Faster algorithms for Steiner tree and related problems: From theory to practice

vorgelegt von
M. Sc.
Daniel Markus Rehfeldt
ORCID: 0000-0002-2877-074X

von der Fakultät II – Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Dietmar Hömberg
Gutachter:	Prof. Dr. Thorsten Koch
Gutachter:	Prof. Dr. Eduardo Uchoa
Gutachter:	Dr. Renato Werneck

Tag der wissenschaftlichen Aussprache: 16. August 2021

Berlin 2021

Abstract

The Steiner tree problem in graphs (SPG) is one of the most studied problems in combinatorial optimization. Part of its theoretical appeal might be attributed to the fact that the SPG generalizes two other classic optimization problems: Shortest paths, and minimum spanning trees. On the practical side, many applications can be modeled as SPG or closely related problems. The SPG has seen impressive theoretical advancements in the last decade. However, the state of the art in (practical) exact SPG solution, set in a series of milestone papers by Polzin and Vahdati Daneshmand, has remained largely unchallenged for almost 20 years. While the DIMACS Challenge 2014 and the PACE Challenge 2018 brought renewed interest into the exact solution of SPGs, even the best new solvers fall far short of reaching the state of the art.

This thesis seeks to once again advance exact SPG solution. Since many practical applications are not modeled as pure SPGs, but rather as closely related problems, this thesis also aims to combine SPG advancements with improvement in the exact solution of such related problems. Initially, we establish a broad theoretical basis to guide the subsequent algorithmic developments. In this way, we provide various new theoretical results for SPG and well-known relatives such as the maximum-weight connected subgraph problem. These results include the strength of linear programming relaxations, polyhedral descriptions, and complexity results. We go on to introduce many algorithmic components such as reduction techniques, cutting planes, graph transformations, and heuristics—both for SPG and related problems. Many of these methods and techniques are provably stronger than previous results from the literature. For example, we introduce a new reduction concept that is strictly stronger than the well-known and widely used bottleneck Steiner distance. We also provide theoretical analyses (e.g. concerning complexity) of the new algorithms. The individual components are combined in an exact branch-and-cut algorithm. Notably, all problem classes can be handled by a single branch-and-cut kernel.

As a result, we obtain an exact solver for SPG and 14 related problems. The new solver is on each of the 15 problem classes faster than all other (problem-specific) solvers from the literature, often by orders of magnitude. In particular, the solver outperforms the long-reigning state-of-the-art solver for SPG. Finally, many benchmark instances from the literature for several problem classes can be solved for the first time to optimality—some containing millions of edges. These problem classes include the SPG, the prize-collecting Steiner tree problem, the maximum-weight connected subgraph problem, and the Euclidean Steiner tree problem.

Zusammenfassung

Das Steinerbaumproblem in Graphen (SPG) ist eines der am besten untersuchten Probleme der kombinatorischen Optimierung. Das große theoretische Interesse für das Problem kann auch darauf zurückgeführt werden, dass das SPG zwei weitere klassische Optimierungsprobleme verallgemeinert: Kürzeste Wege und Minimale Aufspannende Bäume. Auf der anderen (Praxis orientierten) Seite können viele Anwendungen in Industrie ond Forschung als SPG oder verwandte Probleme modelliert werden. Das SPG hat in den letzten 10 Jahren beeindruckende theoretische Fortschritte erfahren. Im Gegensatz dazu hat es in der exakten SPG-Lösung seit fast 20 Jahren praktisch keinen Fortschritt gegeben. Wenngleich die DIMACS Challenge 2014 und die PACE Challenge 2018 neues Interesse für die exakte Lösung von SPGs in der Forschungsgemeinschaft weckten, blieben selbst die besten neuen Verfahren für SPG weit hinter der führenden Lösungstechnologie zurück.

Diese Arbeit wurde mit dem Ziel gestartet, die exakte Lösung von SPGs nun erneut voranzubringen. Da viele praktische Anwendungen nicht als reine SPGs, sondern als eng verwandte Probleme modelliert werden, zielt diese Arbeit auch darauf ab, Fortschritte in der Lösung von SPGs mit Verbesserungen bei der exakten Lösung verwandter Probleme zu kombinieren. Die Arbeit beginnt mit dem Errichten eines breiten theoretischen Fundaments, auf welches die darauffolgenden algorithmischen Entwicklungen aufgebaut werden. So werden etwa verschiedene neue theoretische Ergebnisse für SPG und bekannte Verwandte wie das Maximum-Weight Connected Subgraph Problem eingeführt. Diese Ergebnisse beinhalten etwa Komplexitätsresultate für die betrachteten Probleme, oder stärkere polyedrische Beschreibungen des Lösungsraums. Anschließend werden diverse algorithmische Komponenten wie Reduktionstechniken, Schnittebenen, Graphentransformationen und Heuristiken vorgestellt - sowohl für SPG als auch für verwandte Probleme. Viele dieser Methoden und Techniken sind beweisbar stärker als bisherige Ergebnisse aus der Literatur. Weiterhin werden auch theoretische Analysen (z.B. bezüglich der Komplexität) der neuen Algorithmen gegeben. Die einzelnen Komponenten werden schließlich in einem exakten Branch-and-Cut-Algorithmus kombiniert. Herauszustellen ist, dass alle Problemklassen mit einem einzigen Branch-and-Cut Kern gelöst werden können.

Das praktische Ergebnis dieser Arbeit ist ein exakter Löser für SPG und 14 weitere verwandte Probleme. Der neue Löser ist auf jeder dieser 15 Problemklassen schneller als alle anderen (problemspezifischen) Löser aus der Literatur, oft um Größenordnungen. Insbesondere liefert der neue Löser bessere Ergebnisse als der eingangs erwähnte bisher führende SPG Löser. Weiterhin können viele Benchmark-Instanzen aus der Literatur für mehrere Problemklassen zum ersten Mal gelöst werden - dies beeinhaltet Instanzen mit Millionen an Kanten. Zu diesen Problemklassen gehören das SPG, das prize-collecting SPG, das Maximum-Weight Connected Subgraph Problem und das Euklidische Steinerbaumproblem.

Acknowledgements

First of all, I would like to thank Thorsten Koch for awaking my interest in Steiner tree problems, and for his support and advice during the creation of this thesis. In particular, I would like to thank him for giving me the opportunity to work almost exclusively on the creation of this thesis in the last one and a half years of my PhD studies. I am also very grateful to Eduardo Uchoa and Renato Werneck for agreeing to be part of my PhD committee. I would furthermore like to thank Dietmar Hömberg for agreeing to chair my PhD committee.

I would like to thank my colleagues at TU Berlin and Zuse Institute Berlin (ZIB) for the nice working atmosphere. Even though the project I was working on for most of my time at TU and ZIB had nothing to do with Steiner trees, I learned a lot about programming, (parallel) algorithms, and perseverance, which turned out to also be quite useful for the creation of this thesis. Also, I would like to thank ZIB for allowing me to use their computing clusters for the computational experiments presented in this thesis. I would also like to thank Gerald Gamrath, Ambros Gleixner, Nils Kempke, Benjamin Müller, and Mark Turner for proof-reading parts of this thesis and providing many helpful comments. I would like to thank Yuji Shinano for his collaboration on the parallelization part of this thesis. I would like to thank Cees Duin for sending me a hard copy of his dissertation (which had been written in the old pre-PDF days) from the Netherlands. I would also like to acknowledge the work of the many people who worked on Steiner tree and related problems, and on whose contributions I could built on. Last, but most of all, I would like to thank my family for their support throughout the creation process of this thesis.

Contents

Introduction

Given an undirected graph with non-negative edge weights and a subset of vertices called *terminals*, the *Steiner tree problem in graphs* (SPG) is to find a tree of minimum weight that contains all terminals. The SPG is a classic $\mathcal{NP}$-hard problem, and one of the most studied problems in combinatorial optimization. The geometrical origins of the SPG can be traced back to Pierre de Fermat's famous treatise *Methodus ad disquirendam maximam et minimam* from 1638, and the problem was rediscovered by the likes of Carl Friedrich Gauß and Vojtěch Jarník in the following centuries[1].

Part of the appeal of the SPG might be attributed to the fact that it "lies between" two other classic optimization problems: If there are exactly two terminals, the SPG reduces to the *shortest-path problem*, if all vertices are terminals, the SPG reduces to the *minimum-spanning tree problem*. However, in contrast to the SPG, for both of these problems polynomial-time algorithms are known. Interestingly, if the SPG "stays close enough" to either of these problems, i.e., if either the number of terminals, or the number of non-terminals is bounded, it can also be solved in polynomial time[2]. On the practical side, the large research interest in the SPG can be motivated by the numerous and surprisingly diverse practical applications that can be modeled as SPG or closely related problems. Two classic application areas are network design problems and the design of integrated circuits. Other, more recent, areas are for example systems biology and machine learning.

The SPG has seen numerous theoretical advances in the last 10 years, bringing forth significant improvements for example in complexity and approximability. Indeed, the SPG can be considered a flagship problem in both of these research areas. However, when it comes to (practical) exact algorithms, the picture is significantly more bleak. After flourishing in the 1990s and early 2000s, algorithmic advances came to a staggering halt with the joint PhD theses of Polzin and Vahdati Daneshmand almost 20 years ago. The authors introduced a wealth of new results and algorithms for SPG, and combined them in a computer program that drastically outperformed all previous results from the literature. We will refer to such computer programs for mathematical optimization problems as *solvers*. While there have been some success stories for special classes of SPG instances in the meantime, on the vast majority of benchmark instances Polzin and Vahdati Daneshmand have stayed well out of reach. For example, even the best solvers from the 11th DIMACS Challenge in 2014, dedi-

[1] See e.g., Brazil et al. (2014)
[2] See e.g., Hwang et al. (1992)

cated to Steiner tree problems, are orders of magnitude slower on many benchmark instances, and solve far fewer instances to optimality. Much the same can be said of the various solvers participating at the 3rd PACE Challenge in 2018: Even on the special class of SPG instances used at the PACE Challenge the state of the art solver by Polzin and Vahdati Daneshmand remained largely unchallenged. However, this solver is not publicly available.

Against this backdrop, this thesis aims at advancing once again the state of the art in solving SPGs to optimality. But what, one might ask, is the interest in further improving the solution of Steiner tree problems? We start with some practical points. As just mentioned, many applications are modeled as SPG and related problems—and in the age of big data such models naturally become larger. 15 years ago, the largest SPG instance from the literature had roughly 200 thousand edges. In contrast, the largest Steiner tree instances considered (and solved) in this thesis have up to 10 million edges. A huge leap for an $\mathcal{NP}$-hard problem. Moreover, classic optimization problems such as the traveling salesman problem or the SPG, have often been a testing ground for techniques that can later be used for other problem types. And indeed, a central contribution of this thesis is to extend results for the SPG to related problems.

This thesis can also be seen in the larger picture of the tremendous algorithmic progress of mixed-integer programming (MIP) algorithms in the last decades. MIP solvers exhibit an impressive performance and have become a standard industry tool.[3] Thus, one might question the need of strenuously hand-tailored, problem-specific algorithms and implementations. However, if one starts to model SPG instances as MIPs (using one of the many MIP formulations from the literature), and tries to solve them with leading commercial MIP solvers, one quickly realizes that already medium-sized instances can often not be solved even after weeks of computation (or even run out of memory). In contrast, such instances can be solved in fractions of a second by the SPG solver presented in this thesis.

Besides these practical points, there is also a significant theoretical interest in advancing the exact solution of SPGs. First of all, such an advancement naturally spawns new underlying theoretical results, for example in complexity or polyhedral theory. Also, many techniques developed in this thesis are, arguably, theoretically interesting in their own right, and furthermore lead to several new ($\mathcal{NP}$-hard) optimization problems. Finally, the strong practical results achieved in this thesis serve to show the limits of classic complexity theory: It is possible to solve the overwhelming majority of large-scale SPG benchmark instances to optimality within minutes, including those with hundreds of thousand of edges. We are even able to solve many instances with millions of edges. Indeed, such practical success stories have contributed to the huge prominence of the field of parametrized complexity, which allows for a finer scale classification of complexity. As another example, the tremendous practical success of preprocessing techniques in many combinatorial optimization problems (including SPG) has given rise to the field of *kernelization*[4], which is a flourishing area in theoretical computer science.

---

[3] See e.g., Koch et al. (2013)

[4] See e.g., Fomin et al. (2019b)

The underlying assumption of this thesis is that practical advancements best go hand-in-hand with a solid theoretical understanding of the utilized techniques and algorithms. Thus, we will move from *theory to practice* and base algorithm developments on various new theoretical results, such as the fixed-parameter tractability of the considered problems, or the tightness of their integer programming formulations. To compete with the state of the art in exact SPG solution, we introduce a wide range of intricate algorithmic components, which are finally combined in an exact algorithm. As to practical usability, one also observes that many real-world problems are not modeled as pure SPGs, but rather as closely related problems. This observation also explains the multitude of Steiner tree relatives and generalizations found in the literature. Thus, a significant part of this thesis is devoted to extending and complementing the new theoretical and practical SPG results such that they can be used for several close relatives of the SPG. Finally, the new algorithms developed in this thesis have been implemented in an exact Steiner tree solver. The practical performance of this solver is demonstrated on a wide range of well-established benchmark sets, often originating from practical applications.

Structure and main contributions

This thesis can be roughly divided into two parts. The first part offers an in-depth treatment of SPG and two well-known related problems. For each problem, an extensive theoretical foundation is established, from which various new solution techniques and algorithms are developed. By combining these algorithmic components, we aim to push the limits of (computational) tractability for all three problem classes. We highlight and exploit the strong interrelations between the three problems, but also establish various novel problem-specific results. Each of the three problems is devoted an individual chapter.

The second part of this thesis, starting with Chapter 5, takes a turn towards mostly practical issues. We show how to use the previously established results to readily solve many further related problems. Next, we discuss implementation issues, such as data structures, and show how to parallelize some of the previously introduced algorithms.

The coherence of the algorithmic treatment of all problem classes in this thesis is highlighted by the fact that only a single branch-and-cut kernel is used for all 15 problem classes shown in Table 1 (although also many problem-specific algorithms are used). To this end, an important ingredient is constituted by the (often new) problem transformations depicted in Figure 1 and Figure 2.

In more detail, the structure of the thesis is as follows.

- Chapter 1 provides preliminaries, such as notation.

- Chapter 2 is concerned with the SPG. The chapter starts with a theoretical analysis of two widely used integer programming formulations for SPG. We show conditions for the linear programming relaxations to be exact, and compare the relative strength of the formulations. Subsequently, many new algorithmic components such as reduction techniques, conflicts, and heuristics are introduced.

Table 1: The 15 problem classes considered in this thesis.

Abbreviation	Problem Name
DCSTP	*D*egree-*c*onstrained *S*teiner *t*ree *p*roblem
FTSTP	*F*ull *t*erminal *S*teiner *t*ree *p*roblem
GSTP	*G*roup *S*teiner *t*ree *p*roblem
HCDSTP	*H*op-*c*onstrained *d*irected Steiner tree problem
MWCSP	*M*aximum-*w*eight *c*onnected *s*ubgraph *p*roblem
MWCSPB	*M*aximum-*w*eight *c*onnected *s*ubgraph *p*roblem with *b*udget
NWSTP	Node-weighted *S*teiner *t*ree problem
OARSMT	*O*bstacle-*a*voiding *r*ectilinear *S*teiner *m*inimum *t*ree problem
PCSTP	*P*rize-*c*ollecting *S*teiner *t*ree *p*roblem
PTSTP	*P*artial *t*erminal *S*teiner *t*ree *p*roblem
RMWCSP	*R*ooted *m*aximum-*w*eight *c*onnected *s*ubgraph *p*roblem
RPCSTP	*R*ooted *p*rize-*c*ollecting *S*teiner *t*ree *p*roblem
RSMT	*R*ectilinear *S*teiner *m*inimum *t*ree problem
SAP	*S*teiner *a*rborescence *p*roblem
SPG	*S*teiner tree *p*roblem in *g*raphs

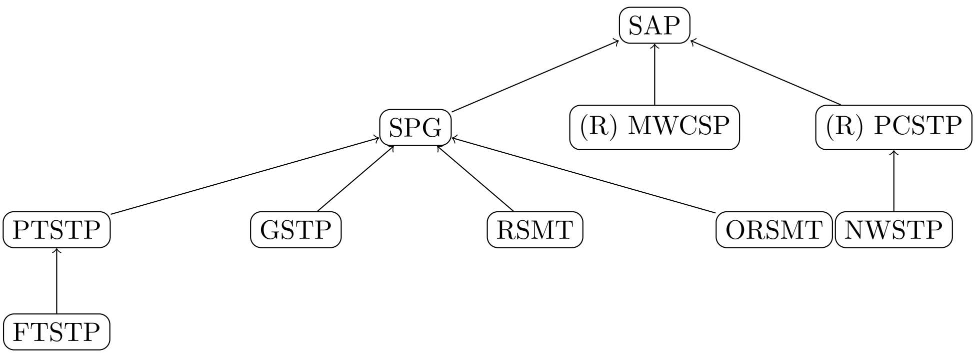


Figure 1: Transformations between problem classes used for computational solution in this thesis.

Several of these methods and techniques are provably stronger than well-known results from the literature. The various components are combined in an exact SPG algorithm. The chapter closes with computational results, including a comparison with the state of the art in exact SPG solution.

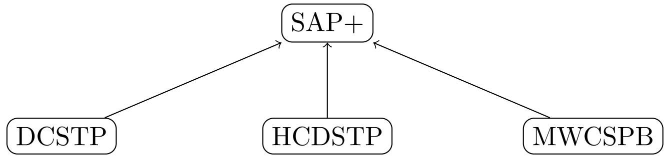


Figure 2: Transformations to SAP with additional constraints used for computational solution in this thesis.

- Chapter 3 covers the maximum-weight connected subgraph problem, a well-known relative of the SPG, which has (real-valued) vertex instead of (non-negative) edge weights. We start with a theoretical analysis of the strength of integer programming formulations, and also give related polyhedral results. Based on the strongest of these formulations, various new solution techniques and algorithms are introduced. These developments include extensions of results from the previous chapter, but also novel contributions—such as graph transformations. Finally, we assemble the individual algorithms within a branch-and-cut framework. Computational results of the resulting solver are also given.

- Chapter 4 discusses an important generalization of both of the previously considered problems: the prize-collecting Steiner tree problem. Initially, we provide new complexity results, showing in particular that the problem is fixed-parameter tractable. We go on to introduce and analyze several new algorithms and techniques. A notable example is a new implication concept that allows us to identify vertices that are contained in all optimal solutions. As in the previous two chapters, these algorithms are finally combined within an exact branch-and-cut solver.

- Chapter 5 shows how to readily extend the algorithmic base established in the previous three chapters to solve further related problems. The focus is on breadth rather than depth: We show how to achieve strong results for many problem classes with little additional algorithmic and implementational effort.

- Chapter 6 contains implementation details of several of the most important algorithms introduced so far—including various data structures and auxiliary algorithms. Furthermore, this chapter provides details on both shared- and distributed memory parallelizations of our branch-and-cut framework.

- Chapter 7 closes the thesis, providing a conclusion as well as suggestions for further research on the topics discussed so far.

Complementary to the theoretical results given in this thesis, a notable practical contribution is the implementation of an exact Steiner tree solver. The new solver is on each of the 15 problem classes it can handle faster than all other (problem-specific) solvers from the literature[5], often by orders of magnitude. In particular, it consistently

[5] It should be noted, though, that for RSMT the initial full Steiner tree generation is not done by our solver

outperforms the long-reigning state-of-the-art solver for SPG. Furthermore, many benchmark instances from the literature for several problem classes can be solved for the first time to optimality—some containing millions of edges. These problem classes include the SPG, the prize-collecting Steiner tree problem, the maximum-weight connected subgraph problem, and the Euclidean Steiner tree problem. We further discuss the software contribution in the following section.

Publications, software, and outreach

Substantial parts of this thesis have been published in or submitted to the following peer-reviewed journals and conference proceedings. Articles submitted to, accepted by, or published in international journals are listed below:

- An article about the theoretical aspects of Chapter 2 and 3 has been submitted to *Networks*. This article is joint work with Thorsten Koch.
- Parts of Chapter 3 have been published in *SIAM Journal on Optimization* (Rehfeldt and Koch, 2019). The article is joint work with Thorsten Koch.
- Parts of Chapter 3 and Chapter 4 have been published in *Journal on Computational Mathematics* (Rehfeldt and Koch, 2018a). The article is joint work with Thorsten Koch.
- Parts of Chapter 3 and Chapter 4 have been published in *Networks* (Rehfeldt et al., 2019). The article is joint work with Thorsten Koch and Stephen Maher. However, we note that several of the algorithms from Rehfeldt et al. (2019) have been replaced by (provably) stronger counterparts in this thesis.
- Parts of Chapter 4 have been accepted for publication by the *INFORMS Journal on Computing*. See Rehfeldt and Koch (2020) for a preprint. The article is joint work with Thorsten Koch.
- Parts of Chapter 5.1 have been published in *IEEE/ACM Transactions on Networking* (Sun et al., 2020). However, the contribution of the author of this thesis is small. It is restricted to an extension of the solver developed in this thesis for the Steiner tree variant considered in the article, as well as computational results.

