S. Schenker, R. Schwarz, F. Serrano, Y. Shinano, S. Vigerske, D. Weninger, M. Winkler, J. T. Witt, and J. Witzig. The SCIP optimization suite 3.2. Technical Report 15-60, ZIB, Takustr.7, 14195 Berlin, 2016.

[65] M. Gansterer, M. Küçüktepe, and R. F. Hartl. The multi-vehicle profitable pickup and delivery problem. *OR Spectrum*, 39(1):303–319, 2017.

[66] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[67] M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, STOC '76, pages 10–22, New York, NY, USA, 1976. ACM.

[68] M. Garg and J. C. Smith. Models and algorithms for the design of survivable multicommodity flow networks with general failure scenarios. *Omega*, 36(6):1057–1071, 2008.

[69] Gecode Team. Gecode: Generic constraint development environment, 2017. Available from `https://gecode.github.io`.

[70] M. Gendreau and J.-Y. Potvin, editors. *Handbook of Metaheuristics.* Springer US, 2010.

[71] B. Gendron, A. Lucena, A. S. da Cunha, and L. Simonetti. Benders decomposition, branch-and-cut, and hybrid algorithms for the minimum connected dominating set problem. *INFORMS Journal on Computing*, 26(4):645–657, 2014.

[72] A. M. Geoffrion. Generalized Benders decomposition. *Journal of optimization theory and applications*, 10(4):237–260, 1972.

[73] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9(6):849–859, 1961.

[74] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem: Part II. *Operations Research*, 11(6):863–888, 1963.

[75] M. T. Godinho, L. Gouveia, and P. Pesneau. On a time-dependent formulation and an updated classification of ATSP formulations. In A. R. Mahjoub, editor, *Progress in Combinatorial Optimization*, pages 223–254. ISTE-Wiley, 2011.

[76] M. T. Godinho, L. Gouveia, and P. Pesneau. Natural and extended formulations for the time-dependent traveling salesman problem. *Discrete Applied Mathematics*, 164:138–153, 2014.

[77] L. Gouveia and M. Ruthmair. Load-dependent and precedence-based models for pickup and delivery problems. *Computers & Operations Research*, 63:56–71, 2015.

[78] 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.

[79] L. Gouveia, L. Simonetti, and E. Uchoa. Modeling hop-constrained and diameter-constrained minimum spanning tree problems as steiner tree problems over layered graphs. *Mathematical Programming*, 128(1-2):123–148, 2011.

[80] L. Gouveia, M. Leitner, and I. Ljubić. Hop constrained steiner trees with multiple root nodes. *European Journal of Operational Research*, 236(1):100–112, 2014.

[81] L. Gouveia, M. Leitner, and I. Ljubić. The two-level diameter constrained spanning tree problem. *Mathematical Programming*, 150:49–78, 2015.

[82] L. Gouveia, M. Leitner, and M. Ruthmair. Layered graph approaches for combinatorial optimization problems. *Computers & Operations Research*, 102:22–38, 2019.

[83] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.

[84] F. Guerriero and L. Talarico. A solution approach to find the critical path in a time-constrained activity network. *Computers & Operations Research*, 37(9): 1557–1569, 2010.

[85] I. Hamdi and T. Loukil. Logic-based Benders decomposition to solve the permutation flowshop scheduling problem with time lags. In *Modeling, Simulation and Applied Optimization (ICMSAO), 2013 5th International Conference on*, pages 1–7, 2013.

[86] L. Häme and H. Hakula. A maximum cluster algorithm for checking the feasibility of dial-a-ride instances. *Transportation Science*, 49(2):295–310, 2015.

[87] J. R. Hardin, G. L. Nemhauser, and M. W. P. Savelsbergh. Strong valid inequalities for the resource-constrained scheduling problem with uniform resource requirements. *Discrete Optimization*, 5(1):19 – 35, 2008.

[88] S. Hartmann and D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

[89] J. N. Hooker. Logic-based branch and bound. In *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*, pages 149–161. John Wiley & Sons, Inc., 2000.

[90] J. N. Hooker. Planning and scheduling by logic-based Benders decomposition. *Operations Research*, 55(3):588–602, 2007.

[91] J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

[92] I. Ioachim, J. Desrosiers, I. Dumas, M. M. Solomon, and D. Villeneuve. A request clustering algorithm for door-to-door handicapped transportation. *Transportation Science*, 29(1):63–78, 1995.