Furthermore, a publication containing large parts of Chapter 6 and parts of Chapter 2 is in preparation, and will be submitted to an international journal. Publications in conference proceedings are as follows:

- An overview of the solver developed for this thesis has been published in the Operations Research Proceedings 2017 (Rehfeldt and Koch, 2018b). This publication is joint work with Thorsten Koch.
- Results from several chapters of this thesis have been published in the proceedings of the 7th International Conference on High Performance Scientific Computing (Rehfeldt et al., 2021). This publication is joint work with Yuji Shinano and Thorsten Koch.

- Parts of Chapter 2 have been published in the proceedings of the 22nd Conference on Integer Programming and Combinatorial Optimization (IPCO) (Rehfeldt and Koch, 2021). An extended version has been submitted to *Mathematical Programming B*. This manuscript is joint work with Thorsten Koch.

- Parts of Chapter 6 have been published in the proccedings of CPAIOR 2019 (Shinano et al., 2019b). This publication is joint work with Yuji Shinano and Thorsten Koch.

- Results from Chapter 6.3 have been published in the proceedings of the 9th IEEE Workshop Parallel / Distributed Combinatorics and Optimization (Shinano et al., 2019a). This publication is joint work with Yuji Shinano and Tristan Gally.

The software developed in the course of this thesis has been combined in the Steiner tree solver SCIP-Jack. A previous version of SCIP-Jack is freely available for academic use as part of the SCIP Optimization Suite (Gamrath et al., 2020). We note that a forerunner of SCIP-Jack existed already prior to the start of this thesis (with the author of this thesis being the main developer). However, more than 95% of the current SCIP-Jack version has been newly implemented as part of this thesis. This current version will be included in the next major release of the SCIP Optimization Suite. The SCIP-Jack version included in the latest SCIP Optimization Suite has been used in several research projects, e.g. van den Boogaart (2018); Iwata and Shigemura (2019); Peters (2021), and has already received notable recognition in the recent literature on Steiner tree and related problems, see e.g. Ljubic (2020). Furthermore, this version of SCIP-Jack successfully competed in the 3rd *Parameterized Algorithms and Computational Experiments Challenge* (Bonnet and Sikora, 2019), dedicated to fixed-parameter tractable SPGs. Even though SCIP-Jack did not include any special algorithms for such problems, it reached first (Track B), second (Track A), and third (Track C) place in the three tracks of the challenge. We note that the current version of SCIP-Jack also outperforms all other competitors in Track A and C, see Section 2.7.2 for more details.

Finally, we note that SCIP-Jack is actively being used in several industrial projects, for example at Open Grid Europe, one of Europe's largest transmission systems operators.

Chapter 1

Preliminaries

This chapter provides preliminaries that are relevant at multiple places of this thesis. Section 1.1 introduces basic notation and concepts. Section 1.2 provides information on the computational experiments conducted in this thesis.

1.1 Notation and basic concepts

Most notation will be introduced as needed. To keep the individual chapters largely self-contained, we will also allow some redundancies and occasionally reintroduce basic concepts. In the following, we merely provide the most common notation, as well as several simple concepts. A list of frequently used abbreviations and names can be found on page 187.

1.1.1 Miscellaneous

The sets of real, rational, and integer numbers are denoted by $\mathbb{R}$, $\mathbb{Q}$, and $\mathbb{Z}$, respectively, and their nonnegative versions by $\mathbb{R}_{\geq 0}$, $\mathbb{Q}_{\geq 0}$, and $\mathbb{Z}_{\geq 0}$. For the strictly positive versions, we write $\mathbb{R}_{>0}$, $\mathbb{Q}_{>0}$, and $\mathbb{Z}_{>0}$. The set of natural numbers is denoted by $\mathbb{N}$. We assume $0 \notin \mathbb{N}$, so $\mathbb{N} = \mathbb{Z}_{>0}$. We write $\mathbb{N}_0 := \mathbb{Z}_{\geq 0}$. For the cardinality of a finite set S we write $|S|$. Throughout this thesis, all vectors are understood to be column vectors. The transpose of a vector or matrix is denoted by the superscript '^{T}'. Let $\mathbb{K}$ be a field. For any function $x : M \mapsto \mathbb{K}$ with M finite, and any $M' \subseteq M$ define $x(M') := \sum_{i \in M'} x(i)$. For an integer programming (IP) formulation F we denote its optimal objective value by $v(F)$. For the linear programming (LP) relaxation of F we denote by $v_{LP}(F)$ the optimal objective value, and by $\mathcal{P}_{LP}(F)$ the set of feasible points. For background information on integer and linear programming see for example Schrijver (1998).

1.1.2 Graph theory

The general graph notation used in this thesis is largely in accordance with Bondy and Murty (2008). It deviates at several places to, for example, be in line with the Steiner tree literature, or to facilitate the presentation of Steiner tree specific concepts.

For a given undirected graph $G = (V, E)$ we define $n := |V|$ and $m := |E|$, and for a directed graph $D = (V, A)$ likewise $n := |V|$ and $m := |A|$. In this thesis, graphs are always simple, i.e. without parallel edges or arcs, and finite, i.e. $n, m < \infty$ holds. We refer to the vertices and edges of a subgraph $G' \subseteq G$ as $V(G')$ and $E(G')$ respectively, and analogously to the vertices and arcs of a directed subgraph $D' \subseteq D$ as $V(D')$ and $A(D')$. An (undirected) edge between vertices $v, w \in V$ is denoted by $\{v, w\}$, a (directed) arc by (v, w). For $U \subseteq V$ we define

$$E[U] := \big\{\{v, u\} \in E \mid v, u \in U\big\}.$$

For $U \subseteq V$ define the induced *edge cut* as $\delta(U) := \{\{u, v\} \in E \mid u \in U, v \in V \setminus U\}$; for a directed graph $D = (V, A)$ define $\delta^+(U) := \{(u, v) \in A \mid u \in U, v \in V \setminus U\}$ and $\delta^-(U) := \delta^+(V \setminus U)$. We also write δ_G or δ_D^+, δ_D^- to distinguish the underlying graph. For a single vertex v we use the short-hand notation $\delta(v) := \delta(\{v\})$, and accordingly for directed graphs. In the undirected case the *degree* of any $v \in V$ is defined as $|\delta(v)|$, i.e. the number of incident edges. In the directed case we distinguish between the *indegree* $|\delta^-(v)|$ and the *outdegree* $|\delta^+(v)|$.

Paths will be considered as subgraphs, and the subpath of a path Q between two vertices $v, w \in V(Q)$ will be denoted by $Q(v, w)$. A path between two vertices v, w will be referred to as (v, w)-path. In an undirected graph, a *walk* is an alternating sequence of vertices and edges $v_1, e_1, v_2, e_2, ..., v_{k-1}, e_{k-1}, v_k$, such that $e_i := \{v_i, v_{i+1}\}, i = 0, ..., k-1$. For a walk W we denote the set of vertices and the set of edges it contains by $V(W)$ and $E(W)$.

Let v and w be two distinct vertices of G. A subset $C \subseteq V \setminus \{v, w\}$ is called (v, w)-*separator*, or (v, w)-*node-separator*, if there is no path from v to w in the graph $(V \setminus C, E[V \setminus C])$. The family of all $(v, w)-$separators is denoted by $\mathcal{C}(v, w)$. Note that $\mathcal{C}(v, w) = \emptyset$ if and only if $\{v, w\} \in E$. For directed graphs we say that $C \subseteq V \setminus \{v, w\}$ is a $(v, w)-$separator if all directed paths from v to w contain a vertex from C.

Given edge costs $c : E \mapsto \mathbb{Q}_{\geqslant 0}$, the triplet (V, E, c) is referred to as *network*. By $d(v, w)$ we denote the cost of a shortest path (with respect to c) between vertices $v, w \in V$. We say that $d(v, w)$ is the *distance* between v and w. For any (distance) function $\tilde{d} : \binom{V}{2} \mapsto \mathbb{Q}_{\geqslant 0}$, and any $U \subseteq V$ we define the $\tilde{d}$-*distance graph* on U as the network

$$D_G(U, \tilde{d}) := (U, \binom{U}{2}, \tilde{c}), \tag{1.1}$$

with $\tilde{c}(\{v, w\}) := \tilde{d}(v, w)$ for all $v, w \in U$. If $\tilde{d}$ is the standard distance (i.e. $\tilde{d} = d$), we write $D_G(U)$ instead of $D_G(U, d)$. Note that we write usually $\tilde{d}(v, w)$ instead of $\tilde{d}(\{v, w\})$.

Finally, we denote by $\alpha(G)$ the maximum number of independent vertices in graph G.

1.1.3 Steiner arborescence problem

We will require the directed equivalent of the SPG, the *Steiner arborescence problem* (*SAP*), see e.g. Hwang et al. (1992), throughout his thesis. The SAP is defined as follows. Given a directed graph $D = (V, A)$, costs $c : A \to \mathbb{Q}_{\geqslant 0}$, a set $T \subseteq V$ of terminals, and a root $r \in T$, a directed tree $S \subseteq D$ is required such that: First, for all $t \in T$ the tree S contains exactly one directed path from r to t. Second, $c(A(S))$ is minimized.

We will use the following IP formulation for the SAP, due to Wong (1984), throughout this thesis. Associate with each arc $a \in A$ a binary variable $y(a)$, indicating whether a is contained in the Steiner arborescence ($y(a) = 1$) or not ($y(a) = 0$).

Formulation 1.1. *Directed Cut Formulation (DCut)*

$$
\begin{aligned}
&\mathit{min} \quad c^T y && && (1.2)\\
y(\delta^-(W)) &\geqslant 1 && \text{for all } W \subset V,\ r \notin W, W \cap T \neq \emptyset, && (1.3)\\
y(a) &\in \{0,1\} && \text{for all } a \in A. && (1.4)
\end{aligned}
$$

The constraints (1.3) make sure that all feasible solutions contain directed paths from the root to each additional terminal.

1.2 Experimental methodology

At several places, this thesis will analyze the practical performance of newly introduced algorithms by computational experiments. In the following, we describe some details concerning our computational methodology.

1.2.1 Hardware and software

All experiments for this thesis except for those from Chapter 6.3 were conducted on a cluster of Intel Xeon CPUs E3-1245 with 3.40 GHz and 32 GB RAM. With the exception of Chapter 6.3, all experiments were performed single-threaded. We ran only one job per compute node at a time, to avoid a distortion of the run time measures—originating for example from shared (L3) cache. We used a version of our Steiner tree solver SCIP-Jack that is embedded into a development version of SCIP 7.0.2 (Gamrath et al., 2020). We used the commercial CPLEX 12.10 (IBM, 2020), and the non-commercial SoPlex 5.0 (Gamrath et al., 2020) as LP solvers.

A previous version of SCIP-Jack is available as part of the SCIP Optimization Suite (Gleixner et al., 2018), which is free for academic use. A newer version of SCIP-Jack including the developments described in this thesis will be made publicly available as part of an upcoming release of the SCIP Optimization Suite. Most of the experiments in this thesis were performed with the same version of SCIP-Jack. Exceptions are marked as such, and include mostly exceptionally long runs. For reasons of reproducibility, the SCIP-Jack versions, as well as the log files of the experiments have been archived.

1.2.2 Averaging and performance variability

To evaluate and compare algorithmic performance on a large set of benchmark instances, we rely on comparing averages and maxima. The classic arithmetic average has the property to be strongly dominated by the largest absolute values. Since, the run times of state-of-the-art Steiner tree solvers usually vary widely even among single benchmark sets, this property seems disadvantageous. Thus, we usually use the *shifted geometric mean* (Achterberg, 2007b) instead, which is has become a standard measure in discrete optimization, see e.g. Mittelmann (2020). We note, that the use of the arithmetic mean would bias most results strongly in favor of SCIP-Jack, which is particularly strong on harder instances—across all problem classes considered in this thesis. Given values $t_1, ..., t_k \in \mathbb{R}_{\geqslant 0}$, and a *shift* $s \in \mathbb{R}_{\geqslant 0}$, the shifted geometric mean is defined as

$$\sqrt[k]{\prod_{i=1}^{k}(t_i + s)} - s. \tag{1.5}$$

Compared to the arithmetic average, the use of a geometric mean brings the benefit of reducing the influence of very hard instances. On the other hand, the use of a shift helps avoid an overrepresentation of very small values. In this thesis we use shifts of $s = 1$ or $s = 10$ (i.e., 1 or 10 seconds) for averaging run times, which are both standard values; see e.g. Gleixner et al. (2018); Mittelmann (2020). We also note that all run times reported for SCIP-Jack in this thesis include the reading time.

Finally, we note that mixed-integer solvers such as SCIP are usually subject to so-called *performance variability* (Lodi and Tramontani, 2013). Broadly speaking, performance variability means a large change of the solving behavior, such as run time, resulting from seemingly neutral changes to the solution process. Such seemingly neutral changes are for example the permutation of the rows and columns of the constraints matrix. Thus, it has become a common practice in the integer programming community to perform computational experiments with several *random seeds*, which modify the behavior of the solution process in a (hopefully) random way; see e.g. Lodi and Tramontani (2013). Similarly, permutations of the constraint matrix are employed. Also, statistical tests are often used. While we have implemented random seed features into SCIP-Jack, the impact on the solution process is far less pronounced than for general mixed-integer solvers. The impact on the run time on most benchmark sets used in this thesis is less than five percent—the impact of permuting the underlying graph is even smaller. A main reason for this small impact might be attributed to the fact that the vast majority of the instances is solved already at the root node of the branch-and-bound tree. Even more, many of the instances are already solved by the highly sophisticated presolving (or reduction) methods developed and implemented in the thesis. Reductions techniques for Steiner tree and related problems are far less susceptible to performance variability—both empirically, and theoretically. See for example Kingston and Sheppard (2003) for theoretical results concerning the robustness of reduction methods for SPG. Consequently, we do not use multiple random seeds or graph permutations for most of the experiments in this thesis.

Chapter 2

The prototype: Steiner tree problem in graphs

The first problem discussed in this thesis is naturally the classic Steiner tree problem in graphs (SPG). The results from this chapter also form a basis for the related problems discussed later on.

2.1 Introduction

This chapter starts with a more formal definition of the SPG: Given an undirected connected graph $G = (V, E)$, edge costs $c : E \to \mathbb{Q}_{>0}$ and a set $T \subseteq V$ of *terminals*, the problem is to find a tree $S \subseteq G$ with $T \subseteq V(S)$ such that $c(E(S))$ is minimized. A tree $S \subseteq G$ such that $T \subseteq V(S)$ is called *Steiner tree*. The vertices in $V \setminus T$ are referred to as *Steiner vertices* or *Steiner nodes*. Note that allowing non-negative or positive edge costs for the SPG is equivalent, since each zero-cost edge can simply be contracted. An illustration of an SPG instance and a corresponding Steiner tree is given in Figure 2.1. For simplicity, no edge costs are specified.

(a) An SPG instance

(b) A feasible solution (Steiner tree)

Figure 2.1: Illustration of a Steiner tree problem in a graph (left) and a possible solution (right). Terminals are drawn as squares, Steiner nodes as circles.

2.1.1 Background

Given the huge number of publications on Steiner tree problems[6], any overview is bound to be incomplete. Still, we try to at least touch upon some of the most important results in prominent research fields. Furthermore, we give a short historical background. As to practical applications of the (classic) SPG, we only note that prominent areas are VLSI design, see e.g. Held et al. (2011), Phylogency, see e.g. Hwang et al. (1992), and telecommunication networks, see e.g. Leitner et al. (2014). For further interesting applications, the reader is referred to Cheng and Du (2004); Noormohammadpour et al. (2017). For a more comprehensive background, the reader is referred to the books Hwang et al. (1992); Korte et al. (2018); Prömel and Steger (2002), and the recent survey Ljubic (2020). The Steiner tree problem is also the subject of several other books, e.g. Cheng and Du (2004); Cieslik (1998); Du et al. (2000); Voss (1990).

Historical notes

Historically, the Steiner problem in graphs is derived from a problem known as the *Euclidean Steiner tree problem*: Given a finite set T of points in the plane, connect them by line segments of minimum total Euclidean length such that any two points are interconnected by line segments either directly or by using intermediary points. More details on this problem are given in Section 5.4. The history of the Steiner tree problem can be traced back to the year 1636 when Pierre de Fermat formulated the special case of $|T| = 3$. Evangelista Torricelli found an elegant solution to this problem already before 1640 (Prömel and Steger, 2002).

More than 100 years later, the general Euclidean Steiner tree problem was independently formulated by Carl Friedrich Gauß and Joseph Diaz Gergonne (Brazil et al., 2014). The name *Steiner* is derived from Jakob Steiner (1796 - 1863), who held a chair of geometry at Berlin university. The, somewhat misleading, attribution of the problem name to Jakob Steiner is due to the famous treatise *What is Mathematics?* (Courant et al., 1941).

The Steiner tree problem in graphs as it is known today can be found in a publication by Hakimi (1971). Since then, hundreds of articles concerning the Steiner tree problem in graphs have been published; see Hwang et al. (1992) for a comprehensive, albeit outdated, survey, or Ljubic (2020) for an up-to-date one.

A detailed account of the rich history of the (Euclidean) Steiner tree problem can be found in Brazil et al. (2014).

Complexity results

The decision variant of the SPG is strongly $\mathcal{NP}$-complete, and the optimization variant thus $\mathcal{NP}$-hard. Indeed, the decision variant of the SPG is one of the famous 21 $\mathcal{NP}$-complete problems by Karp (1972). However, there are several polynomially solvable special cases of the SPG. The two most important ones are the case $|T| = 2$,

[6] As of November 2020, a search for the term *Steiner tree* produced 29 200 results on Google Scholar.

which corresponds to finding a shortest path between two vertices, and $|T| = n$, which corresponds to finding a minimum spanning tree in G. Further polynomially solvable special cases can be found in Hwang et al. (1992).

Dreyfus and Wagner (1971) and Levin (1971) described a dynamic programming algorithm for SPG that runs in

$$O(3^{|T|}n + 2^{|T|}n^2 + n^2 \log n + mn).$$

Hakimi (1971) also observed that the SPG is also fixed-parameter tractable (FPT) in the number of Steiner nodes—by using simple enumeration. Buchanan et al. (2018) showed that this algorithm by Hakimi (1971) is essentially best possible under the Strong Exponential-Time Hypothesis (SETH). The run time of the algorithm by Dreyfus and Wagner (1971) and Levin (1971) was later improved by Erickson et al. (1987). However, the leading exponential term was unchanged. Fuchs et al. (2007b) were the first to break the $O^\star(3^{|T|})$ bound. Subsequently, Fuchs et al. (2007a) improved the run time to $O^\star((2+\varepsilon)^{|T|})$ for any sufficiently small and fixed $\varepsilon > 0$. However, the (hidden) term depending on ε grows very quickly with decreasing ε. Vygen (2011) introduced an algorithm that runs in time

$$O(n|T|2^{|T|+\log_2 |T| \log_2 n}).$$

This algorithm is only outperformed by that of Fuchs et al. (2007a) if the ratio of terminals, i.e. $\frac{|T|}{n}$, is small. However, for sufficiently small terminal ratio, the algorithm by Erickson et al. (1987) is still the fastest known one. Additionally, for a terminal ratio greater than approximately $\frac{1}{2}$, the enumeration scheme from Hakimi (1971) is the fastest known solution method (for general SPG). Moreover, Marx et al. (2018) show that under the Exponential-Time Hypothesis (ETH), SPG cannot be solved in $2^{o(|T|)}n^{O(1)}$, even for planar graphs with unit weights. Several articles have also exploited the fact that the SPG is FPT with respect to the tree-width of the underlying graph, see e.g. Chimani et al. (2012).

A different line of research has focused on polynomial space algorithms for SPG, see e.g. Fomin et al. (2013); Kisfaludi-Bak et al. (2020a). A polynomial space algorithm that is faster than any of the algorithms mentioned in this section so far was given by Nederlof (2009). However, this algorithm only works for a restricted set of (integer) edge weights. See also Fomin et al. (2019a) for a recent related result.

Similarly, many articles consider exact SPG algorithms only for planar graphs. For recent results see Kisfaludi-Bak et al. (2020b). Dom et al. (2014) showed that the SPG parameterized by the number of terminals does not admit a polynomial kernel (under certain general complexity assumptions widely believed to be true). A linear programming based fixed-parameter tractable algorithm for the SPG is described in Siebert et al. (2020a)

Approximation algorithms

Bern and Plassmann (1989) demonstrated that the SPG is MAXSNP-hard, even for edge weights in $\{1, 2\}$. I.e., there exists an $\varepsilon > 0$ such that finding a $(1+\varepsilon)$-approximation to this problem is $\mathcal{NP}$-hard. Chlebík and Chlebíková (2008) further showed that approximating the SPG within a factor of $\frac{96}{95}$ is $\mathcal{NP}$-hard. However, the SPG in planar graphs has an approximation scheme, which was found by Borradaile et al. (2009).

A 2-approximation of the SPG was already found by Gilbert and Pollak (1968). The idea is to compute a minimum spanning tree in the subgraph of the metric closure of G induced by T. This and other 2-approximation algorithms were also published by several other authors, El-Arbi (1978); Kou et al. (1981); Takahashi and Matsuyama (1980). An $O(m+n\log n)$ algorithm for computing the minimum-spanning-tree-based 2-approximation of the SPG is suggested in Mehlhorn (1988); see also Kou (1990).

Zelikovsky (1993) was the first to break the 2-approximability bound for SPG. He introduced a, much acclaimed, $\frac{11}{6}$-approximation algorithm. A faster realization of the algorithm is given by Duin and Voss (1997). The ratio has subsequently been improved to 1.75 by Berman and Ramaiyer (1994), to 1.65 by Karpinski and Zelikovsky (1997), to 1.60 by Hougardy and Prömel (1999), and to 1.55 by Robins and Zelikovsky (2005). Finally, the currently best approximation ratio of 1.39 was given by Byrka et al. (2013), who used an LP-based approach with randomized iterative rounding. This approach uses a hypergraphic IP formulation of the SPG that was introduced by Polzin and Daneshmand (2003). In Goemans et al. (2012) a faster and de-randomized version of the 1.39-approximation algorithm by Byrka et al. (2013) is given.

Recently, a new research area has emerged that works on fixed-parameter tractable approximation algorithms; so essentially a combination of the current and the previous section. Results for SPG in this area can be found in the survey Feldmann et al. (2020).

Finally, despite the considerable theoretical advancements sketched above, the empirical results of known approximation algorithms are clearly inferior to those of both heuristic and integer-programming-based methods, see e.g. Beyer and Chimani (2019); Ciebiera et al. (2014). The approximation algorithms fall short both in terms of run time and solution quality.

Exact algorithms

While the algorithm introduced by Dreyfus and Wagner (1971) can in principle solve any SPG, it is hopelessly slow and also too memory intensive for practical purposes. Consequently, several authors, e.g. Shore et al. (1982); Beasley (1984), suggested more practical algorithms in the following years. A comprehensive overview of exact algorithms up to 1990 is given in Hwang and Richards (1992). A later milestone was the exact algorithm by Duin (1993), incorporating his work on reductions (Duin and Volgenant, 1989a,b) and heuristics (Duin and Voss, 1997). This exact algorithm used branch-and-bound, but only a dual heuristic instead of linear programming. Computational experiments by Lucena and Beasley (1998) show that the implementation by Duin (1993) is more than three orders of magnitude faster than any of the solvers by Chopra et al. (1992); Beasley (1989); Lucena and Beasley (1998), which were the best alternatives at this time. Five years later, Koch and Martin (1998) introduced a branch-and-cut algorithm with sophisticated separation procedures, combined with reduction techniques and primal heuristics. Koch and Martin (1998) were able to solve all problem instances that had hitherto been discussed in the literature to optimality. Subsequently, de Aragão et al. (2001) developed a branch-and-bound algorithm, based on dual heuristics, that could solve several large-scale, VLSI instances which had been newly introduced by Koch and Martin (1998) for the first time to optimality. A key ingredient of this branch-and-bound algorithm were the reduction methods introduced in Uchoa et al. (2002).

A huge leap was made by the joint PhD theses of Polzin (2003) and Vahdati Daneshmand (2004), whose work was also published in a series of papers (Polzin and Daneshmand, 2001a, 2002, 2001b, 2003, 2006). The authors combined known methods with many new algorithms within a branch-and-bound framework. These new algorithms include various sophisticated reduction techniques, a dynamic programming algorithm, a (provably) stronger integer programming formulation, and primal and dual heuristics. Furthermore, a prominent feature of Polzin (2003); Vahdati Daneshmand (2004) is the efficient implementation of their algorithms. The resulting solver drastically outperformed any competitor—on many benchmark instances even by three or more orders of magnitude. Furthermore, it could solve many instances for the first time to optimality. In fact, the solver has remained the state of the art until today. See Polzin and Vahdati-Daneshmand (2014) for more recent computational results of their solver, and Polzin and Vahdati-Daneshmand (2009) for an overview of the algorithmic components.

In 2014, the 11th DIMACS Challenge, dedicated to Steiner tree problems, took place. In the wake of the Challenge, several new SPG solvers were introduced in the literature. Fischetti et al. (2017) introduced a branch-and-cut solver, including reduction techniques, and a variety of heuristics. The solver is especially tailored towards previously unsolvable instances, and instances with unit edge weights. Notably, the solver won the exact SPG category at the DIMACS Challenge. Pajor et al. (2017) introduced a branch-and-bound solver based on the same dual heuristic already used in Duin (1993)—albeit Pajor et al. (2017) introduced a more efficient implementation. Furthermore, the solver includes extensions and more efficient implementations of primal heuristics from Ribeiro et al. (2001) and Uchoa and Werneck (2010). The solver by Pajor et al. (2017) won the heuristic SPG category at the DIMACS Challenge. Hougardy et al. (2017) provided an extension of the algorithm by Dreyfus and Wagner (1971), which is far more efficient in practice, but retains the worst-case bound. However, their solver works only for instances with no more than 64 terminals.

Overall, the 11th DIMACS Challenge brought considerable progress on the solution of notoriously hard SPG instances that had been designed to defy known solution techniques, see Koch et al. (2001); Rosseti et al. (2004). Several of these instances could be solved for the first time to optimality. In particular, Fischetti et al. (2017) and Pajor et al. (2017) showed strong results for some of these instances. Additionally, Gamrath et al. (2017) could solve several instances by running a branch-and-bound search in parallel on a supercomputer. However, on the vast majority of (real-world) instances from the literature, Polzin (2003); Vahdati Daneshmand (2004) (whose solver did not compete at the DIMACS Challenge) stayed well out of reach: For many benchmark instances, their solver is more than two orders of magnitude faster, and it can furthermore solve far more instances to optimality.

In 2018, the 3rd PACE Challenge (Bonnet and Sikora, 2019) took place, dedicated to fixed-parameter tractable algorithms for SPG. Thus, the PACE Challenge considered mostly instances with a small number of terminals, or with small tree-width. Solvers that participated in the PACE Challenge are for example described in Fichte et al. (2020); Hušek et al. (2020); Iwata and Shigemura (2019). Notably, the solver by Iwata and Shigemura (2019) has the worst-case complexity of Erickson et al. (1987), but is faster than Hougardy et al. (2017) in practice. The solver can also handle instances with more than 64 terminals. Furthermore, a forerunner of the solver presented in this thesis competed at the PACE Challenge, see also Section 2.7.2.

While many different techniques have been employed for the exact solution of SPG in the literature in the last 50 years, widely used ingredients are as follows: Reduction techniques for both preprocessing and propagation, Lagrangian or dual-ascent relaxations for calculating strong lower bounds, various heuristics for calculating primal bounds, and branch-and-cut or branch-and-price methods based on MIP formulations for proving optimality.

2.1.2 Contribution and structure

This chapter aims to once again advance the state of the art in exact SPG solution. To this end, we proceed as follows.

- Section 2.2 provides a theoretical basis for the integer-programming-based exact solution approach of this chapter. We analyze (mixed) integer programming formulations of the SPG that are widely used in theory and practice. Several new results are given. In particular, we provide new, and stronger, conditions under which the LP-relaxation of the well-known (bi-)directed cut formulation has no integrality gap.

- Section 2.3 is based on a combination of three concepts: Implications, conflicts, and reductions. As a result, various new SPG techniques are conceived. By using a new implication concept, a distance function is conceived that provably dominates the well-known bottleneck Steiner distance. As a result, several reduction techniques that are stronger than results from the literature can be designed. We show how to derive conflict information between edges from the above methods. Further, we introduce a new reduction operation whose main purpose is to introduce additional conflicts.

- Section 2.4 introduces a more general version of the powerful so-called extended reduction techniques. We furthermore introduce stronger reduction criteria, and make use of both the previously introduced new distance concept and the conflict information.

- Section 2.5 reviews and introduces several primal SPG heuristics—intended to accelerate exact SPG solution. The new heuristics integrate some of the previously introduced implication and reduction techniques.

- Section 2.6 integrates the previously introduced algorithmic components into an exact branch-and-cut algorithm. We also discuss further components such as separation methods, decomposition, domain propagation, or branching. Furthermore, we introduce an exact dynamic programming algorithm (based heavily on reduction methods), which is employed for solving decomposed subproblems.

– Section 2.7 provides computational results. Besides showing the impact of individual algorithmic components, we provide comparisons with the state of the art on a large collection of well-established benchmark sets from the literature.

The resulting exact SPG solver consistently outperforms the current state-of-the-art solver from Polzin (2003); Vahdati Daneshmand (2004)—both with respect to the run time and the number of solved instances. Furthermore, we can solve several SPG benchmark instances for the first time to optimality.

2.2 Integer programming formulations

Many (mixed) integer programming formulations for the SPG have been described in the literature, see e.g. Goemans and Myung (1993); Magnanti and Wolsey (1995); Polzin and Daneshmand (2001a) for overviews. This section gives new theoretical results for some well-known IP formulations for SPG, which are also used in state-of-the-art SPG solvers. Indeed, also the SPG solver developed as part of this thesis is based on one of the IP formulations discussed in the following.

We are mainly concerned with the strength of the respective LP-relaxation, which is of crucial importance for the practical success of an IP or MIP formulation—not only for SPG, but also for many other optimization problems, such as the TSP (Applegate et al., 2006). The strength of LP-relaxations for SPG has thus been widely discussed in the literature. Furthermore, LP-relaxations play a crucial role in the best-known SPG approximation algorithm, see Byrka et al. (2013). In this section, we assume that the edge costs c are real-valued and positive. The reader is reminded of the definitions and concepts introduced in Chapter 1.

2.2.1 Cut and flow formulations

A natural way to formulate the SPG as an integer program is by associating with each edge $e \in E$ a binary variable $x(e)$, indicating whether e is contained in the Steiner tree ($x(e) = 1$) or not ($x(e) = 0$). This conception paves the way for a cut-based formulation introduced in Aneja (1980):

Formulation 2.1. *Undirected Cut Formulation (UCut)*

$$\min c^T x \tag{2.1}$$

$$x(\delta(W)) \geqslant 1 \quad \textit{for all } W \subset V,\ 0 < |W \cap T| < |T|, \tag{2.2}$$

$$x(e) \in \{0,1\} \quad \textit{for all } e \in E. \tag{2.3}$$

One verifies that the constraints (2.2) ensure the existence of paths from each terminal to all other ones in a feasible solution. In this way, it can be readily demonstrated that *UCut* is correct. We note that a feasible but not optimal solution to

$UCut$ is not necessarily the incidence vector of a Steiner tree. Indeed, the convex hull of all $x \in \mathbb{N}_0^E$ that satisfy (2.2) is of blocking type, i.e. its recession cone equals $\mathbb{R}^E_{\geqslant 0}$.

It is well-known that any SPG can be transformed to an SAP by replacing each edge by two anti-parallel arcs of the same cost, and distinguishing an arbitrary terminal as the root. This procedure results in a one-to-one correspondence between the respective solution sets. The SPG IP formulation that consists of Formulation 1.1 applied to this SAP is called *bidirected cut formulation* ($BDCut$). This formulation is widely known, and often used in Steiner tree solvers, see e.g. Fischetti et al. (2017); Polzin and Daneshmand (2001b).

The relation between the directed and undirected formulation has been widely discussed in the literature, see e.g. Chopra and Rao (1994); Magnanti and Wolsey (1995). We briefly state the most important results here:

- $v_{LP}(UCut) \leqslant v_{LP}(BDCut)$, and $\sup\left\{\frac{v_{LP}(BDCut)}{v_{LP}(UCut)}\right\} = 2$ (Duin, 1993).
- The value $v_{LP}(BDCut)$ is independent of the choice of the root in the transformed SAP, as shown by Goemans and Myung (1993). See also Corollary 4.28 for a shorter proof.

The undirected formulation can be tightened by the *Steiner partition inequalities* introduced in Grötschel and Monma (1990) and the *odd hole inequalities* by Chopra and Rao (1994), but still $BDCut$ remains strictly stronger than $UCut$. However, by adding auxiliary variables, the undirected cut formulation can be made as tight as the bidirected one (Goemans and Myung, 1993). Still, the latter is computationally more attractive, as violated inequalities can be quickly separated by using maximum-flow algorithms; see Section 6.2.4 for more details. Additionally, $BDCut$ can readily be tightened by adding additional constrains, as we will see below.

Another well-known formulation, see e.g. Wong (1984), is based on flows.

Formulation 2.2. *Directed Multicommodity Flow Formulation (DF)*

$$\min \quad c^T y \tag{2.4}$$

$$\text{s.t. } f^t(\delta^-(v)) - f^t(\delta^+(v)) = \begin{cases} 1 & \text{if } v = t; \\ 0 & \text{if } v \in V \setminus \{r, t\} \end{cases} \quad \text{for all } v \in V, t \in T \setminus \{r\}, \tag{2.5}$$

$$f^t \leqslant y \qquad \text{for all } t \in T \setminus \{r\}, \tag{2.6}$$

$$f^t \geqslant 0 \qquad \text{for all } t \in T \setminus \{r\}, \tag{2.7}$$

$$y \in \{0,1\}^A. \tag{2.8}$$

By using the max-flow/min-cut theorem, one shows that DF is an extended formulation of $DCut$, i.e., $proj_y(\mathcal{P}_{LP}(DF)) = \mathcal{P}_{LP}(DCut)$, see e.g. Duin (1993). Both formulations can be strengthened by the so-called *flow-balance constraints* from Duin (1993); Koch and Martin (1998):

$$y(\delta^-(v)) \leqslant y(\delta^+(v)) \qquad \text{for all } v \in V \setminus T. \tag{2.9}$$

We will refer to the extensions of the above formulations that additionally include (2.9) as $DCut_{FB}$ and DF_{FB}, respectively. The flow-balance constraints are commonly used in SPG solvers, e.g. Koch and Martin (1998); Polzin and Daneshmand (2002), and have moreover been applied to several related problem, see e.g. Ljubic et al. (2006); Leitner et al. (2018b).

Although the flow-balance constraints are widely used, the only theoretical results for SPG or SAP in the literature that the author is aware of are given by Duin (1993) and Polzin and Daneshmand (2001a), who provide examples where $v_{LP}(DCut_{FB}) > v_{LP}(DCut)$. For the two-stage stochastic SPG, Leitner et al. (2018b) give a corresponding result. Next, we give a stronger (new) result, which will be used several times in this thesis.

Lemma 2.3. *If $|T| \leqslant 3$, then $v_{LP}(DCut_{FB}) = v(DCut_{FB})$.*

Proof. For the case of $|T| = 1$ and $|T| = 2$ the lemma holds already without the flow-balance constraints. So let (V, A, T, c, r) be an SAP with two terminals t, u besides the root r. We additionally require that a feasible solution does not have any leaves apart from r, t, u. For this so-called two-terminal Steiner tree problem a complete polyhedral description is given by Ball et al. (1989):

$$f(\delta^-(v)) - f(\delta^+(v)) \geqslant \begin{cases} -1 & \text{if } v = r; \\ 0 & \text{otherwise;} \end{cases} \quad \text{for all } v \in V, \tag{2.10}$$

$$(f + f^t)(\delta^-(v)) - (f + f^t)(\delta^+(v)) = \begin{cases} 1 & \text{if } v = t; \\ -1 & \text{if } v = r; \\ 0 & \text{otherwise;} \end{cases} \quad \text{for all } v \in V, \tag{2.11}$$

$$(f + f^u)(\delta^-(v)) - (f + f^u)(\delta^+(v)) = \begin{cases} 1 & \text{if } v = u; \\ -1 & \text{if } v = r; \\ 0 & \text{otherwise;} \end{cases} \quad \text{for all } v \in V, \tag{2.12}$$

$$f + f^t + f^u \leqslant y, \tag{2.13}$$

$$y, f, f^t, f^u \in \mathbb{R}^A_{\geqslant 0}. \tag{2.14}$$

The above description is based on the following observation: Any feasible arborescence for the two-terminal Steiner tree problem consists of a path from r to a splitter node v, as well as a v-t and a v-u path. Note that any of these paths can be a single node.

Let (f^t, f^u, y) be an optimal LP solution to DF_{FB}. Assume that this solution is minimal, i.e. for any feasible solution $(\tilde{f}^t, \tilde{f}^u, \tilde{y}) \leqslant (f^t, f^u, y)$ it holds that $(\tilde{f}^t, \tilde{f}^u, \tilde{y}) = (f^t, f^u, y)$. We will show that there exist $\hat{f}, \hat{f}^t, \hat{f}^u \in \mathbb{R}^A$ such that $(\hat{f}, \hat{f}^t, \hat{f}^u, y)$ is contained in the polyhedron described above. Define for all $a \in A$:

$$\hat{f}(a) := \min\{f^t(a), f^u(a)\}, \tag{2.15}$$

$$\hat{f}^t(a) := \max\{f^t(a) - f^u(a), 0\}, \tag{2.16}$$

$$\hat{f}^u(a) := \max\{f^u(a) - f^t(a), 0\}. \tag{2.17}$$

First, we show (2.10). Let $v \in V$. Because of the assumed minimality of (f^t, f^u, y) we obtain:

$$\hat{f}(\delta^-(v)) - \hat{f}(\delta^+(v)) = (f^t + f^u - y)(\delta^-(v)) - (f^t + f^u - y)(\delta^+(v)) \tag{2.18}$$
$$= (f^t + f^u)(\delta^-(v)) - (f^t + f^u)(\delta^+(v)) + y(\delta^+(v)) - y(\delta^-(v)). \tag{2.19}$$

If $v = r$, then

$$(f^t + f^u)(\delta^-(v)) - (f^t + f^u)(\delta^+(v)) = -2 \tag{2.20}$$

and

$$y(\delta^+(v)) - y(\delta^-(v)) \geqslant 1, \tag{2.21}$$

thus (2.19) implies that (2.10) holds. If $v \in \{t, u\}$, then

$$(f^t + f^u)(\delta^-(v)) - (f^t + f^u)(\delta^+(v)) = 1 \tag{2.22}$$

and

$$y(\delta^+(v)) - y(\delta^-(v)) \geqslant -1. \tag{2.23}$$

Finally, if $v \in V \setminus \{r, t, u\}$, the flow-balance constraints imply that (2.19) is non-negative.

Next, consider (2.11)—and equivalently (2.12). By definition it holds that

$$(\hat{f} + \hat{f}^t)(\delta^-(v)) - (\hat{f} + \hat{f}^t)(\delta^+(v)) = f^t(\delta^-(v)) - f^t(\delta^+(v)), \tag{2.24}$$

which implies (2.11). Likewise, (2.13) follows from the definition of $\hat{f}$, $\hat{f}^t$, and $\hat{f}^u$. □

Note that the lemma is best possible in the sense that there exist SAP instances with $|T| = 4$ such that $v_{LP}(DCut_{FB}) \neq v(DCut_{FB})$, see e.g. Liu (1990); Polzin and Daneshmand (2001a).

We close with a new result for SPG, which is a direct consequence of Lemma 2.3.

Theorem 2.4. *If $|T| \leqslant 3$, then $v_{LP}(BDCut_{FB}) = v(BDCut_{FB})$.*

As for the previous lemma, one can readily show that the theorem is best possible, see e.g. Polzin and Daneshmand (2001a) for an SPG instance with $|T| = 4$ such that $v_{LP}(BDCut_{FB}) \neq v(BDCut_{FB})$.

2.2.2 Formulations for unweighted Steiner tree problems

Given an undirected connected graph $G = (V, E)$ and a set $T \subseteq V$ of *terminals*, the *unweighted Steiner tree problem in graphs* (USPG) is to find a tree $S \subseteq G$ with $T \subseteq V(S)$ such that $|E(S)|$ is minimized. The USPG can also be seen as a Steiner tree problem with uniform edge weights. Many of the hardest Steiner tree benchmark instances are unweighted, see Koch et al. (2001) for an overview. Moreover, many theoretical articles consider just the unweighted case, see e.g. Nederlof (2013) for complexity results.

This section analyzes and compares two formulations for the USPG. First, we analyze the *BDCut* formulation in the context of USPG. Second, we analyze the node-separator formulation from Fischetti et al. (2017), which can only be applied to problems without (or with uniform) edges weights.