[93] S. Jain and P. van Hentenryck. Large neighborhood search for dial-a-ride problems. In J. Lee, editor, *International Conference on Principles and Practice of Constraint Programming — CP 2011*, volume 6876 of *Lecture Notes in Computer Science*, pages 400–413. Springer, Berlin, Heidelberg, 2011.

[94] T. Jatschka. An iterative refinement algorithm for high resolution scheduling problems. Master's thesis, TU Wien, Vienna, Austria, 2017.

[95] J.-J. Jaw, A. R. Odoni, H. N. Psaraftis, and N. H. M. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20(3):243–257, 1986.

[96] O. Kabadurmus and A. E. Smith. Multi-commodity k-splittable survivable network design problems with relays. *Telecommunication Systems*, pages 1–11, 2015.

[97] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[98] L. Khachiyan. A polynomial algorithm for linear programming. *Soviet Mathematics Doklady*, 20(1):191–194, 1979.

[99] V. Klee and G. J. Minty. How good is the simplex algorithm? In *Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin)*, pages 159–175. Academic Press, New York, 1972.

[100] A. Konak. Network design problem with relays: A genetic algorithm with a path-based crossover and a set covering formulation. *European Journal of Operational Research*, 218(3):829–837, 2012.

[101] O. Koné, C. Artigues, P. Lopez, and M. Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13, 2011.

[102] S. Kulturel-Konak and A. Konak. A local search hybrid genetic algorithm approach to the network design problem with relay stations. In S. Raghavan, B. Golden, and E. Wasil, editors, *Telecommunications Modeling, Policy, and Technology*, volume 44 of *Operations Research/Computer Science Interfaces*, pages 311–324. Springer US, 2008.

[103] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[104] Singapore Land Transport Authority. Electronic road pricing (erp). http://www.lta.gov.sg/content/ltaweb/en/roads-and-motoring/managing-traffic-and-congestion/electronic-road-pricing-erp.html, 2018. accessed 2018-09-13.

[105] G. Laporte and M. M. B. Pascoal. Minimum cost path problems with relays. *Computers & Operations Research*, 38(1):165–173, 2011.

[106] E. L. Lawler and J. K. Lenstra. Machine scheduling with precedence constraints. In I. Rival, editor, *Ordered Sets: Proceedings of the NATO Advanced Study Institute held at Banff, Canada, August 28 to September 12*, volume 83 of *NATO Advanced Study Institute Series*, pages 655–675. Springer Netherlands, 1982.

[107] M. Leitner, M. Ruthmair, and G. R. Raidl. Stabilizing branch-and-price for constrained tree problems. *Networks*, 61(2):150–170, 2013.

[108] M. Leitner, I. Ljubić, M. Riedler, and M. Ruthmair. Exact approaches for network design problems with relays. *INFORMS Journal on Computing*, To appear, 2018.

[109] M. Leitner, I. Ljubić, M. Riedler, and M. Ruthmair. Exact approaches for the directed network design problem with relays. *Omega*, submitted.

[110] X. Li, Y. P. Aneja, and J. Huo. Using branch-and-price approach to solve the directed network design problem with relays. *Omega*, 40(5):672–679, 2012.

[111] X. Li, S. Lin, S. Chen, Y. P. Aneja, P. Tian, and Y. Cui. An iterated metaheuristic for the directed network design problem with relays. *Computers & Industrial Engineering*, 113(Supplement C):35–45, 2017.

[112] Y. Li, O. Ergun, and G. L. Nemhauser. A dual heuristic for mixed integer programming. *Operations Research Letters*, 43(4):411–417, 2015.

[113] S. Lin, X. Li, K. Wei, and C. Yue. A tabu search based metaheuristic for the network design problem with relays. In *Service Systems and Service Management (ICSSSM), 2014 11th International Conference on*, pages 1–6, 2014.

[114] I. Ljubić and S. Gollowitzer. Layered graph approaches to the hop constrained connected facility location problem. *INFORMS Journal on Computing*, 25(2): 256–270, 2013.

[115] I. Ljubić, 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. *Math. Program.*, 105(2-3):427–449, 2006.

[116] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[117] A. Lucena, N. Maculan, and L. Simonetti. Reformulations and solution algorithms for the maximum leaf spanning tree problem. *Computational Management Science*, 7(3):289–311, 2010.

[118] R. Macedo, C. Alves, J. M. V. de Carvalho, F. Clautiaux, and S. Hanafi. Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. *European Journal of Operational Research*, 214(3): 536–545, 2011.

[119] A. K. Mackworth. Consistency in networks of relations. *Artificial intelligence*, 8(1): 99–118, 1977.