Initially, we introduce the node-based USPG formulation by Fischetti et al. (2017). This formulation associates with each node $v \in V$ a binary variable $x(v)$, which indicates whether v is contained in a Steiner tree ($x(v) = 1$) or not ($x(v) = 0$). Connectivity is modeled by using node-separators (see Section 1.1.2).

Formulation 2.5. *Terminal Node Separator Formulation (TNCut)*

$$
\begin{aligned}
&min \quad x(V) - 1 && && (2.25)\\
&\quad s.t.\ x(C) \geqslant 1 && \textit{for all } t,u \in T, t \neq u, C \in \mathcal{C}(t,u), && (2.26)\\
&\qquad\quad x(v) = 1 && \textit{for all } v \in T, && (2.27)\\
&\qquad\quad x(v) \in \{0,1\} && \textit{for all } v \in V. && (2.28)
\end{aligned}
$$

Note that in Fischetti et al. (2017) a more general version of TNCut for the prize-collecting USPG is used. However, the prize-collecting USPG is essentially a maximum-weight connected subgraph problem, see Chapter 3. The results of this section can be partly extended to this more general variant (which is done in Section 3.2.2 for the non-rooted case), but for simplicity, we now consider the USPG only.

Exactness of the bidirected cut formulation

This section formulates conditions under which the bidirected cut formulation has no integrality gap. A simple reduction technique for USPG is to contract adjacent terminals (and delete one edge from each resulting pair of multi-edges). The following proposition shows that the absolute integrality gap of *BDCut* is invariant under this operation.

Proposition 2.6. *Let I be an USPG instance with adjacent terminals t, u. Let I' be the USPG obtained from contracting t and u. It holds that:*

$$v_{LP}(BDCut(I)) = v_{LP}(BDCut(I')) + 1. \tag{2.29}$$

Proof. Throughout the proof we assume that u is the root for the *BDCut* formulation, i.e. $r = u$. It is well-known that the choice of the root does not affect $v_{LP}(BDCut)$ (this result also follows from the proof of Theorem 2.7). Furthermore, let $D' = (V', A')$ be the bidirected graph obtained by contracting r and t and let r' be the new vertex. I.e., $V' = (V \setminus \{r, t\}) \cup \{r'\}$.

First, we show that $v_{LP}(BDCut(I)) \geqslant v_{LP}(BDCut(I')) + 1$. Let y be an optimal LP solution to $BDCut(I)$. The optimality of y implies that $y(\delta^-(t)) = 1$, see Polzin and Daneshmand (2001a). Create a new optimal solution $\tilde{y}$ as follows. Set $\tilde{y}(a) := y(a)$ for all $a \in A \setminus \delta^-(t)$, $\tilde{y}(a) := 0$ for all $a \in \delta^-(t) \setminus \{(r, t)\}$, and $\tilde{y}((r, t)) := 1$. Note that for any cut $\delta^-(U)$ with $U \subset V \setminus \{r\}$ such that $\delta^-(U) \cap \delta^-(t) \neq \emptyset$ it holds that $(r,t) \in \delta^-(U)$. Thus, $\tilde{y}(\delta^-(U)) \geqslant 1$. Consequently, $\tilde{y}$ satisfies (1.3). Define an LP solution y' to $BDCut(I')$ as follows: $y'(a) := \tilde{y}(a)$ for all $a \in A' \cap A$, and $y'(a) := 0$ for all $a \in \delta^-_{D'}(r')$. For any $a = (r',v) \in \delta^+_{D'}(r')$ proceed as follows. If $(r,v),(t,v) \in A$, set $y'(a) := \tilde{y}((r,v)) + \tilde{y}((t,v))$; if $(t,v) \notin A$, set $y'(a) := \tilde{y}((r,v))$; otherwise, set $y'(a) := \tilde{y}((t,v))$. Because of $y(\delta^-(v)) \leqslant 1$, we have in any case that $y'(a) \leqslant 1$.

It remains to show that $v_{LP}(BDCut(I)) \leqslant v_{LP}(BDCut(I'))+1$. Given an optimal LP solution y' to $BDCut(I')$ we define a corresponding LP solution y to $BDCut(I)$. First, $y((r,t)) := 1$, $y((t,r)) := 0$. Second, $y(a) := y'(a)$ for all $a \in A' \cap A$, and $y(a) := 0$ for all $a \in \delta^-(\{r,t\})$. Next, consider the remaining edges $\delta^+(\{r,t\})$. If $(r,v),(t,v) \in A$ set $y((r,v)) := y'(r',v)$, $y((t,v)) := 0$; otherwise, for $a = (r,v)$ or $a = (t,v)$ set $y(a) = y'((r',v))$. □

With this result at hand, we obtain the following theorem (recall that $\alpha(G)$ denotes the independence number of graph G).

Theorem 2.7. *Consider an USPG on a graph G. If $\alpha(G) \leqslant 3$, then $v_{LP}(BDCut) = v(BDCut)$.*

Proof. Consider a USPG instance $I = (G,T,c)$ with $\alpha(G) \leqslant 3$. Let $I' = (G',T',c')$ be the USPG obtained by (repeatedly) contracting all adjacent terminals. Let $D' = (V',A')$ be the bidirected equivalent of G'. Proposition 2.6 implies the following: $v_{LP}(BDCut(I)) = v(BDCut(I))$ if and only if $v_{LP}(BDCut(I')) = v(BDCut(I'))$. Furthermore, because of $\alpha(G) \leqslant 3$ it holds that $|T'| \leqslant 3$. For $|T'| < 3$, the $BDCut$ formulation is well-known to have no integrality gap. So assume $|T'| = 3$. By construction of I', the terminals form an independent set. Further, let y be an optimal LP solution to $BDCut(I')$ with an arbitrary $r \in T'$ being the root.

Suppose that $v_{LP}(BDCut(I')) \neq v(BDCut(I'))$. By Lemma 2.3, there is a $v \in V' \setminus T'$ such that

$$y(\delta^+(v)) < y(\delta^-(v)). \tag{2.30}$$

Because of $\alpha(G) \leqslant 3$, at least one of the terminals needs to be adjacent to v. We may assume that this property holds for r. Otherwise, we can readily create another optimal LP solution $\tilde{y}$ that satisfies (2.30) and has a root adjacent to v: Assume that a $t \in T \setminus \{r\}$ is adjacent to v and let f^t be a unit flow from r to t such that $f^t \leqslant y$; define $\tilde{y}((q,u)) := y((q,u)) - f^t((q,u)) + f^t((u,q))$ for all $(u,q) \in A'$.

Define a new LP solution y' from y as follows. For $a_0 := (r,v)$ set $y'(a_0) := y(\delta^+(v))$. For any $a \in \delta^-(v) \setminus \{a_0\}$ set $y'(a) := 0$. For all (remaining) $a \in A' \setminus \delta^-(v)$ set $y'(a) := y(a)$. Note that because of (2.30) it holds that $y'(A') < y(A')$. It remains to be shown that y' is feasible. Suppose that there is a $U \subseteq V' \setminus \{r\}$ with $U \cap T' \neq \emptyset$ and $y'(\delta^-(U)) < 1$. Because y is feasible, it has to hold that $v \in U$. Let $\tilde{U} := U \setminus \{v\}$.

By the construction of y' it holds that

$$\begin{aligned} y(\delta^-(\tilde{U})) &= y'(\delta^-(\tilde{U})) \\ &= y'(\delta^-(\tilde{U})) + y'((r,v)) - y'(\delta^+(v)) \\ &\leqslant y'(\delta^-(U)) \\ &< 1, \end{aligned}$$

which contradicts the feasibility of y. Consequently, we have shown that

$$v_{LP}(BDCut(I')) = v(BDCut(I'))$$

and, thus, $v_{LP}(BDCut(I)) = v(BDCut(I))$. □

The theorem is best possible; i.e., there exist USPG instances such that $\alpha(G) = 4$ and $v_{LP}(BDCut) \neq v(BDCut)$, see e.g. Duin (1993); Filipecki and Van Vyve (2020).

Comparison of edge and node based formulation

Formulation 2.5 ($TNCut$) was used within a branch-and-cut algorithm by the most successful solver (Fischetti et al., 2017) at the 11th DIMACS Challenge (DIMACS, 2015). Furthermore, this solver was able to solve several USPG benchmark instances that had been unsolved for more than a decade to optimality. Thus, one might wonder how this formulation theoretically compares with the better-known bidirected cut formulation. As the next proposition shows, $BDCut$ is always stronger than $TNCut$ and the relative gap can be rather large.

Proposition 2.8. *It holds that $v_{LP}(TNCut) \leqslant v_{LP}(BDCut)$. Furthermore,*

$$\sup \left\{ \frac{v_{LP}(BDCut)}{v_{LP}(TNCut)} \right\} \geqslant 2, \tag{2.31}$$

where the supremum is taken over all USPG instances.

Proof. For the first inequality consider an optimal LP solution y to $BDCut$. Define $x \in \mathbb{R}^V$ by $x(v) := y(\delta^-(v))$ for all $v \in V \setminus \{r\}$ and $x(r) := 1$. The optimality of y implies $x(v) \leqslant 1$ for all v, see Polzin and Daneshmand (2001a). Let $t, u \in T$ with $t \neq u$ and $C_{tu} \in \mathcal{C}(t,u)$. We will show that C_{tu} satisfies (2.26). If $C_{tu} \cap T \neq \emptyset$, then $x(C_{tu}) \geqslant 1$, because $x(q) \geqslant 1$ for all $q \in T$ due to (1.3) and the definition of x. Thus, (2.26) holds. If $C_{tu} \cap T = \emptyset$, let U_r be the connected component in the graph induced by $V \setminus C_{tu}$ with $r \in U_r$. By definition of C_{tu}, either $t \notin U_r$ or $u \notin U_r$. Therefore, $y(\delta^+(U_r)) \geqslant 1$, which implies $y(\delta^-(C_{tu})) \geqslant 1$ because of $\delta^+(U_r) \subset \delta^-(C_{tu})$. Now we obtain from the definition of x that

$$x(C_{tu}) \geqslant y(\delta^-(C_{tu})) \geqslant 1.$$

Finally, by construction of x we have that

$$x(V) - 1 = \sum_{v \in V} y(\delta^-(v)) = y(A);$$

note that $y(\delta^-(r)) = 0$ because y is optimal.

For (2.31) we construct the following family of USPG instances. For any $k \geqslant 3$ let I_k be the USPG instance with with $k+k^2$ nodes, $k+k^2$ edges, and k terminals defined as follows. Let t_i for $i = 1, ...k$ be the terminals and define for each $i \in \{1, ..., k\}$ Steiner nodes $v_{i,j}$, $j = 1, ...k$. For each $i \in \{1, ..., k\}$ define edges $\{t_i, v_{i,1}\}$, $\{t_{(i+1) \bmod k}, v_{i,k}\}$, and $\{v_{i,j}, v_{i,j+1}\}$ for $j = 1, ..., k-1$. Instance I_3 is shown in Figure 2.2. A feasible (and indeed optimal) LP solution x to $TNCut(I_k)$ is given by $x(t) := 1$ for all terminals t and $x(v) := 0.5$ for any Steiner node v. Its objective is $\frac{k^2}{2} + k - 1$. On the other hand, $v_{LP}(BDCut(I_k)) = k + k(k-1) = k^2$. Thus,

$$\lim_{k\to\infty} \frac{v_{LP}(DCut(I_k))}{v_{LP}(TNCut(I_k))} = \lim_{k\to\infty} \frac{k^2}{\frac{k^2}{2} + k - 1} = 2, \tag{2.32}$$

which concludes the proof. □

Corollary 2.9. *The (relative) integrality gap of $TNCut$ is at least 2.*

Note that one can strengthen $TNCut$ by constraints that correspond to the flow-balance constraints for $BDCut$, see Fischetti et al. (2017). However, if compared to $BDCut_{FB}$, the results of Proposition 2.8 remain the same for this stronger version of $TNCut$. As to the practical performance of $TNCut$, we note that the SPG solver developed in this thesis, which is based on $BDCut_{FB}$, also solves more of the aforementioned notoriously hard, unweighted benchmark instances than the solver of Fischetti et al. (2017) (which uses $TNCut$). See Section 2.7 for the computational results.

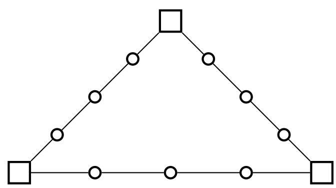


Figure 2.2: USPG instance I_3. Terminals are drawn as squares.

2.3 Implications, conflicts, and reductions

Informally, reduction methods transform a given instance to another, *reduced*, one, such that any optimal solution to the reduced instance can be re-transformed to an optimal solution to the original instance. In Section 2.3.4 we give a more formal definition. Reduction techniques have been a key ingredient in exact SPG solvers, see e.g. Duin (1993); Koch and Martin (1998); Uchoa et al. (2002); Polzin and Daneshmand (2001b). However, reduction techniques are also useful to improve the performance of heuristics, or even approximation algorithms (Beyer and Chimani, 2015).

This section reviews key results from the literature and introduces several new techniques that are (provably) stronger than previously known methods. A vital ingredient in several of the new techniques is the integration of new implication and conflict concepts. Later on we will see that these concepts are also useful beyond reduction methods.

2.3.1 Bottleneck distances and implications

Among the various SPG reduction techniques from the literature, the *bottleneck Steiner distance* introduced in Duin and Volgenant (1989a) is arguably the most important one, being the backbone of several powerful reduction methods. This section introduces a provably stronger distance concept, and discusses several applications for improved reduction methods.

The bottleneck Steiner distance

Let P be a simple path with at least one edge. The *bottleneck length* (Duin and Volgenant, 1989a) of P is

$$bl(P) := \max_{e \in E(P)} c(e). \tag{2.33}$$

Let $v, w \in V$. Let $\mathcal{P}(v, w)$ be the set of all simple paths between v and w. The *bottleneck distance* (Duin and Volgenant, 1989a) between v and w is defined as

$$b(v, w) := \inf\{bl(P) \mid P \in \mathcal{P}(v, w)\}, \tag{2.34}$$

with the common convention that $\inf \emptyset = \infty$. Note that $b(v, w)$ is equal to the bottleneck length of the path between v and w on any minimum spanning tree of (G, c), as observed in Dreyfus and Wagner (1971).

Now consider the distance graph $D := D_G(T \cup \{v, w\})$. Let b_D be the bottleneck distance in D. Define the *bottleneck Steiner distance* or *special distance* (Duin and Volgenant, 1989a) between v and w as

$$s(v, w) := b_D(v, w). \tag{2.35}$$

One also finds alternative, path-based definitions of the bottleneck Steiner distance in the literature, but these are weaker than the above definition. The bottleneck Steiner distance is arguably the most important reduction concept for SPG, with various applications. The arguably best-known one is the following criterion, which allows for edge deletion (Duin and Volgenant, 1989a).

Theorem 2.10. *Let $e = \{v, w\} \in E$. If $s(v, w) < c(e)$, then no minimum Steiner tree contains e.*

Note the beautiful analogy between bottleneck distance applied to MST, and bottleneck Steiner distance applied to SPG: Any edge $e = \{v, w\}$ that satisfies $b(v, w) < c(e)$ cannot be part of an MST. Otherwise, e could be replaced by an edge of cost at most $b(v, w)$ to obtain a spanning tree of smaller cost. Any edge $e = \{v, w\}$

that satisfies $s(v,w) < c(e)$ cannot be part of a minimum Steiner tree. Otherwise, e could be replaced by a path in G corresponding to an edge in $D = D_G(T \cup \{v,w\})$ with cost at most $b_D(v,w)$. In this case, one would obtain a Steiner tree of smaller cost. We also point out that bottleneck Steiner distances can be computed in polynomial time, but in practice (heuristic) approximations are used. See Polzin and Daneshmand (2001b) for a state-of-the-art algorithm.

A stronger bottleneck concept

In the following, we describe a generalization of the bottleneck Steiner distance. Initially, for an edge $e = \{v,w\}$ define the *restricted bottleneck distance* $\overline{b}(e)$ (Polzin and Daneshmand, 2001b) as the bottleneck distance between v and w on $(V, E\setminus\{e\}, c)$.

The basis of the new bottleneck Steiner concept is formed by a node-weight function that we introduce in the following. For any $v \in V \setminus T$ and $F \subseteq \delta(v)$ define

$$p^+(v,F) := \max\left\{0, \sup\{\overline{b}(e) - c(e) \mid e \in F, e \cap T \neq \emptyset\}\right\}. \tag{2.36}$$

We call $p^+(v,F)$ the *F-implied profit* of v. The following observation motivates the subsequent usage of the implied profit. Assume that $p^+(v,\{e\}) > 0$ for an edge $e \in \delta(v)$. If a Steiner tree S contains v, but not e, then there is a Steiner tree S' with $e \in E(S')$ such that $c(E(S')) + p^+(v,\{e\}) \leqslant c(E(S))$.

Let $v,w \in V$. Consider a finite walk $W = (v_1, e_1, v_2, e_2, ..., e_{r-1}, v_r)$ with $v_1 = v$ and $v_r = w$. We say that W is a (v,w)-walk. For any $k,l \in \mathbb{N}$ with $1 \leqslant k \leqslant l \leqslant r$ define the subwalk $W(k,l) := (v_k, e_k, v_{k+1}, e_{k+1}, ..., e_{l-1}, v_l)$. W will be called *Steiner walk* if $V(W) \cap T \subseteq \{v,w\}$ and v,w are contained exactly once in W (the latter condition could be omitted, but has been added for ease of presentation). The set of all Steiner walks from v to w will be denoted by $\mathcal{W}_T(v,w)$. With a slight abuse of notation we define $\delta_W(u) := \delta(u) \cap E(W)$ for any walk W and any $u \in V$. Define the *implied Steiner cost* of a Steiner walk $W \in \mathcal{W}_T(v,w)$ as

$$c_p^+(W) := \sum_{e \in E(W)} c(e) - \sum_{u \in V(W)\setminus\{v,w\}} p^+\left(u, \delta(u) \setminus \delta_W(u)\right). \tag{2.37}$$

Further, set

$$P_W^+ := \{u \in V(W) \mid p^+\left(u, \delta(u) \setminus \delta_W(u)\right) > 0\} \cup \{v,w\}. \tag{2.38}$$

Define the *implied Steiner length* of W as

$$l_p^+(W) := \max\{c_p^+(W(v_k, v_l)) \mid 1 \leqslant k \leqslant l \leqslant r,\ v_k, v_l \in P_W^+\}. \tag{2.39}$$

Define the *implied Steiner distance* between v and w as

$$d_p^+(v,w) := \min\{l_p^+(W) \mid W \in \mathcal{W}_T(v,w)\}. \tag{2.40}$$

Note that $d_p^+(v,w) = d_p^+(w,v)$. At last, consider the distance graph $D^+ := D_G(T \cup \{v,w\}, d_p^+)$. Let b_{D^+} be the bottleneck distance in D^+. Define the *implied bottleneck Steiner distance* between v and w as

$$s_p(v,w) := b_{D^+}(v,w). \tag{2.41}$$

Note that $s_p(v,w) \leqslant s(v,w)$ and that the inequality can be strict. Indeed, $\frac{s(v,w)}{s_p(v,w)}$ can become arbitrarily large. Thus, the following result provides a strictly stronger reduction criterion than Theorem 2.10.

Theorem 2.11. *Let $e = \{v,w\} \in E$. If $s_p(v,w) < c(e)$, then no minimum Steiner tree contains e.*

Proof. Assume $s_p(v,w) < c(e)$ and let S be a Steiner tree with $e \in E(S)$. We will show the existence of a Steiner tree S' with $e \notin E(S')$ such that $c(E(S')) \leqslant c(E(S))$, which concludes the proof. First, remove e from S to obtain a new subgraph $\tilde{S}$, which consists of exactly two connected components. Assume that each connected component contains at least one terminal (otherwise the proof is already finished). Consider a (v,w)-path P in D^+ such that $bl_{D^+}(P) = b_{D^+}(v,w)$. Let $\{t,u\}$ be an edge on P such that t and u are in different connected components of $\tilde{S}$ (where t and u are considered in the original SPG). Let $\tilde{S}^t$ and $\tilde{S}^u$ be the connected components of $\tilde{S}$ such that $t \in V(\tilde{S}^t)$ and $u \in V(\tilde{S}^u)$. By the definition of the bottleneck length it holds that

$$d_p^+(t,u) \leqslant s_p(v,w). \tag{2.42}$$

Let $W \in \mathcal{W}_T(t,u)$ such that

$$l_p^+(W) = d_p^+(t,u). \tag{2.43}$$

Assume that W is given as $W = (v_1, e_1, ..., e_{r-1}, v_r)$. Define $b := \min\{k \in \{1,...,r\} \mid v_k \in V(\tilde{S}^u)\}$ and $a := \max\{k \in \{1,...,b\} \mid v_k \in V(\tilde{S}^t)\}$. Further, define $x := \max\{k \in \{1,...,a\} \mid v_k \in P_W^+\}$ and $y := \min\{k \in \{b,...,r\} \mid v_k \in P_W^+\}$. By definition, $x \leqslant a < b \leqslant y$ and furthermore:

$$\sum_{e \in E(W(a,b))} c(e) - \sum_{v \in V(W(a,b)) \setminus \{v_x, v_y\}} p^+\left(v, \delta(v) \setminus \delta_{W(x,y)}\right) \leqslant c_p^+\left(W(x,y)\right). \tag{2.44}$$

Reconnect $\tilde{S}^t$ and $\tilde{S}^u$ by $W(a,b)$, which yields a connected subgraph S'_0 with $T \subseteq V(S'_0)$. Assume that S'_0 is a tree (otherwise remove any redundant edges).[7] It holds that

$$\sum_{e \in E(S'_0)} c(e) \leqslant \sum_{e \in E(S)} c(e) + \sum_{e \in E(W(a,b))} c(e) - c(\{v,w\}). \tag{2.45}$$

Let $v_1^+, v_2^+, ..., v_z^+$ be the vertices in $P^+_{W(a,b)} \setminus \{v_a, v_b\}$. Choose for each $i = 1,...,z$ an edge $e_i^+ \in \delta(v_i^+) \setminus \delta_{W(x,y)}(v_i^+)$ such that $e_i^+ \cap T \neq \emptyset$ and

$$\overline{b}(e_i^+) - c(e_i^+) = p^+(v_i^+, \delta(v_i^+) \setminus \delta_{W(x,y)}). \tag{2.46}$$

Note that all e_i^+ are pairwise disjoint (just as the v_i^+).

[7] Because we assume all edges to be of positive cost, S'_0 will in fact always be a tree.

We will construct Steiner trees S'_i for $i \in \{1, ..., z\}$ that satisfy

$$\sum_{e \in E(S'_i)} c(e) \leqslant \sum_{e \in E(S'_0)} c(e) - \sum_{k=1}^{i} p^+(v_k^+, \delta(v) \setminus \delta_{W(x,y)}), \tag{2.47}$$

as well as

$$\bigcup_{k=i+1}^{z} \{e_k^+\} \cap E(S'_i) = \emptyset, \tag{2.48}$$

and

$$V(S'_i) = V(S'_0). \tag{2.49}$$

One readily verifies that S'_0 satisfies (2.47)-(2.49). Let $i \in \{1, ..., z\}$ and assume that (2.47)-(2.49) hold for S'_{i-1}. Thus, $e_i^+ \notin E(S'_{i-1})$. Let P_i be the (unique) path in S'_{i-1} between v_i^+ and the terminal t_i with $\{t_i\} = e_i^+ \cap T$. Choose any $\tilde{e}_i \in E(P_i)$ with $c(\tilde{e}_i) = bl(P_i)$. Define the tree S'_i by $V(S'_i) := V(S'_{i-1})$ and $E(S'_i) := \big(E(S'_{i-1}) \setminus \{\tilde{e}_i\}\big) \cup \{e_i^+\}$. We claim that S'_i satisfies (2.47)-(2.49). Equality (2.48) follows from the fact that all e_i^+ are disjoint. And (2.49) follows from the construction of S'_i. For (2.47), observe that by definition of the bottleneck distance it holds that $c(\tilde{e}_i) \geqslant \overline{b}(e_i^+)$ and therefore

$$\overline{b}(e_i^+) - c(e_i^+) \leqslant c(\tilde{e}_i) - c(e_i^+). \tag{2.50}$$

Thus, equation (2.46) implies that S'_i satisfies (2.47).

Finally, set $S' := S'_z$. Because of (2.49) it holds that $T \subseteq V(S')$. Furthermore, one obtains:

$$\sum_{e \in E(S')} c(e) \overset{(2.47)}{\leqslant} \sum_{e \in E(S'_0)} c(e) - \sum_{k=1}^{z} p^+(v_k^+, \delta(v_k^+) \setminus \delta_{W(x,y)}) \tag{2.51}$$

$$\overset{(2.45)}{\leqslant} \sum_{e \in E(S)} c(e) + \sum_{e \in E(W(a,b))} c(e) - c(\{v,w\}) - \sum_{k=1}^{z} p^+(v_k^+, \delta(v_k^+) \setminus \delta_{W(x,y)}) \tag{2.52}$$

$$\overset{(2.44)}{\leqslant} \sum_{e \in E(S)} c(e) - c(\{v,w\}) + c_p^+(W(x,y)) \tag{2.53}$$

$$\overset{(2.43)}{\leqslant} \sum_{e \in E(S)} c(e) - c(\{v,w\}) + l_p^+(W) \tag{2.54}$$

$$\overset{(2.42)}{\leqslant} \sum_{e \in E(S)} c(e) - c(\{v,w\}) + s_p(v,w) \tag{2.55}$$

$$\leqslant \sum_{e \in E(S)} c(e), \tag{2.56}$$

where the last inequality follows from the initial assumptions. □

Furthermore, we define the *restricted implied bottleneck Steiner distance* $\overline{s}_p(v, w)$ between any $v, w \in V$ as the implied bottleneck Steiner distance between v and w in the SPG $(V, E \setminus \{\{v, w\}\}, c)$. One obtains the following corollary.

Corollary 2.12. *Let $e = \{v, w\} \in E$. If $\overline{s}_p(v, w) \leqslant c(e)$, then at least one minimum Steiner tree does not contain e.*

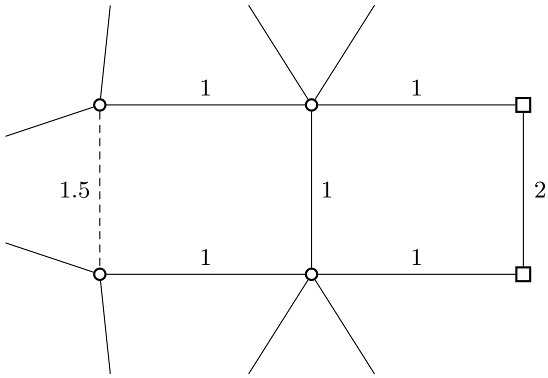


Figure 2.3: Segment of a Steiner tree instance. Terminals are drawn as squares. The dashed edge can be deleted by employing Theorem 2.11.

Figure 2.3 shows a segment of an SPG instance for which Theorem 2.11 allows for the deletion of an edge, but Theorem 2.10 does not. The implied bottleneck Steiner distance between the endpoints of the dashed edge is 1—corresponding to a walk along the four non-terminal vertices. The edge can thus be deleted. In contrast, the (standard) bottleneck Steiner distance between the endpoints is 1.5 (corresponding to the edge itself). Unfortunately, already computing the implied Steiner distance is hard, as the following proposition shows.

Proposition 2.13. *Computing the implied Steiner distance is $\mathcal{NP}$-hard.*

The proposition can for example be proved by a reduction from the Hamiltonian path problem. See also Section 3.3.2, which shows the $\mathcal{NP}$-hardness of a related concept for the maximum-weight connected problem. A proof of Proposition 2.13 can be formulated along the same lines, but with more technicalities. Thus, we omit it here.

Despite this $\mathcal{NP}$-hardness, one can devise heuristics that provide upper bounds on s_p. These upper bounds are always at least as strong as those used for s, and are empirically often stronger. We will discuss one such heuristic in Section 6.2.2.

Bottleneck Steiner reductions beyond edge deletion

This section discusses applications of the implied bottleneck Steiner distance that allow for additional reduction operations: Edge contraction and node replacement. We start with the former. For an edge e and vertices v, w define $b_e(v,w)$ as the bottleneck distance between v and w on $(V, E\setminus\{e\}, c)$. With this definition at hand, we introduce a generalization of the classic *NSV* reduction test from Duin and Volgenant (1989b).

Proposition 2.14. *Let $\{v,w\} \in E$ and $t_i, t_j \in T, t_i \neq t_j$. If*

$$s_p(v,t_i) + c(\{v,w\}) + s_p(w,t_j) \leqslant b_{\{v,w\}}(t_i,t_j), \tag{2.57}$$

then there is a minimum Steiner tree S with $\{v,w\} \in E(S)$.

Proof sketch. Unfortunately, the use of the implied bottleneck Steiner distance makes the proof of the proposition far more difficult than that of the original result from Duin and Volgenant (1989b). To avoid an abundance of technicalities, we therefore only provide a proof sketch.

Assume there is an optimal solution S such that $\{v,w\} \notin E(S)$. Remove from $E(S)$ an edge on the (unique) path between t_i and t_j in S of maximum cost. This operation results in two disjoint trees: S_i with $t_i \in S_i$ and S_j with $t_j \in S_j$. By definition of $b_{\{v,w\}}(t_i,t_j)$ it holds that

$$c(E(S_i)) + c(E(S_j)) + b_{\{v,w\}}(t_i,t_j) \leqslant c(E(S)). \tag{2.58}$$

Now the sketchy part starts: Similar to the proof of Theorem 2.11, condition (2.57) allows us to connect S_i to v such that the resulting tree $\tilde{S}_i$ satisfies

$$c(E(\tilde{S}_i)) \leqslant c(E(S_i)) + s_p(v,t_i). \tag{2.59}$$

Equivalently, we can connect S_j to w with the result satisfying

$$c(E(\tilde{S}_j)) \leqslant c(E(S_j)) + s_p(w,t_j). \tag{2.60}$$

However, the above is only true, because the two Steiner walks that correspond to $s_p(v,t_i)$ and $s_p(w,t_j)$ in (2.59) and (2.60), respectively, have no vertex in common. If they had a vertex in common, one could build a new Steiner walk W_0 with $l_p^+(W_0) \leqslant s_p(v,t_i) + s_p(w,t_j)$ out of the two above Steiner walks, such that W_0 connects S_i and S_j. This walk W_0 could then be used to reconnect S_i and S_j to a Steiner tree of weight smaller than $b_{\{v,w\}}(t_i,t_j)$.

Finally, we define $\tilde{S}$ as the union of $\tilde{S}_i$, $\tilde{S}_j$, and $\{v,w\}$. This connected subgraph is not necessarily a tree, but can be made one without increasing $c(E(\tilde{S}))$ by deleting an edge from each cycle. From (2.58), (2.59), and (2.60) it follows that

$$c(E(\tilde{S})) \leqslant c(E(S)), \tag{2.61}$$

which concludes the proof. □

If criterion (2.57) is satisfied, one can contract edge $\{v,w\}$ and make the resulting vertex a terminal. The original criterion from Duin and Volgenant (1989b) uses the standard distance in (2.57) instead of the implied bottleneck Steiner distance. We note that using the (standard) bottleneck Steiner distance in (2.57) does not improve the

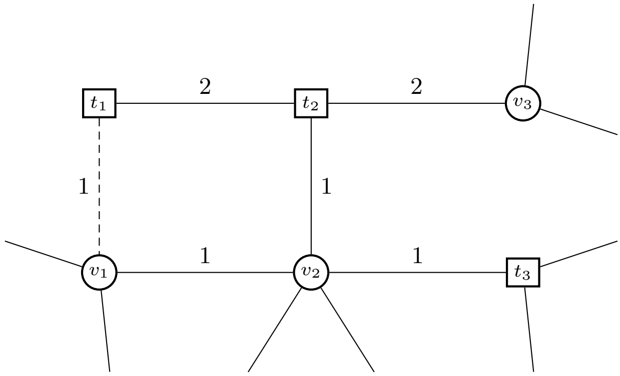


Figure 2.4: Segment of a Steiner tree instance. Terminals are drawn as squares. The dashed edge can be contracted by employing Proposition 2.14.

original test. However, using the implied bottleneck Steiner distance leads to a strictly stronger criterion, as the example in Figure 2.4 shows. Note that $b_{\{t_1,v_1\}}(t_1,t_3) = 2$ and $s_p(v_1,t_3) = 1$. Thus, (2.57) is satisfied for edge $\{t_1,v_1\}$ and terminals t_1,t_3.

The following proposition allows one to identify edges that are candidates for edge contraction. Afterwards, the bottleneck distances can be computed for all these edges in $O(m + n \log n)$ amortized time (Duin, 1993).

Proposition 2.15. *Let $\{v,w\} \in E$ and $t_i, t_j \in T, t_i \neq t_j$. If* (2.57) *holds, then there is a minimum spanning tree S_{MST} on (V,E,c) such that $\{v,w\} \in E(S_{MST})$.*

Proof. Assume there is a spanning tree S such that $\{v,w\} \notin E(S)$. Remove from $E(S)$ an edge on the (unique) path between t_i and t_j in S of maximum cost. This operation results in two disjoint trees: S_i with $t_i \in S_i$ and S_j with $t_j \in S_j$. By definition of $b_{\{v,w\}}(t_i,t_j)$ it holds that

$$c(E(S_i)) + c(E(S_j)) + b_{\{v,w\}}(t_i,t_j) \leqslant c(E(S)). \tag{2.62}$$

If v and w are in different trees, one can add $\{v,w\}$ to connect S_i and S_j and obtain a spanning tree of no higher cost than S. Otherwise, assume that $v,w \in V(S_j)$. Let W_i be a Steiner walk from v to t_i with $l_p^+(W_i) = s_p(v,t_i)$. There is at least one edge $\{p,q\} \in E(W_i)$ such that $p \in V(S_i)$ and $q \in V(S_j)$. By definition it holds that $c(\{p,q\}) \leqslant l_p^+(W_i)$. Thus, one can add both $\{p,q\}$ and $\{v,w\}$ to S_i, S_j to obtain a connected spanning subgraph S'. Because of condition (2.57) and (2.62) it holds that

$$c(E(S')) \leqslant c(E(S)). \tag{2.63}$$

Delete any edge other than $\{v,w\}$ on the cycle in $E(S')$ that includes $\{v,w\}$. In this way one obtains a spanning tree S'' of no higher cost than S. $\square$

Now we turn to a different reduction operation. To this end, we first introduce a reduction criterion based on the standard bottleneck Steiner distance. Besides being a new technique, this result also serves to highlight the complications that arise if one attempts to formulate similar conditions based on the implied bottleneck Steiner distance.

Proposition 2.16. *Let $D := D_G(T, d)$. Let Y be a minimum spanning tree in D. Write its edges $\{e_1^Y, e_2^Y, ..., e_{|T|-1}^Y\} := E(Y)$ in non-ascending order with respect to their weight in D. Let $v \in V \setminus T$. If for all $\Delta \subseteq \delta(v)$ with $|\Delta| \geqslant 3$ it holds that:*

$$\sum_{i=1}^{|\Delta|-1} d(e_i^Y) \leqslant \sum_{e \in \Delta} c(e), \tag{2.64}$$

then there is at least one minimum Steiner tree S such that $|\delta_S(v)| \leqslant 2$.

The proposition follows from Corollary 2.29, which we will introduce in Section 2.4. If the conditions (2.64) are satisfied for a vertex $v \in V \setminus T$, one can *pseudo-eliminate* (Duin and Volgenant, 1989b) or *replace* (Polzin, 2003) vertex v, i.e., delete v and connect any two distinct vertices $u, w \in N(v)$ by a new edge $\{u, w\}$ of weight $c(\{v, u\}) + c(\{v, w\})$. A vertex replacement might require additional edges to be added. However, these edges can often be removed by other methods, such as the criterion from Theorem 2.11. In the implementation for this thesis, vertices are only replaced if the total number of edges is not increased.

The SPG depicted in Figure 2.5 exemplifies why Proposition 2.16 cannot be formulated by using the implied Steiner distance. The weight of the minimum spanning tree Y for $D_G(T, d)$ is 4, but the weight of a minimum spanning tree with respect to the implied bottleneck Steiner distance is 2. Similarly also the $\mathrm{NTD}_{\mathbf{k}}$ reduction technique described below cannot be directly formulated by using the implied bottleneck distance. Still, it is possible to formulate a similar criterion that makes use of the implied bottleneck distance. Unfortunately, both the result and the corresponding proof are more involved than those of their edge elimination counterparts (see Theorem 2.11). Thus, we omit the details here. The important point is to make sure that the selected Steiner walks do not overlap at vertices with a positive implied profit. The code developed for this thesis only includes a very limited implementation of these replacement methods.

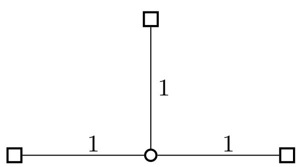


Figure 2.5: SPG instance. Terminals are drawn as squares

The (standard) bottleneck Steiner distance can further be utilized for another classic reduction test: *Non-Terminals of Degree k* (*$NTD_{\mathbf{k}}$*) which was introduced in Duin and Volgenant (1989b) and is based on the following proposition:

Proposition 2.17. *Let $v \in V \setminus T$. There is a minimum Steiner tree S with $|\delta_S(v)| \leqslant 2$ if for each $\Delta \subseteq N(v)$ with $|\Delta| \geqslant 3$ the following holds: $c(\delta(v) \cap \delta(\Delta))$ is not less than the weight of a minimum spanning tree for the distance network $D_G(\Delta, s)$.*

A proof is given in Duin and Volgenant (1989b). Also, the proposition is a special case of Corollary 2.29, which will be introduced in Section 2.4.

2.3.2 Bound-based reduction techniques

Bound-based reduction techniques are preprocessing methods that identify edges and vertices for elimination by examining whether they induce an lower bound that exceeds a given upper bound (Polzin and Daneshmand, 2001b). In this section a bound-based reduction concept is introduced that generalizes the Voronoi-regions concept from Polzin and Daneshmand (2001b). Note that the bounding technique described in this section can be seen as a special case of the bound-based reduction technique for the prize-collecting Steiner tree problem, which will be introduced in Chapter 4. Thus, no proofs are provided for most of the results in this section.

Terminal-regions decomposition

The base of the reduction technique is the following new concept: a *terminal-regions decomposition* of an SPG—with underlying graph (V, E)—is a partition $H = \{H_t \subseteq V \mid t \in T\}$ of V such that for each $t \in T$ the subgraph $(H_t, E[H_t])$ is connected and $T \cap H_t = \{t\}$ holds. Each of the H_t will be called a *region* of H. Define for all $t \in T$

$$r_H(t) := \min\{d(t, v) \mid v \notin H_t\}. \tag{2.65}$$

In Polzin and Daneshmand (2001b) a special terminal-regions decomposition called *Voronoi-regions decomposition* is used. The more general results presented here allow us to improve on the Voronoi preprocessing methods introduced in Polzin and Daneshmand (2001b). However, it will also turn out that finding an optimal terminal-regions decomposition is $\mathcal{NP}$-hard. The following three propositions not only improve on the results from Polzin and Daneshmand (2001b) by using a more general decomposition, but also by making use of the following distance function. Given vertices $v_i, v_j \in V$ define $\underline{d}(v_i, v_j)$ as the length of a shortest path between v_i and v_j without intermediary terminals. In Duin (1993) an $O(m + n \log n)$ algorithm was introduced to compute for each non-terminal v_i a constant number of r $\underline{d}$-nearest terminals $\underline{v}_{i,1}, \underline{v}_{i,2}, ..., \underline{v}_{i,r}$ (if existent) along with the corresponding paths. In the remainder of this section it will be assumed that a terminal-regions decomposition H is given. Moreover, for ease of presentation it will be assumed that the terminals $T = \{t_1, t_2, ..., t_k\}$ are ordered such that $r_H(t_1) \leqslant r_H(t_2) \leqslant ... \leqslant r_H(t_k)$.

Proposition 2.18. *Let $v_i \in V \setminus T$ and set $k := |T|$. If there is a minimum Steiner tree S such that $v_i \in V(S)$, then*

$$\underline{d}(v_i, \underline{v}_{i,1}) + \underline{d}(v_i, \underline{v}_{i,2}) + \sum_{q=1}^{k-2} r_H(t_q) \tag{2.66}$$

is a lower bound on the weight of S.

Each vertex $v_i \in V \setminus T$ such that the affiliated lower bound stated in Proposition 2.18 exceeds a known upper bound can be eliminated. Moreover, if a solution S corresponding to the upper bound is given and v_i is not contained in it, the latter can already be eliminated if the lower bound stated in Proposition 2.18 is equal to the cost of S. A similar proposition holds for edges in a minimum Steiner tree:

Proposition 2.19. *Let $\{v_i, v_j\} \in E$ and set $k := |T|$. If there is minimum Steiner tree S such that $\{v_i, v_j\} \in E(S)$, then L defined by*

$$L := c(\{v_i, v_j\}) + \underline{d}(v_i, \underline{v}_{i,1}) + \underline{d}(v_j, \underline{v}_{j,1}) + \sum_{q=1}^{k-2} r_H(t_q) \tag{2.67}$$

if $\underline{v}_{i,1} \neq \underline{v}_{j,1}$ and

$$L := c(\{v_i, v_j\}) + \min\{\underline{d}(v_i, \underline{v}_{i,1}) + \underline{d}(v_j, \underline{v}_{j,2}), \underline{d}(v_i, \underline{v}_{i,2}) + \underline{d}(v_j, \underline{v}_{j,1}\} + \sum_{q=1}^{k-2} r_H(t_q) \tag{2.68}$$

otherwise, is a lower bound on the weight of S.