[120] V. Maniezzo and A. Mingozzi. A heuristic procedure for the multi-mode project scheduling problem based on Benders' decomposition. In J. Weglarz, editor, *Project Scheduling: Recent Models, Algorithms and Applications*, pages 179–196. Springer US, Boston, MA, 1999.

[121] V. Maniezzo, T. Stützle, and S. Voß, editors. *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming.* Springer US, 2010.

[122] J. Maschler, M. Riedler, M. Stock, and G. R. Raidl. Particle therapy patient scheduling: First heuristic approaches. In *PATAT 2016: Proceedings of the 11th International Conference of the Practice and Theory of Automated Timetabling*, pages 223–244, Udine, Italy, 2016.

[123] J. Maschler, T. Hackl, M. Riedler, and G. R. Raidl. An enhanced iterated greedy metaheuristic for the particle therapy patient scheduling problem. In *Proceedings of the 12th Metaheuristics International Conference*, pages 465–474, Barcelona, Spain, 2017.

[124] J. Maschler, M. Riedler, and G. R. Raidl. Particle therapy patient scheduling: Time estimation to schedule sets of treatments. In A. Quesada-Arencibia, J. C. Rodríguez, and R. Moreno-Díaz, editors, *Extended Abstracts of the Sixteenth International Conference on Computer Aided Systems Theory (EUROCAST 2017)*, pages 106–107, Gran Canaria, Spain, 2017.

[125] J. Maschler, M. Riedler, and G. R. Raidl. Particle therapy patient scheduling: Time estimation for scheduling sets of treatments. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Computer Aided Systems Theory – EUROCAST*

2017, volume 10671 of *Lecture Notes in Computer Science*, pages 364–372. Springer International Publishing, 2018.

[126] G. B. Mertzios, I. Sau, M. Shalom, and S. Zaks. Placing regenerators in optical networks to satisfy multiple sets of requests. *IEEE/ACM Transactions on Networking*, 20(6):1870–1879, 2012.

[127] R. Mohr and T. C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28(2):225–233, 1986.

[128] R. H. Möhring, A. S. Schulz, F. Stork, and M. Uetz. Solving project scheduling problems by minimum cut computations. *Management Science*, 49(3):330–350, 2003.

[129] I. Nath, M. Chatterjee, and U. Bhattacharya. A survey on regenerator placement problem in translucent optical network. In *2014 International Conference on Circuits, Systems, Communication and Information Technology Applications (CSCITA)*, pages 408–413, 2014.

[130] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience, New York, NY, USA, 1988.

[131] K. Neumann, C. Schwindt, and J. Zimmermann. *Project Scheduling with Time Windows and Scarce Resources.* Springer Berlin Heidelberg, 2003.

[132] M. Palpant, C. Artigues, and P. Michelon. LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 131(1-4):237–257, 2004.

[133] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Dover Publications, Mineola, NY, USA, 1998.

[134] J. Paquette, F. Bellavance, J.-F. Cordeau, and G. Laporte. Measuring quality of service in dial-a-ride operations: The case of a Canadian city. *Transportation*, 39 (3):539–564, 2012.

[135] S. N. Parragh, K. F. Dörner, and R. F. Hartl. A survey on pickup and delivery models: Part II: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58:81–117, 2008.

[136] S. Pelletier, O. Jabali, and G. Laporte. 50th anniversary invited article—goods distribution with electric vehicles: Review and research perspectives. *Transportation Science*, 50(1):3–22, 2016.

[137] 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.

[138] H. N. Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2): 130–154, 1980.

[139] 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*, pages 71–102. Springer US, Boston, MA, 2010.

[140] X. Qiu and S. Feuerriegel. A multi-vehicle profitable pickup and delivery selection problem with time windows. In *Proceedings of the European Conference on Information Systems (ECIS) 2014*, 2014.

[141] X. Qiu, S. Feuerriegel, and D. Neumann. Making the most of fleets: A profit-maximizing multi-vehicle pickup and delivery selection problem. *European Journal of Operational Research*, 259(1):155–168, 2017.

[142] S. Raghavan and D. Stanojević. Branch and price for wdm optical networks with no bifurcation of flow. *INFORMS Journal on Computing*, 23(1):56–74, 2011.

[143] Q. Rahman, S. Bandyopadhyay, and Y. Aneja. Optimal regenerator placement in translucent optical networks. *Optical Switching and Networking*, 15:134–147, 2015.

[144] R. Rahmaniani, T. G. Crainic, M. Gendreau, and W. Rei. The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817, 2017.