If the above lower bound L for an edge $e \in E$ exceeds a known upper bound, e can be eliminated. The following proposition allows us to replace vertices.

Proposition 2.20. *Let $v_i \in V \setminus T$. If there is a minimum Steiner tree S such that $\delta_S(v_i) \geqslant 3$, then*

$$\underline{d}(v_i, \underline{v}_{i,1}) + \underline{d}(v_i, \underline{v}_{i,2}) + \underline{d}(v_i, \underline{v}_{i,3}) + \sum_{q=1}^{k-3} r_H(t_q) \tag{2.69}$$

with $k := |T|$ is a lower bound on the weight of S.

To efficiently apply Proposition 2.18, one would like to maximize (2.66)—and for Proposition 2.19 and Proposition 2.20 to maximize (2.67) and (2.69), respectively. Unfortunately, this problem turns out to be $\mathcal{NP}$-hard. The decision variant of the problem can be stated as follows. Let $\alpha \in \mathbb{N}_0$ and let $G_0 = (V_0, E_0)$ be an undirected, connected graph with edge cost $c : E \to \mathbb{N}$. Furthermore, let $T_0 \subseteq V$, and assume that $\alpha < |T_0|$. For each terminal-regions decomposition H_0 of G_0 define $T_0' \subsetneq T_0$ such that $|T_0'| = \alpha$ and $r_{H_0}(t') \geqslant r_{H_0}(t)$ for all $t' \in T_0'$ and $t \in T_0 \setminus T_0'$. Let:

$$C_{H_0} := \sum_{t \in T_0 \setminus T_0'} r_{H_0}(t). \tag{2.70}$$

We now define the α *terminal-regions decomposition problem* as follows: Given a $k \in \mathbb{N}$, is there a terminal-regions decomposition H_0 such that $C_{H_0} \geqslant k$? In the following proposition it is shown that this problem is $\mathcal{NP}$-complete, which forthwith establishes the $\mathcal{NP}$-hardness of finding a terminal-regions decomposition that maximize (2.66), (2.67), (2.68), or (2.69)—which corresponds to $\alpha = 2$ and $\alpha = 3$, respectively.

Proposition 2.21. *For each $\alpha \in \mathbb{N}_0$ the α terminal-regions decomposition problem is $\mathcal{NP}$-complete.*

Proof. Given a terminal-regions decomposition H_0 it can be tested in polynomial time whether $C_{H_0} \geqslant k$. Consequently, the terminal-regions decomposition problem is in $\mathcal{NP}$.

In the remainder it will be shown that the, $\mathcal{NP}$-complete (Garey and Johnson, 1979), independent set problem can be reduced to the terminal-regions decomposition problem. To this end, let $G_{ind} = (V_{ind}, E_{ind})$ be an undirected, connected graph and $k \in \mathbb{N}$. The problem is to determine whether an independent set in G_{ind} of cardinality at least k exists. To establish the reduction, construct a graph G_0 from G_{ind} as follows. Initially, set $G_0 = (V_0, E_0) := G_{ind}$, and $T_0 := V_0$. Next, extend G_0 by replacing each edge $e_l = \{v_i, v_j\} \in E_0$ with a vertex v'_l and the two edges $\{v_i, v'_l\}$ and $\{v_j, v'_l\}$. Define edge weights $c_0(e) = 1$ for all $e \in E_0$ (which includes the newly added edges). If $\alpha > 0$, choose an arbitrary $v_i \in V_0 \cap V_{ind}$ and add for $j = 1, ..., \alpha$ vertices $t_i^{(j)}$ to both V_0 and T_0. Finally, add for $j = 1, ..., \alpha$ edges $\{v_i, t_i^{(j)}\}$ with $c_0(\{v_i, t_i^{(j)}\}) = 2$ to E_0.

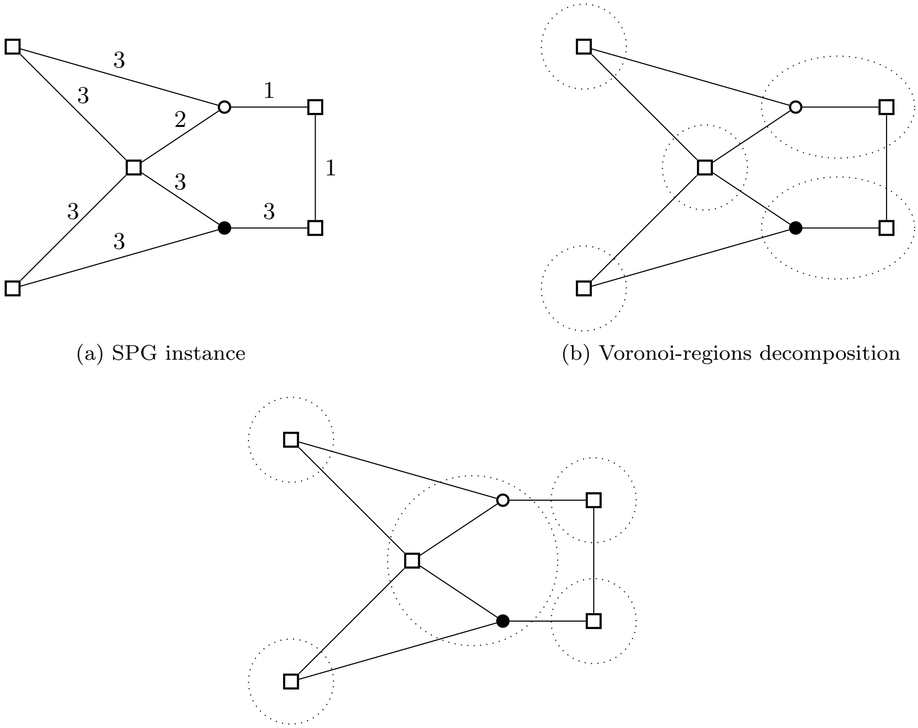
First, one observes that the size $|V_0| + |E_0|$ of the new graph G_0 is a polynomial in the size $|V_{ind}| + |E_{ind}|$ of G_{ind}. Next, $r_{H_0}(v_i) = 2$ holds for a vertex $v_i \in G_0 \cap G_{ind}$ if and only if H_{v_i} contains all (newly inserted) adjacent vertices of v_i in G_0. Moreover, in any terminal-regions decomposition H_0 for (G_0, c_0), it holds that $r_{H_0}(t_i^{(j)}) = 2$ for $j = 1, ..., \alpha$. Hence, there is an independent set in G_{ind} of cardinality at least k if and only if there is a terminal-regions decomposition H_0 for (V_0, E_0, T_0, c_0) such that

$$C_{H_0} \geqslant |V_{ind}| + k.$$

This proves the proposition. □

Figure 2.6 depicts an SPG, a corresponding Voronoi-regions decomposition as described in Polzin and Daneshmand (2001b), and an alternative terminal-regions decomposition. The second terminal-regions decomposition yields a stronger lower bound than the Voronoi-regions decomposition and indeed allows to eliminate a vertex if an upper bound that is sufficiently close to the optimal solution value is known.

For computing a terminal-regions decomposition, we, unsurprisingly, resort to heuristic methods. More details are given in Section 4.3, where an extension of the terminal-regions decomposition to the prize-collecting Steiner tree problem is introduced. Computational experiments for this thesis have shown that it is in many cases possible to improve on the bound provided by the Voronoi-regions decomposition,



(a) SPG instance

(b) Voronoi-regions decomposition

(c) terminal-regions decomposition

Figure 2.6: Illustration of a Steiner tree instance (a), a Voronoi-decomposition (b), and a second terminal-regions decomposition (c). Terminals are drawn as squares. If an upper bound less than 11 is known, the vertex drawn filled in (c) can be deleted by means of the terminal-regions decomposition depicted in (c), but not by means of the Voronoi-regions decomposition.

and that significantly stronger graph reductions can be achieved. Still, empirically the methods only work well for (some) sparse instances with few terminals. Thus, we only execute the terminal-regions decomposition tests for these kind of instances.

Reduced-costs reductions

In Wong (1984) a dual-ascent algorithm for the SAP was introduced that, empirically, both provides strong lower bounds and allows for fast computation—defying its worst-case time complexity of $O(|A| \min\{|V||T|, |A|\})$ (Polzin and Daneshmand, 2001b). Practically efficient implementations of this algorithm can be found in Duin (1993) and Pajor et al. (2017). We use an implementation that is similar to these two. In Section 3.4 we give a description of the dual-ascent algorithm (in the context of the maximum-weight connected subgraph problem).

At termination, dual-ascent provides a dual solution to the LP-relaxation of

Formulation 1.1. The reduced-costs of this dual LP solution can be used for an SPG reduction criterion, see e.g. Duin (1993). Given an SPG instance, consider an equivalent (bidirected) SAP (V, A, T, c, r). Let $v \in V/T$, and let $S^\star$ be an optimal Steiner arborescence to the given SAP instance. Let L_{DA} be the lower bound obtained by dual-ascent. If $S^\star$ contains v, the weight of $S^\star$ can be bounded from below by L_{DA} plus the length (with respect to the reduced-costs provided by dual-ascent) of a shortest path from the r to v and the length of a shortest path from v to a closest terminal (other than the root). Hence, v can be deleted if the just defined bound exceeds a known upper bound U. Similarly, an arc (v, w) can be deleted if its reduced-cost plus the reduced-cost distance from r to v plus the reduced-cost distance from w to a closest terminal exceeds $U - L_{DA}$. The test can be extended to the case of equality if a solution corresponding to U is given—and if the arc to be eliminated is not contained in this solution. Whenever a vertex can be deleted in the SAP, the same is true for its counterpart in the original SPG. Similarly, if two anti-parallel arcs of the SAP have been shown to be removable, the corresponding edge of the SPG can be discarded. Finally, one can formulate a similar test to replace (or pseudo-eliminate) vertices.

The above procedure can also be performed by using the reduced-costs obtained during a branch-and-cut solution based on *BDCut*. Instead of deleting edges or vertices, one fixes the corresponding variables to 0 in the IP formulation. However, vertex replacements cannot be directly transferred to the IP formulation, see Section 2.6 for more details.

2.3.3 Further reduction techniques

Besides the methods described already, the literature describes a large number of further SPG reduction techniques. See e.g. Hwang et al. (1992) for an overview of reduction techniques published before 1992. An overview of newer methods is given in Polzin (2003). However, most of these methods are dominated by (or are special cases of) the techniques described in Duin (1993); Polzin and Daneshmand (2001b, 2002). As we have seen in this chapter, the latter reduction techniques are in turn dominated by the new techniques introduced in this thesis.

In particular, several trivial reduction methods are only special cases of the methods introduced so far. For example, a simple reduction method is to delete any Steiner vertex of degree 1. This method is, however, just a special case of Proposition 2.17. Still, one independent class of SPG reduction methods has not been covered so far. We will shortly describe those methods in the following.

Terminal separator methods

It is well known that bi-connected components of an SPG instance can be solved independently, see e.g. Hwang et al. (1992). In Polzin and Daneshmand (2006) a more general decomposition method is introduced, based on *terminal separators*. Let $I = (V, E, T, c)$ be an SPG instance. A $T' \subset T$ is called terminal separator if $G' := (V \setminus T', E[V \setminus T'])$ is not connected. In the case of $|T'| > 1$, the biconnected components of G' cannot be solved independently, but a case distinction is necessary.

In Polzin and Daneshmand (2006) several techniques are described to speed-up this case distinction, based on the bottleneck Steiner distance. We note that one could also use the implied bottleneck Steiner distance introduced in this thesis instead. Notwithstanding the improvements by Polzin and Daneshmand (2006), the case distinction can still be prohibitively expensive on large bi-connected components. Thus, Polzin and Daneshmand (2006) also use reduction tests on (small) individual bi-connected components. If an edge is contained in an optimal solution to the sub-SPG for all possible cases, it must be included in the original instance I. Conversely, if an edge is never contained in an optimal solution to a sub-SPG, it can be removed from the original instance I. Finally, Polzin and Daneshmand (2006) describe a sophisticated bound-based reduction approach that does not require explicit case distinction.

For this thesis, we have only fully implemented the latter bound-based approach. We use a limited version of the exact approach. To find terminal separators, we use a maximum-flow algorithm on a split-graph obtained from the given SPG instance I—as suggested in Polzin and Daneshmand (2006). We use a newly implemented maximum-flow algorithm with warm start-capabilities, described in Section 6.2.4.

2.3.4 From reductions to conflicts

This section shows an additional advantage of the node replacement reduction: The creation of conflicts between the newly inserted edges. Furthermore, a new replacement operation is introduced. We say that a set $E' \subset E$ with $|E'| \geqslant 2$ is in *conflict* if no minimum Steiner tree contains more than one edge of E'.

Node replacement

Unfortunately, this section requires some additional technicalities regarding reduction methods. Recall that we have seen three types of reductions so far: Edge deletion, edge contraction, and node replacement. For simplicity, we assume in the following that a reduction is only performed if it retains *all* optimal solutions. E.g., we only delete an edge if we can show that there is no minimum Steiner tree that contains this edge. We say that such a reduction is *valid*. We start with an SPG instance $I = (G, T, c)$, and consider a series of subsequent, valid reductions (of one of the three above types) that are applied to I. In each reduction step $i \geqslant 0$, the current instance $I^{(i)} = (G^{(i)}, T^{(i)}, c^{(i)})$ is transformed to instance $I^{(i+1)} = (G^{(i+1)}, T^{(i+1)}, c^{(i+1)})$. We set $I^{(0)} := I$. We define ancestor information for each $i = 0, 1, ..., k$ by $\Pi^{(i)} : E^{(i)} \to \mathcal{P}(E)$ and $\Pi^{(i)}_{FIX} \subseteq E$. We set $\Pi^{(0)}(e) := \{e\}$ for all $e \in E$, and $\Pi^{(0)}_{FIX} = \emptyset$. Consider a reduced instance $I^{(i)}$. If we contract an edge $e \in E^{(i)}$, we set $\Pi^{(i+1)}_{FIX} := \Pi^{(i)}_{FIX} \cup \Pi^{(i)}(e)$; otherwise, we set $\Pi^{(i+1)}_{FIX} := \Pi^{(i)}_{FIX}$. If we replace a vertex $v \in V^{(i)}$, we set for each newly inserted edge $\{u, w\}$—with $u, w \in N(v)$—$\Pi^{(i+1)}(\{u, w\}) := \Pi^{(i)}(\{v, u\}) \cup \Pi^{(i)}(\{v, w\})$. For all other remaining edges e we set $\Pi^{(i+1)}(e) := \Pi^{(i)}(e)$. Overall, one observes the following.

Observation 2.22. *Let I be an SPG and let $I^{(k)}$ be the SPG obtained from performing a series of k valid reductions on I. For any Steiner tree $S^{(k)}$ for $I^{(k)}$, the* *tree S with*

$$E(S) = \bigcup_{e \in E^{(k)}(S^{(k)})} \Pi^{(k)}(e) \cup \Pi^{(k)}_{FIX}$$

is a Steiner tree for I, and it holds that

$$c\left(E(S)\right) = c^{(k)}\left(E^{(k)}(S^{(k)})\right) + c\left(\Pi^{(k)}_{FIX}\right).$$

Furthermore, if $S^{(k)}$ is optimal for $I^{(k)}$, then S is optimal for I.

Polzin and Daneshmand (2002) observed that two edges that originate from a common edge by a series of replacements cannot both be contained in a minimum Steiner tree. Using the above notation, we can formulate the condition as follows: If $e_1, e_2 \in E^{(k)}$ satisfy $\Pi^{(k)}(e_1) \cap \Pi^{(k)}(e_2) \neq \emptyset$, then there is no minimum Steiner tree that contains both e_1 and e_2. As we will see in Section 2.4, such conflict information can be used for further reductions.

In the following, we will introduce an edge conflict criterion that is strictly stronger than the one from Polzin and Daneshmand (2002). Initially, we define additional ancestor information for each $i = 0, 1, ..., k$. Namely, sets of *replacement ancestors* $\Lambda^{(i)} : E^{(i)} \to \mathcal{P}(\mathbb{N})$, and $\Lambda^{(i)}_{FIX} \in \mathcal{P}(\mathbb{N})$. We set $\Lambda^{(0)}(e) := \emptyset$ for all $e \in E$, and $\Lambda^{(0)}_{FIX} := \emptyset$. Further, we define $\lambda^{(0)} := 0$. Consider a reduced instance $I^{(i)}$. If we contract an edge $e \in E^{(i)}$, we set $\Lambda^{(i+1)}_{FIX} := \Lambda^{(i)}_{FIX} \cup \Lambda^{(i)}(e)$. If we replace a vertex $v \in V^{(i)}$, we set $\lambda^{(i+1)} := \lambda^{(i)} + 1$. Further, we define the replacement ancestors for each newly inserted edge $\{u, w\}$, with $u, w \in N(v)$, as follows:

$$\Lambda^{(i+1)}(\{u, w\}) := \Lambda^{(i)}(\{v, u\}) \cup \Lambda^{(i)}(\{v, w\}) \cup \{\lambda^{(i)}\}.$$

If no node replacement is performed, we set $\lambda^{(i+1)} := \lambda^{(i)}$. For the replacement ancestors one obtains the following technical, but nevertheless important, result (proven in Appendix A.1.1).

Proposition 2.23. *Let I be an SPG and let $I^{(k)}$ be the SPG obtained from performing a series of k valid reductions on I. Further, let $e_1, e_2 \in E^{(k)}$. If $\Lambda^{(k)}(e_1) \cap \Lambda^{(k)}(e_2) \neq \emptyset$, then no minimum Steiner tree $S^{(k)}$ for $I^{(k)}$ contains both e_1 and e_2.*

Corollary 2.24. *Let I, $I^{(k)}$ as in Proposition 2.23, and let $e \in E^{(k)}$. If $\Lambda^{(k)}(e) \cap \Lambda^{(k)}_{FIX} \neq \emptyset$, then no minimum Steiner tree $S^{(k)}$ for $I^{(k)}$ contains e.*

Note that any edge e as in Corollary 2.24 can be deleted.

Edge replacement

This subsection introduces a new replacement operation, whose primary benefit lies in the conflicts it creates. We start with a condition that allows us to perform this operation.

Proposition 2.25. *Let $e = \{v, w\} \in E$ with $e \cap T = \emptyset$. Define*

$$\mathcal{D} := \{\Delta \subseteq (\delta(v) \cup \delta(w)) \setminus \{e\} \mid \Delta \cap \delta(v) \neq \emptyset, \Delta \cap \delta(w) \neq \emptyset\}.$$

For any $\Delta \in \mathcal{D}$ let

$$U_\Delta := \{u \in V \mid \{u, v\} \in \Delta \vee \{u, w\} \in \Delta\}.$$

If for all $\Delta \in \mathcal{D}$ with $|\Delta| \geqslant 3$ the weight of a minimum spanning tree on $D_G(U_\Delta, s)$ is smaller than $c(\Delta)$, then each minimum Steiner tree S satisfies $|\delta_S(v)| \leqslant 2$ and $|\delta_S(w)| \leqslant 2$.

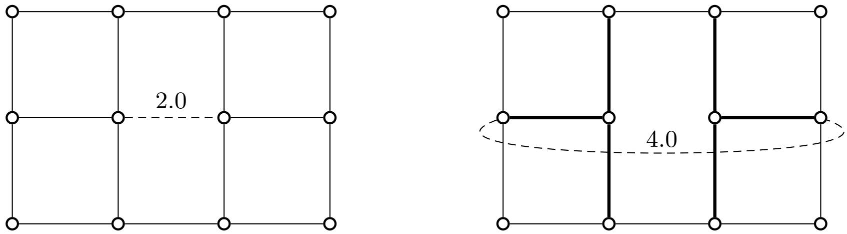
The proposition can be proven by using Corollary 2.29, which will be introduced in Section 2.4. If the condition of Proposition 2.25 is successful, we can perform what we will call a *path replacement* of e: We delete e and add for each pair $p, q \in V$ with $p \in N(v) \setminus \{w\}, q \in N(w)) \setminus \{v\}, p \neq q$ an edge $\{p, q\}$ with weight $c(\{p, v\}) + c(\{v, w\}) + c(\{q, w\})$. At first glance, the apparent increase in the number of edges by this operation seems highly disadvantageous. However, due to the increased weight, the new edges can often be deleted by using the criterion from Theorem 2.11. Furthermore, an edge does not need to be inserted if any two of the three edges it originates from have a common replacement ancestor. Indeed, we only perform a path replacement if at most one of the new edges needs to be inserted. The case that all new edges can be deleted is in principle also covered by the extended reduction technique introduced in the next section (albeit being potentially far more expensive). If exactly one new edge remains, we create new replacement ancestors as follows: Let $\hat{e} = \{p, q\}$ be the newly inserted edge. Initially, set $\lambda^{(i+1)} := \lambda^{(i)}$ and $\Lambda^{(i+1)}(\hat{e}) := \Lambda^{(i)}(\{p, v\}) \cup \Lambda^{(i)}(\{v, w\}) \cup \Lambda^{(i)}(\{v, q\})$. Next, for each $e' \in (\delta(v) \cup \delta(w)) \setminus \{e\}$ increment $\lambda^{(i+1)}$, and add $\lambda^{(i+1)}$ to $\Lambda^{(i+1)}(\hat{e})$ and $\Lambda^{(i+1)}(e')$. One can show that Proposition 2.23 remains valid if path replacement is added to the list of valid reduction operations.