[145] G. R. Raidl, T. Baumhauer, and B. Hu. Boosting an exact logic-based Benders decomposition approach by variable neighborhood search. In *Proceedings of the 3rd International Conference on Variable Neighborhood Search*, volume 47 of *Electronic Notes in Discrete Mathematics*, pages 149–156. Elsevier, 2014.

[146] G. R. Raidl, T. Baumhauer, and B. Hu. Speeding up logic-based Benders' decomposition by a metaheuristic for a bi-level capacitated vehicle routing problem. In M. J. Blesa, C. Blum, and S. Voß, editors, *Hybrid Metaheuristics*, volume 8457 of *Lecture Notes in Computer Science*, pages 183–197. Springer International Publishing, 2014.

[147] M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In M. Gendreau and J. Potvin, editors, *Handbook of Metaheuristics*, pages 283–319. Springer US, Boston, MA, 2010.

[148] M. Riedler and G. R. Raidl. Solving a selective dial-a-ride problem with logic-based Benders decomposition. *Computers & Operations Research*, 96:30–54, 2018.

[149] M. Riedler, T. Jatschka, J. Maschler, and G. R. Raidl. An iterative time-bucket refinement algorithm for a high-resolution resource-constrained project scheduling problem. *International Transactions in Operational Research*, 2017. doi: 10.1111/itor.12445. available online.

[150] M. Riedler, M. Ruthmair, and G. R. Raidl. Strategies for iteratively refining layered graph models. In *Hybrid Metaheuristics: 11th International Workshop, HM 2019*, Lecture Notes in Computer Science. Springer International Publishing, to appear.

[151] S. Ropke and J.-F. Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, 2009.

[152] S. Ropke, J.-F. Cordeau, and G. Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007.

[153] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[154] M. Ruthmair. *On Solving Constrained Tree Problems and an Adaptive Layers Framework*. PhD thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Vienna, Austria, 2012.

[155] 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. J. Wöginger, editors, *Integer Programming and Combinatoral Optimization. IPCO 2011*, volume 6655 of *Lecture Notes in Computer Science*, pages 376–388, Berlin, Heidelberg, 2011. Springer.

[156] M. Schneider, A. Stenger, and D. Goeke. The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science*, 48(4):500–520, 2014.

[157] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[158] T. R. Sexton and L. D. Bodin. Optimizing single vehicle many-to-many operations with desired delivery times: I. Scheduling. *Transportation Science*, 19(4):378–410, 1985.

[159] T. R. Sexton and L. D. Bodin. Optimizing single vehicle many-to-many operations with desired delivery times: II. Routing. *Transportation Science*, 19(4):411–435, 1985.

[160] C. S. Sung and S. H. Song. Branch-and-price algorithm for a combined problem of virtual path establishment and traffic packet routing in a layered communication network. *Journal of the Operational Research Society*, 54(1):72–82, 2003.

[161] E. S. Thorsteinsson. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In *Principles and Practice of Constraint Programming — CP 2001*, volume 2239 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2001.

[162] T. A. M. Toffolo, H. G. Santos, M. A. M. Carvalho, and J. A. Soares. An integer programming approach to the multimode resource-constrained multiproject scheduling problem. *Journal of Scheduling*, 19(3):295–307, 2016.

[163] T. T. Tran and J. C. Beck. Logic-based Benders decomposition for alternative resource scheduling with sequence dependent setups. In *Proceedings of the 20th European Conference on Artificial Intelligence*, ECAI'12, pages 774–779, Amsterdam, The Netherlands, The Netherlands, 2012. IOS Press. ISBN 978-1-61499-097-0.

[164] H. Üster and P. Kewcharoenwong. Strategic design and analysis of a relay network in truckload transportation. *Transportation Science*, 45:505–523, 2011.

[165] H. Üster and N. Maheshwari. Strategic network design for multi-zone truckload shipments. *IIE Transactions*, 39(2):177–189, 2007.

[166] P. van Hentenryck, Y. Deville, and C.-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57(2):291–321, 1992.

[167] 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.

[168] X. Wang and A. C. Regan. On the convergence of a new time window discretization method for the traveling salesman problem with time window constraints. *Computers & Industrial Engineering*, 56(1):161–164, 2009.

[169] J. Westerlund, M. Hästbacka, S. Forssell, and T. Westerlund. Mixed-Time mixed-integer linear programming scheduling model. *Industrial & Engineering Chemistry Research*, 46(9):2781–2796, 2007.

[170] D. Wheatley, F. Gzara, and E. Jewkes. Logic-based Benders decomposition for an inventory-location problem with service constraints. *Omega*, 55:10–23, 2015.