Figure 2.7 illustrates an application of Proposition 2.25. In this example, all but one replacement edges can be deleted by using a simple alternative path argument. While the number of edges remains unchanged, six new conflicts are created.

2.4 From Steiner distances and conflicts to extended reduction techniques

At the end of the last section (Section 2.3.4) we have seen a reduction method that inspects a number of trees (of depth 3) that extend an edge considered for replacement. This section continues along this path, based on the reduction concepts introduced so far.

Given a tree Y (e.g. a single edge), *extended reduction techniques* use an enumeration of trees that contain Y to show that there is an optimal Steiner tree that does not contain Y. The trees are built by iteratively enlarging or *extending* Y. During this process, reduction, conflict, and implication techniques are employed to rule out these extensions of Y. In this way, extended reduction techniques are loosely related to the concepts of probing and conflict (graph) analysis for MIP, see e.g. Achterberg (2007a); Savelsbergh (1994).



(a) SPG instance segment (b) Segment after edge replacement

Figure 2.7: Segment of a Steiner tree instance (showing only non-terminals). All edges except for the dashed ones have unit weight. The dashed edge in (2.7a) has been replaced in (2.7b). All edges that are in conflict with the replacement edge in (2.7b) are drawn in bold.

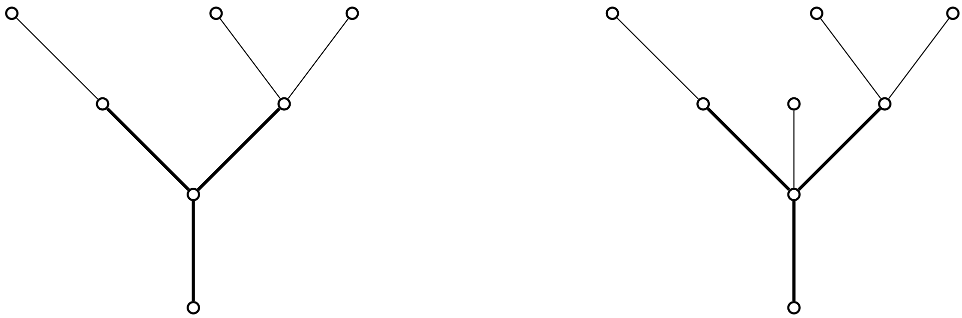
The idea of extension was first introduced in Winter (1995) for the rectilinear Steiner tree problem (see Section 5.4). Later the idea was adopted by Uchoa et al. (2002)[8] for the SPG. The authors achieved strong practical results on a set of (then) large-scale VLSI instances. The next advancement came in Duin (2000), where backtracking was used, together with a number of new reduction criteria for the enumerated trees. Finally, Polzin and Daneshmand (2002) introduced the up-to-now strongest extended reduction techniques, which improved and complemented the previous results. The authors showed that their highly sophisticated algorithm could drastically reduce the size of many benchmark SPG instances, and even allowed for the solution of previously intractable instances.

In the following, we introduce new extended reduction algorithms that (provably) dominate those by Polzin and Daneshmand (2002).

2.4.1 The framework

For a tree Y in G, let $L(Y) \subseteq V(Y)$ be the set of its leaves. We start with several definitions from Polzin and Daneshmand (2002). Let Y, Y' be trees with $Y' \subseteq Y$. The *linking set* between Y and Y' is the set of all vertices $v \in V(Y')$ such that there is a path $Q \subseteq Y$ from v to a leaf of Y with $V(Q) \cap V(Y') = \{v\}$. Note that Q can consist of a single vertex. Y' is *peripherally contained* in Y if the linking set between Y and Y' is $L(Y')$. Figure 2.8 exemplifies this concept. To motivate those definitions, consider a path Q without inner terminals between vertices v and w. For Q to not be peripherally contained in a minimum Steiner tree it is sufficient that $s(v, w)$ is smaller than the weight of Q. However, this condition is not sufficient to show that Q is not contained in a minimum Steiner tree. However, if Q is indeed contained in a minimum Steiner tree, at least one of its inner vertices needs to be of degree greater than 2 in this tree. Thus, we can exploit this observation to enumerate extensions of

[8] The article was published after Duin (2000), but had been available as a preprint already three years earlier.



(a) Peripherally contained tree (b) Not peripherally contained tree

Figure 2.8: Illustration of peripherally inclusion. The bold subtree is peripherally contained in the entire tree in Figure 2.8a, but not in Figure 2.8b.

Q from those inner vertices and attempt to rule those extensions out. Such kind of deductions are used in extended reduction techniques.

For any $P \subseteq V(Y)$ with $|P| > 1$ let Y_P be the union of the (unique) paths between any $v, w \in P$ in Y. Note that Y_P is a tree, and that $Y_P \subseteq Y$ holds. P is called *pruning set* if it contains the linking set between Y_P and Y. Additionally, we will use the following new definition: P is called *strict pruning set* if it is equal to the linking set between Y_P and Y. Figure 2.9 provides an example of pruning and strict pruning sets. One readily verifies the following property of pruning sets.

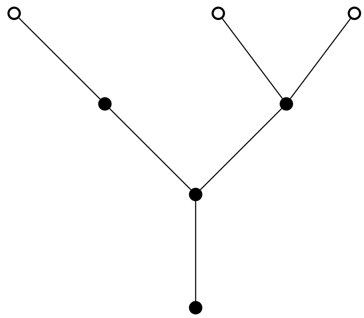
Observation 2.26. *Let Y be a tree, and let $Y' \subseteq Y$ be a tree that is peripherally contained in Y. Further, let $P \subseteq V(Y')$. If P is a pruning set for Y', then P is also a pruning set for Y. If P is a strict pruning set for Y', then P is also a strict pruning set for Y.*

Additionally, we define a stronger, and new, inclusion concept. Consider a tree $Y \subseteq G$, and a subtree Y'. Let P be a pruning set for Y'. We say that Y' is *P-peripherally contained* in Y if P is a pruning set for Y. Now let P be a strict pruning set for Y'. We say that Y' is *strictly P-peripherally contained* in Y if P is a strict pruning set for Y. From Observation 2.26 one obtains the following important property.

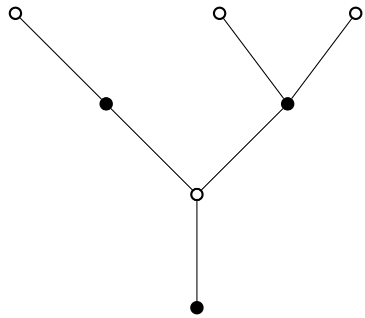
Observation 2.27. *Let $Y \subseteq G$ be a tree, let $Y' \subseteq Y$ be a subtree, and let P be a pruning set for Y'. If Y' is peripherally contained in Y, then Y' is also P-peripherally contained in Y.*

In fact, we will use the contraposition of the observation: If Y' is not P-peripherally contained in Y, then Y' is not peripherally contained in Y. Note that an equivalent property holds for strict pruning sets.

Given a tree Y and a set $E' \subseteq E$, we write with a slight abuse of notation $Y + E'$ for the subgraph with the edge set $E(Y) \cup E'$. Algorithm 2.1 shows a high level



(a) Pruning set



(b) Strict pruning set

Figure 2.9: Illustration of pruning and strict pruning sets. The filled vertices in Figure 2.9a form a (non-strict) pruning set, whereas the filled vertices in Figure 2.9b constitute a strict pruning set.

description of the extended reduction framework used in this thesis. The framework is similar to the one introduced in Polzin and Daneshmand (2002), but more general.[9] Note that the algorithm is recursive.

A possible input for Algorithm 2.1 is an SPG instance together with a single edge. If the algorithm returns *true*, the edge can be deleted. Besides ExtensionSets, which is described in Algorithm 2.2, the extended reduction framework contains the following subroutines:

- RuledOut(I, Y, P) is given an SPG $I = (G, T, c)$, a tree $Y \subseteq G$, and a pruning set P for Y such that $V(Y_P) \cap T \subseteq L(Y_P)$. The routine returns *true* if Y is shown to not be P-peripherally contained in any minimum Steiner tree. Otherwise, the routine returns $false$.
- RuledOutStrict(I, Y, P) is given an SPG $I = (G, T, c)$, a tree $Y \subseteq G$, and a strict pruning set P for Y such that $V(Y_P) \cap T \subseteq L(Y_P)$. The routine returns *true* if Y is shown to not be strictly P-peripherally contained in any minimum Steiner tree. Otherwise, the routine returns $false$.
- StrictPruningSets(I, Y) is given an SPG $I = (G, T, c)$, a tree $Y \subseteq G$. It returns a subset of all strict pruning sets for Y. A typical strict pruning set is $L(Y)$.
- Truncate(I, Y) is given an SPG $I = (G, T, c)$, and a tree $Y \subseteq G$. The routine returns *true*, if no further extensions of Y should be performed; otherwise the routine returns $false$.

[9] We note, however, that the framework presented in Polzin and Daneshmand (2002) is (slightly) erroneous. E.g., in their equivalent to Step 4 of Algorithm 2.2, their method only checks whether the extended tree $Y + \{e\}$ is not peripherally contained with linking set $L(Y) \cup \{w\}$ in a minimum Steiner tree. However, this check does not rule out the extension of the current tree Y via the single edge $\{v, w\}$.

– Promising(I, Y, v) is given an SPG $I = (G, T, c)$, a tree $Y \subseteq G$, and a vertex $v \in L(Y)$. The routine returns *true* if further extensions of Y from v should be performed; otherwise the routine returns $false$.

The usage of P-peripheral inclusion in RuledOut might appear somewhat awkward, but is necessary for ruling-out not only trees (as in line 2 of Algorithm 2.1), but also all possible extensions via a single edge (as in line 4 of Algorithm 2.2).

Algorithm 2.1: Extended-RuledOut

Data: SPG instance $I = (G, T, c)$, tree Y with $Y \cap T \subseteq L(Y)$
Result: *true* if Y is shown to not be peripherally contained in any minimum Steiner tree; $false$ otherwise

```
1  foreach P ∈ StrictPruningSets(I, Y) do
2      if RuledOutStrict(I, Y, P) then
3          return true
4  if Truncate(I, Y) then
5      return false
6  foreach v ∈ L(Y) do
7      if v ∈ T or not Promising(I, Y, v) then
8          continue
9      success := true
10     foreach E′ ∈ ExtensionSets(I, Y, v) do
11         if not Extended-RuledOut(I, Y + E′) then
12             success := false
13     if success then
14         return true
15 return false
```

In Lines 1-3 of Algorithm 2.1, we try to peripherally rule-out tree Y. If that is not possible, we try to recursively extend Y in Lines 6-14. Since (given positive edge weights) no minimum Steiner tree has a non-terminal leaf, we can extend from any of the non-terminal leaves of Y. Note that ruling-out all extensions along one single leaf is sufficient to rule-out Y. The correctness of Extended-RuledOut can be proven by induction (under the assumption that the subroutines are correct). We also remark that it is under certain conditions possible to replace the condition *not peripherally contained in any minimum Steiner tree* by the condition *not peripherally contained in at least one minimum Steiner tree*. See also the discussion following Theorem 2.28.

Although the extended reduction framework shown in Algorithm 2.1 looks simple, an efficient realization is highly intricate. Not least, because the interaction of many different algorithmic components needs to be taken into account. Also, the re-use of intermediate results obtained during the tree extension (such as bottleneck Steiner

Algorithm 2.2: ExtensionSets

Data: SPG instance $I = (G, T, c)$, tree Y, vertex $v \in V(Y)$
Result: Set $\Gamma \subseteq \mathcal{P}(\delta(v))$ such that for all non-empty $\gamma \in \mathcal{P}(\delta(v)) \setminus \Gamma$, the tree $Y + \rho$ is not peripherally contained in any minimum Steiner tree.

$Q := \emptyset$
$R := \emptyset$
foreach $e := \{v, w\} \in \delta(v) \setminus E(Y)$ **do**
  if RuledOut$(I, Y + \{e\}, L(Y) \cup \{w\})$ **then**
    continue
  if RuledOutStrict$(I, Y + \{e\}, L(Y) \cup \{w\})$ **then**
    $R := R \cup \{e\}$
    continue
  $Q := Q \cup \{e\}$
return $(\mathcal{P}(Q) \setminus \emptyset) \cup R$

distances) is non-trivial. Indeed, the implementation of extended reduction techniques for this thesis encompasses more than 20 000 lines of C code[10], and includes many further algorithmic ideas. In the following, we concentrate on mathematical descriptions of the subroutines for ruling-out enumerated trees. Implementation details of several key components—including nitty-gritty issues such as CPU cache-efficiency—are given in Section 6.2.3.

2.4.2 Reduction criteria

In this section, we introduce several elimination criteria used within RuledOut and RuledOutStrict. In fact, both of these routines consist of several subalgorithms that check different criteria for eliminating the given tree. Note that any criterion that is valid for RuledOut is also valid for RuledOutStrict. We also note that several of the criteria in this section are similar to results from Polzin (2003); Polzin and Daneshmand (2002), but are all stronger. Throughout this section we consider a graph $G = (V, E)$ and an SPG instance $I = (G, T, c)$.

Consider a tree $Y \subseteq G$, and a pruning set P for Y such that $V(Y_P) \cap T \subseteq L(Y_P)$. For each $p \in P$ let $\overline{Y}_p \subset Y$ such that $V(\overline{Y}_p)$ is exactly the set of vertices $v \in V(Y)$ that satisfy the following: For any $q \in P \setminus \{p\}$ the (unique) path in Y from v to q contains p. Note that when removing $E(Y_P)$ from Y, each non-trivial connected component equals one $\overline{Y}_p$. Further, note that $p \in V(\overline{Y}_p)$ for all $p \in P$. Let $G_{Y,P} = (V_{Y,P}, E_{Y,P})$ be the graph obtained from $G = (V, E)$ by contracting for each $p \in P$ the subtree $\overline{Y}_p$ into p. For any parallel edges, we keep only one of minimum weight. We identify the contracted vertices $V(\overline{Y}_p)$ with the original vertex p. Overall, we thus have $V_{Y,P} \subseteq V$.

[10] We note, however, that this code count includes comments, as well as various correctness tests (which are only executed in debug mode). We also note that the same reductions could be achieved with (much) less than half of the code size, but at the expense of an increased run time.

Let $c_{Y,P}$ be the edge weights on $G_{Y,P}$ derived from c. Let

$$T_{Y,P} := \big(T \cap V_{Y,P}\big) \cup \{p \in P \mid T \cap V(\overline{Y}_p) \neq \emptyset\}. \tag{2.71}$$

Finally, let $s_{Y,P}$ be the bottleneck Steiner distance on $(G_{Y,P}, T_{Y,P}, c_{Y,P})$. With these definitions at hand, we are able to formulate a reduction criterion that generalizes a number of results from the literature. See Hwang et al. (1992); Polzin (2003) for similar, but weaker, conditions.

Theorem 2.28. *Let $Y \subseteq G$ be a tree, and let P be a pruning set for Y such that $V(Y_P) \cap T \subseteq L(Y_P)$. Let $I_{Y,P}$ be the SPG on the distance network $D_{G_{Y,P}}\big(V_{Y,P}, s_{Y,P}\big)$ with terminal set P. If the weight of a minimum Steiner tree for $I_{Y,P}$ is smaller than $c(E(Y_P))$, then Y is not P-peripherally contained in any minimum Steiner tree for I.*

Proof. Let S be a (not necessarily minimum) Steiner tree for I such that Y is P-peripherally contained in S. Let $S_{Y,P}$ be a minimum Steiner tree for $I_{Y,P}$. Let $\tilde{S} \subset G$ be the forest defined as follows:

$$V(\tilde{S}) := (V(S) \setminus V(Y_P)) \cup V(S_{Y,P}), \tag{2.72}$$

$$E(\tilde{S}) := E(S) \setminus E(Y_P). \tag{2.73}$$

Let $\tilde{\mathcal{C}}$ be the set of connected components of $\tilde{S}$. Further, let $f : V \to \tilde{\mathcal{C}} \cup \{\emptyset\}$ such that $f(v) = \tilde{C}$ if $v \in V(\tilde{C})$ for a $\tilde{C} \in \tilde{\mathcal{C}}$, and $f(v) = \emptyset$ otherwise. Note that each $\tilde{C} \in \tilde{\mathcal{C}}$ contains at least one vertex of P, and thus also at least one vertex of $S_{Y,P}$. Also, $f(v) \neq \emptyset$ for all $v \in V(S_{Y,P})$. Further, note that for each of the contracted subtrees $\overline{Y}_p$ there is a $\tilde{C} \in \tilde{\mathcal{C}}$ with $\overline{Y}_p \subseteq \tilde{C}$. In the following, we will iteratively connect all the components in $\tilde{\mathcal{C}}$.

While $|\tilde{\mathcal{C}}| > 1$ proceed as follows. Choose a $(v, w) \in E(S_{Y,P})$ with $f(v) \neq f(w)$ such that $s_{Y,P}(v, w)$ is minimized. Let W be a (v, w)-walk in $G_{Y,P}$ corresponding to $s_{Y,P}(v, w)$. Because of $f(v) \neq f(w)$, there is at least one subwalk $Q = W(q, r)$ of W such that $f(q), f(r) \neq \emptyset$, $f(q) \neq f(r)$, and $f(u) = \emptyset$ for all $u \in V(Q) \setminus \{q, r\}$. Note that $c(E(Q)) \leqslant s_{Y,P}(v, w)$, because $f(t) \neq \emptyset$ for all $t \in T$. As long as such a path Q exists, proceed as follows. Add Q to $\tilde{S}$, and remove from $E(S_{Y,P})$ an (arbitrary) edge of the path between $f(q)$ and $f(r)$ in $S_{Y,P}$. Also, update $\tilde{\mathcal{C}}$ and f. Note that the weight of the removed edge (with respect to $s_{Y,P}$) is at most $s_{Y,P}(q, r)$.

Once $|\tilde{\mathcal{C}}| = 1$, one notes that the summed up weight of all newly inserted paths (with respect to c) does not exceed the weight of $S_{Y,P}$ (with respect to $s_{Y,P}$). Because the weight of $S_{Y,P}$ is smaller than $c(E(Y_P))$, we obtain from the construction of $\tilde{S}$ that

$$c(E(\tilde{S})) < c(E(S)), \tag{2.74}$$

which concludes the proof. □

In practice, one does not need to explicitly form $G_{Y,P}$. Instead, one can use the (original) bottleneck Steiner distances between the connected components of the graph induced by $E(Y) \setminus E(Y_P)$. Note that one can also extend Theorem 2.28 to the case of equality if at least one vertex of Y_P is not contained in any of the paths corresponding to the s values used for edges of $S_{Y,P}$. However, in the context of extended reduction techniques one needs to be careful to not discard all of several equivalent extensions. We omit the quite technical details, but merely note that allowing for equality (and adding suitable checks) can have a significant impact for some instances.

In practice, computing a minimum Steiner tree (or even an approximation) on $D_{G_{Y,P}}\big(V_{Y,P}, s_{Y,P}\big)$ is often too expensive. In such cases, the following corollary provides a strong alternative.

Corollary 2.29. *Let Y, P as in Theorem 2.28. Let (P', P'') be a partition of P. Let F' be a minimum spanning tree on $D_{G_{Y,P}}\big(P', s_{Y,P}\big)$, and let z' be the weight of F'. Let F'' be a minimum spanning tree in $D_{G_{Y,P}}\big(T_{Y,P}, s_{Y,P}\big)$. Write $\{e_1^{F''}, e_2^{F''}, ..., e_{|T_{Y,P}|-1}^{F''}\} := E_{Y,P}(F'')$ such that $s_{Y,P}(e_i^{F''}) \geqslant s_{Y,P}(e_j^{F''})$ for $i < j$. Define*

$$z'' := \sum_{i=1}^{|P''|} s_{Y,P}(e_i^{F''}). \tag{2.75}$$

If $z' + z'' < c(E(Y_P))$, then Y is not P-peripherally contained in any minimum Steiner tree for I.