[171] R. Wolfler Calvo and A. Colorni. An effective and fast heuristic for the dial-a-ride problem. *4OR*, 5(1):61–73, 2007.

[172] L. A. Wolsey. *Integer Programming.* Wiley-Interscience, New York, NY, USA, 1998.

[173] Y. Xiao and A. Konak. A variable neighborhood search for the network design problem with relays. *Journal of Heuristics*, pages 1–28, 2017.

[174] B. Yıldız and O. E. Karaşan. Regenerator location problem and survivable extensions: A hub covering location perspective. *Transportation Research Part B: Methodological*, 71:32–55, 2015.

[175] B. Yıldız and O. E. Karaşan. Regenerator location problem in flexible optical networks. *Operations Research*, 65(3):595–620, 2017.

[176] B. Yıldız, O. Arslan, and O. E. Karaşan. A branch and price approach for routing and refueling station location model. *European Journal of Operational Research*, 248(3):815–826, 2016.

[177] B. Yıldız, O. E. Karaşan, and H. Yaman. Branch-and-price approaches for the network design problem with relays. *Computers & Operations Research*, 92:155–169, 2018.

[178] Z. Zhu, X. Chen, F. Ji, L. Zhang, F. Farahmand, and J. P. Jue. Energy-efficient translucent optical transport networks with mixed regenerator placement. *Journal of Lightwave Technology*, 30(19):3147–3156, 2012.

CURRICULUM VITAE

PERSONAL INFORMATION

name	Martin Riedler
date and place of birth	August 1, 1989, Linz, Austria
email	martin_riedler@gmx.at
phone	+43 (680) 243 937 5

WORK EXPERIENCE

2014–2018 University Assistant

TU Wien
Institute of Logic and Computation,
Algorithms and Complexity Group.

2015–2017 Project Assistant

TU Wien
Research collaboration on particle therapy patient scheduling with cancer treatment center EBG MedAustron GmbH, Wiener Neustadt, Austria.

2008 Recruit

Austrian Armed Forces
Mandatory military service at the Ostarrichikaserne, Amstetten, Austria.

EDUCATION

2014–Present TU Wien, Vienna, Austria

Doctoral programme in Engineering Sciences
Field of Study: Computer Science
Thesis: *Advances in Decomposition Approaches for Mixed Integer Linear Programming*
Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther R. Raidl

2012–2014 TU Wien, Vienna, Austria

Master of Science
Field of Study: Computational Intelligence
Thesis: *Exact Approaches to the Network Design Problem with Relays*
Advisors: Univ.Lektorin Mag.rer.nat. Dr.techn. Ivana Ljubić & Dipl.-Ing. Dr.techn. Markus Leitner & Dipl.-Ing. Dr.techn. Mario Ruthmair

2009–2012 TU Wien, Vienna, Austria

Bachelor of Science
Field of Study: Software & Information Engineering

TEACHING

2014WS–2018WS

Algorithmics
Introduction to mathematical programming.

2017WS

Heuristic Optimization Techniques
Heuristic algorithms for solving optimization problems.

2015SS–2018SS

Algorithms and Data Structures
Basics of algorithm analysis, algorithms, and data structures.

2014WS–2018SS

Scientific Methodologies

Basic seminar on algorithms.

2014WS–2018SS

Seminar on Algorithms

Advanced seminar on algorithms.

PUBLICATIONS

submitted

Omega

M. Leitner, I. Ljubić, M. Riedler, and M. Ruthmair. Exact approaches for the directed network design problem with relays. *Omega*, submitted

to appear

INFORMS Journal on Computing

M. Leitner, I. Ljubić, M. Riedler, and M. Ruthmair. Exact approaches for network design problems with relays. *INFORMS Journal on Computing*, to appear

2019

Lecture Notes in Computer Science

M. Riedler, M. Ruthmair, and G. R. Raidl. Strategies for iteratively refining layered graph models. In *Hybrid Metaheuristics: 11th International Workshop, HM 2019*, Lecture Notes in Computer Science. Springer International Publishing, 2019. to appear

2018

Computers & Operations Research

M. Riedler and G. R. Raidl. Solving a selective dial-a-ride problem with logic-based Benders decomposition. *Computers & Operations Research*, 96:30–54, 2018

2017

International Transactions in Operational Research

M. Riedler, T. Jatschka, J. Maschler, and G. R. Raidl. An iterative time-bucket refinement algorithm for a high-resolution resource-constrained project scheduling problem. *International Transactions in Operational Research*, 2017. doi: 10.1111/itor.12445. available online