Proof. First, note that if $P'' = \emptyset$, then the corollary follows directly from Theorem 2.28, because z' is a lower bound on the weight of a minimum Steiner tree in $I_{Y,P}$. Thus, we assume $P'' \neq \emptyset$ in the following.

Suppose there is a minimum Steiner tree S for I such that Y is P-peripherally contained in S. Define $\tilde{S}$ as in the proof of Theorem 2.28. Further, proceed as in the proof of Theorem 2.28 to reconnect all connected components of $\tilde{S}$ that contain a vertex from P'. As a result, $\tilde{S}$ has at most $|P''| + 1$ connected components. Because S is assumed to be optimal, each connected component of $\tilde{S}$ contains at least one terminal. Thus, we can reconnect the remaining connected components similarly to Theorem 2.28, by using paths corresponding to edges of F''. We need to add at most $|P''|$ such paths. Overall, we have increased the weight of $\tilde{S}$ by at most $z' + z''$. From $z' + z'' < c(E(Y_P))$ we obtain that

$$c(E(\tilde{S})) < c(E(S)), \tag{2.76}$$

which contradicts the optimality of S. □

As for Theorem 2.28, the contractions in Corollary 2.29 should only be performed implicitly in practice. Furthermore, one requires a careful implementation to avoid a recomputation from scratch of the two minimum spanning trees in Corollary 2.29 for each enumerated tree in Algorithm 2.1.

Next, let $Y \subseteq G$ be a tree with pruning set P, and let $v, w \in V(Y)$ and let Q be the path between v, w in Y. We define a *pruned tree bottleneck* between v and w as a subpath $Q(a, b)$ of Q that satisfies $|\delta_Y(u)| = 2$ and $u \notin P$ for all $u \in V(Q(a, b)) \setminus \{a, b\}$,

$V(Q(a,b)) \cap T \subseteq \{a,b\}$, and maximizes $c(V(Q(a,b)))$. The weight $c(V(Q(a,b)))$ of such a pruned tree bottleneck is denoted by $b_{Y,P}(v,w)$. Using this definition and the implied bottleneck Steiner distance, we obtain the following result.

Proposition 2.30. *Let Y be a tree, let P be a pruning set for Y, and let $v, w \in V(Y)$. If $s_p(v,w) < b_{Y,P}(v,w)$, then Y is not P-peripherally contained in any minimum Steiner tree.*

The proposition can be proven in a similar way as Theorem 2.11 (and is indeed a generalization of the latter).

Another criterion can be devised by using the reduced costs of the bidirected cut formulation (*BDCut*). Let $D = (V, A)$ be the bidirected equivalent of G, and let $r \in T$ be the root for *BDCut*. Consider a dual solution to *BDCut*, with reduced costs $\tilde{c}$, and with objective value $\tilde{L}$. Further, for any $v, w \in V$, let $\tilde{d}(v,w)$ be the length for a shortest, directed path from v to w in A with respect to the reduced costs. From the observation that an optimal Steiner arborescence cannot contain any cycles, we obtain the following result with standard linear programming arguments:

Proposition 2.31. *Let Y be a tree. Let $P = \{p_1, ..., p_k\}$ be a strict pruning set for Y such that there is a $k' \leqslant k$ with $p_i \in T$ if and only if $i > k'$. Further, assume that $V(Y_P) \cap T \subseteq L(Y_P)$, and $|P| < |T|$. The weight of any Steiner tree that strictly P-peripherally contains Y is at least*

$$\tilde{L} + \min_{i \in \{1,\dots,k\}} \max_{\{t_1,..t_{i-1},t_{i+1},\dots,t_{k'}\} \subseteq T \setminus V(Y_P)} \{\tilde{d}(r,p_i) + \sum_{j \leqslant k', j \neq i} \tilde{d}(p_j, t_j)\} \tag{2.77}$$

Given an upper bound on the cost of a minimum Steiner tree, this proposition can be used in the RuleOutStrict routine. In practice, we only use a lower bound on the max subterm in (2.77).

Finally, another important reduction criteria is constituted by edge conflicts—this result follows directly from Proposition 2.23.

Corollary 2.32. *Let $I^{(k)}$ be an SPG obtained from performing a series of k valid reductions on an SPG I. Let $Y \subseteq G^{(k)}$ be a tree, and let P a pruning set for Y. If there are distinct edges $e_1, e_2 \in E^{(k)}(Y)$ such that $\Lambda^{(k)}(e_1) \cap \Lambda^{(k)}(e_2) \neq \emptyset$, then Y is not P-peripherally contained in any minimum Steiner tree.*

2.5 Primal heuristics

In the mathematical world, heuristics might be characterized as the Cinderellas[11] of discrete optimization. Both are mostly kept out of view (in the case of heuristics because of the somewhat embarrassing lack of theoretical performance guarantees), but they do much of the real work behind the scenes. Thus, heuristics are very popular in the more practically minded operations research community. In the case of the

[11] according to Wikipedia, Cinderella is a *folk tale about unjust oppression and triumphant reward*

SPG, the number of articles concerned with primal heuristics is huge. An overview of articles up to 1992 can be found in Hwang et al. (1992). Newer developments are referenced in Duin and Voss (1997); Pajor et al. (2017); Polzin and Daneshmand (2001b); Ribeiro et al. (2001). See also the recent survey by Ljubic (2020).

Obtaining Steiner trees of small weight is often of independent relevance, but the corresponding upper bounds on the optimal solution value are also highly relevant for exact SPG solution. For example, such upper bounds are indispensable for the bound-based reductions described in Section 2.3.2. Additionally, tight upper bounds are highly important for ruling-out subproblems during branch-and-bound.

We use three (well-known) local-search heuristics in our implementation: *Vertex-insertion*, *key-path exchange*, and *key-vertex elimination*. Given a Steiner tree S, vertex-insertion computes for each vertex $v \notin V(S)$ that is adjacent to S an MST on the graph induced by $V(S) \cup \{v\}$. This MST is also a Steiner tree. *Key-vertices* of a Steiner tree S are all $v \in V(S)$ with $v \in T$ or $|\delta_S(v)| \geqslant 3$. A *key-path* is a path in S with a key-vertex at both endpoints, but without any intermediary key-vertices. Key-path exchange attempts to replace key-paths in S by other paths of smaller cost between the same endpoints. Similarly, for key-vertex elimination in each step a non-terminal key-vertex and all adjoining key-paths are removed from S, and an attempt is made to reconnect the resulting subtrees at lower cost. Our implementation of these heuristics follows Uchoa and Werneck (2010).

In the following, we describe several so-called *construction heuristics*—heuristics that build feasible solutions from scratch.

2.5.1 Shortest path heuristic and implications

The *shortest path heuristic* is arguably the best-known primal SPG heuristic. Introduced in 1980 by Takahashi and Matsuyama (1980), it has found its way into various publications, e.g. Hwang et al. (1992); de Aragão and Werneck (2002); Polzin and Daneshmand (2001b); Pajor et al. (2017). The algorithm starts with a tree S consisting of a single vertex and iteratively connects S by a shortest path to a terminal closest to S. The run time of the heuristic is in $O(|T|(m + n \log n))$. As a simple postprocessing step, one can compute a minimum spanning tree on $(V(S), E[S])$ and iteratively remove non-terminal leaves. An efficient implementation is given in de Aragão and Werneck (2002). Obviously, the solution provided by the heuristic depends on the starting point. Since the heuristic is empirically very fast, it is therefore usually run from several vertices. This section shows how to use the implication concept introduced in Section 2.3.1 to (empirically) improve the algorithm.

Let $v_0 \in V$, and initially set $S := \{v_0\}$. Define a distance array $\tilde{d}$ and a predecessor array *pred* by $\tilde{d}[u] := \infty$, $pred[u] := null$ for all $u \in V \setminus \{v_0\}$, and $\tilde{d}[v_0] := 0$, $pred[v_0] := v_0$. Define for all $v \in V \setminus T$:

$$\tilde{p}(v) := \max\left\{0, \sup\left\{\bar{b}(e) - c(e) \mid e = \{v, w\} \in \delta(v), w \in T \setminus V(S)\right\}\right\}. \tag{2.78}$$

For all $v \in T$ set $\tilde{p}(v) := 0$. Essentially, (2.78) is a weaker version of the implied profit from Section 2.3.1. Finally, set $Q := \{v_0\}$.

While $Q \neq \emptyset$ let $v := \arg\min_{u \in Q} \tilde{d}[u]$. If $v \in T$, add the path P from v to S, marked by the predecessor array, to S, add $V(P)$ to Q, and set $\tilde{d}[u] := 0$ for all $u \in V(P)$. Furthermore, update (2.78). For all $\{v, w\} \in \delta(v)$ proceed as follows. If

$$\tilde{d}[v] + c(\{v, w\}) - \min \left\{ c(\{v, w\}), \tilde{p}(v), \tilde{d}[v] \right\} < \tilde{d}[w], \tag{2.79}$$

then set $\tilde{d}[w]$ to the left hand side of (2.79), and add w to Q. Further, set $pred[w] := v$.

Note that (2.79) provides a bias for paths computed by the heuristic to include vertices of implied profit. In this way, the distance associated with a path also reflects the cost needed to connect additional terminals later on. Note that the minimum spanning tree computed during postprocessing will always contain the edge associated with each vertex of positive implied profit contained in S. For performance reasons, we use in (2.78) instead of $\overline{b}(e)$ for $e = \{v, w\}, w \in T$, the value $\min_{e' \in \delta(w) \setminus \{e\}} c(e')$.

2.5.2 Reduction based heuristics

This section introduces heuristics that make heavy use of SPG reduction techniques. We with start with two heuristics from Polzin and Daneshmand (2001b), and end this section with a new heuristic

Prune

While the following heuristic is originally based on Voronoi diagram reductions (Polzin and Daneshmand, 2001b), we instead describe a modification of the heuristic based on the (stronger) terminal-regions decomposition introduced in Section 2.3.2. Recall that for the terminal-regions decomposition test an upper bound is provided by the weight of a given solution. In the *prune* heuristic, the bound is chosen such that a predefined number of edges is eliminated. Thereupon, all exact reduction methods are executed on the reduced problem, motivated by the assumption that the (possibly inexact) eliminations performed by the bound-based method will allow for further (exact) reductions. This procedure is repeated several times. On the final reduced problem, a solution is computed by using the above shortest path heuristic. This solution is retransformed to a solution to the original SPG instance. To avoid infeasibility, initially a feasible solution is computed (by using the shortest-path heuristic introduced above) of which no vertices or edges are allowed to be deleted by the (inexact) bound-based method.

Ascend-and-prune

Ascend-and-prune is borne from the combination of the prune heuristic and dual-ascent. Let I be an SPG instance, and I' the equivalent bidirected SAP. The ascend-and-prune heuristic computes a solution on the subproblem $\tilde{I}$ constituted by the (undirected) edges of I corresponding to zero-reduced-cost paths in I' from the root to all additional terminals. This solution on $\tilde{I}$ is computed by the prune heuristic. The ascend-and-prune heuristic is motivated by the assumption that notable similarities exist between an optimal (or near-optimal) Steiner tree and the LP solution corresponding to the reduced costs provided by dual-ascent. Finally, we note that the idea of searching for a solution on the subproblem induced by dual-ascent can already be found in Wong (1984).

Recombine-and-reduce

Consider an SPG instance $I = (V, E, T, c)$ and let $\mathcal{L}$ with $|\mathcal{L}| \geqslant 2$ be a set of feasible solutions to I. The *recombine-and-reduce* heuristic tries to compute new Steiner trees out of the solutions in $\mathcal{L}$. The key component of the heuristic is an operation that we will refer to as *n-merging*. Given a solution $S \in \mathcal{L}$, and an $n \in \mathbb{N}$ with $n \geqslant 2$, define the n-merging operation as follows: Choose $\mathcal{L}' \subseteq \mathcal{L} \setminus \{S\}$ such that $|\mathcal{L}'| = n - 1$ (in a pseudo-random way). Let $G_S := (V_S, E_S) := \bigcup_{S' \in \mathcal{L}'} S' \cup S$. If G_S consists of several biconnected components, we perform the following procedure for each of these biconnected components individually. In the following, we assume that G_S is biconnected. Let $I_S := (V_S, E_S, T, c \restriction_{E_S})$. Applying the reduction techniques introduced in this thesis to I_S, we obtain a new SPG I'_S. During the reduction process one often obtains feasible solutions to I_S (since several primal heuristics are performed). Let $\tilde{S}$ be the best such solution, if existent.

Another solution is computed on I_S after perturbing the edge costs $c \restriction_{E_S}$ as follows. Consider for each edge $e \in E_S$ the ancestor set $\Pi(e) \subseteq E$ defined in Section 2.3.4. Define

$$\alpha(e) := \frac{\sum_{S' \in \mathcal{L}'} |\Pi(e) \cap E(S')|}{|\Pi(e)|}. \tag{2.80}$$

Next, the cost of each edge $e \in E_S$ is multiplied by a (pseudo-random) number that is anti-proportional to $\alpha(e)$—i.e., the number increases as $\alpha(e)$ decreases. On this perturbed SPG instance, a heuristic solution is computed and retransformed to the original solution space. This solution is compared with the solution obtained during the reduction process, and the better of the two is retained.

The recombine-and-reduce heuristic is clustered around the n-merging operation: Given a new solution S, in each run we consecutively perform several n-merging operations with varying n. When a solution S' is generated during an n-merging such that $c(E(S')) < c(E(S))$, we set $S := S'$ and add S' to $\mathcal{L}$. Moreover, in this case a new run is started after the conclusion of the current one. The total number of runs is limited.

We note that a forerunner of this heuristic was already included in a SCIP-Jack version that predates this thesis. Moreover, this older version is itself a generalization of an approach described in Pajor et al. (2017).

2.6 Solving to optimality

For solving SPG instances to optimality we rely on two methods: dynamic programming and branch-and-cut. We note, however, that in this thesis the latter is the far more important method. For both dynamic programming and branch-and-cut we make heavy use of the algorithms introduced so far.

2.6.1 Combining extended reductions and dynamic programming

The well-known exact SPG algorithm by Dreyfus and Wagner (1971) exploits the fact that any optimal Steiner tree S for an SPG (G, T, c) can be split at any $v \in V(S)$ into two non-empty trees S_1 and S_2 such that $T_1 := V(S_1) \cap T \neq \emptyset$, $T_2 := V(S_2) \cap T \neq \emptyset$, and:

1. $T_1 \cap T_2 \subseteq \{v\}$ and $T_1 \cup T_2 = T$,
2. S_1 is optimal for $(G, T_1 \cup \{v\}, c)$, and S_2 is optimal for $(G, T_2 \cup \{v\}, c)$.

This observation can be exploited to find an optimal Steiner tree in a dynamic programming fashion—by recursively computing an optimal Steiner tree for any terminal set $T' \cup \{v\}$ for any $T' \subseteq T$ and any $v \in V \setminus T'$. In Erickson et al. (1987) a slightly improved version of the algorithm by Dreyfus and Wagner (1971) is given that achieves a run time of $O(3^{|T|}n + 2^{|T|}(m + n \log n))$. The authors use the fact that given an optimal Steiner tree for a terminal set T', optimal Steiner trees for terminal sets $T' \cup \{v\}$ for all $v \in V \setminus T'$ can be computed by just one execution of Dijkstra's algorithm.

There have been notable efforts during the last years to make the algorithms by Dreyfus and Wagner (1971) and Erickson et al. (1987) competitive in practice, see e.g. Hougardy et al. (2017); Iwata and Shigemura (2019) for prominent examples. However, these implementations can usually not match the state of the art solver from Polzin (2003); Vahdati Daneshmand (2004) even for instances with few terminals. A critical component of practical realizations of the above dynamic programming scheme are so-called *pruning techniques*, which allow one to discard optimal Steiner trees for certain terminal sets. In this way, often a dramatic speed-up can be obtained as compared to naive implementations, see e.g. Iwata and Shigemura (2019). In this context, one notices that most of the reduction criteria introduced for extended reduction methods in Section 2.4.2 can also be used for pruning. The resulting methods are considerably stronger than the pruning techniques employed in the literature so far. Our dynamic programming implementation combines these techniques with the node-separator concept from Iwata and Shigemura (2019). Unfortunately, however, our implementation is only competitive with branch-and-cut for instances with less than 20 terminals—and these instances are usually solved quickly by either approach. Still, in the context of decomposition techniques, where we often obtain many small sub-problems, the dynamic programming algorithm has turned out to be useful.

We also tentatively implemented FPT algorithms based on tree-width, see e.g. Chimani et al. (2012), and on the *border* concept by Polzin and Daneshmand (2006). However, we could only achieve performance gains on very few instances compared to our default approach. Furthermore, we observed an overall performance degradation even when we limited the execution of these algorithms to instances with small tree-width.

2.6.2 Branch-and-cut

This section describes how to assemble the various techniques introduced so far within an (exact) branch-and-cut algorithm. All algorithms discussed in this chapter so far are part of the following three classes, which form the main pillars of our exact algorithm:

- reduction techniques,
- heuristics (primal and dual), and
- IP formulation and cutting planes.

The dynamic programming algorithm introduced in the previous section can be seen as an exception to this classification. However, this algorithm is not often applied (and usually only for sub-problems).

Notably, the three algorithmic classes are deeply intertwined. For example, reduction methods are crucial for the success of prune, ascend-and-prune, and recombine-and-reduce, while the quality of the primal bound obtained by these heuristics determines the effectiveness of the bound-based reduction methods. Additionally, reduced problems usually show a smaller integrality gap for the IP formulation, and require less time for solving the LP-relaxation. In turn, the reduced-costs from the LP-relaxation can be used for further reductions.

In the following, we list the main components of the branch-and-cut framework that is used for exact SPG solution in this thesis. Additional technical details of the branch-and-cut procedure—both for SPG and the subsequently discussed related problems—can be found in Section 6.2.

Presolving For presolving, the reduction methods described in this thesis are executed iteratively within a loop. This loop is reiterated as long as a predefined percentage of edges has been eliminated during the previous round. Naturally, the, empirically, faster reduction methods are performed first. Empirically, the order of the reduction techniques only has a small impact on the strength of the presolving. Indeed, for the classic SPG reduction techniques from Duin and Volgenant (1989b) this behavior can also be theoretically verified, see Kingston and Sheppard (2003). For computing reduced-costs during presolving, we employ dual-ascent.

Domain propagation During branch-and-bound we use the reduced-costs from the LP-relaxations for fixing arcs of the $BDCut$ formulation to 0. To this end, we use the path-based criterion used in Section 2.3.2. Additionally, whenever a predefined percentage of all arcs have been newly fixed during the branch-and-bound procedure, further reduction techniques are applied as follows. Let $I = (V, E, T, c)$ be the considered SPG instance. All edges $\{v, w\} \in E$ such that both $y((v, w))$ and $y((w, v))$ have been fixed to 0 are removed from I. Edges $\{v, w\} \in E$ such that either $y((v, w))$ or $y((w, v))$ has been fixed to 1 are contracted. Additionally, I is modified as to reflect the branching history, see paragraph *Branching* below. Finally, several of the reduction methods described in this thesis are performed on I, and the changes are retranslated into arc fixing. While the deletion of edges can be directly translated into variable fixings, the situation is more complicated for vertex replacements. Therefore, other authors, e.g. Polzin and Daneshmand (2001b), use only edge deleting reductions for domain propagation. However, by using the ancestor concept introduced in

Section 2.3.4, one can readily devise a criterion to employ all reduction operations used in this thesis for domain propagation.

Proposition 2.33. *Let $I = (V, E, T, c)$ be an SPG and let $I^{(k)}$ be the SPG obtained from performing a series of k valid reductions on I. Define an SPG instance $I' = (V', E', T', c')$ as follows:*

$$E' := \bigcup_{e \in E^{(k)}} \Pi^{(k)}(e) \cup \Pi^{(k)}_{FIX},$$

$V' = \{v \in V \mid \exists e \in E', v \in e\}$, $T' := T$, and $c' := c \restriction_{E'}$. Any optimal solution to I' is an optimal solution to I.

We do not provide a proof, but note that the validity of the proposition follows from Observation 2.22.

Decomposition It is well-known that the biconnected components of the graph underlying an SPG instance can be solved separately. Given the super-linear run time of most algorithms that we employ, such a decomposition can lead to significant speed-ups. While SPG instances usually do not have articulation points in their original form, this property sometimes changes after the application of reduction techniques. Therefore, we use decomposition into biconnected components both during presolving and during branch-and-cut.

Primal heuristics We try to retain the best solution found during presolving, to provide it as an initial primal solution. Additionally, we use the reduced costs obtained during the creation of the initial cutting planes (as detailed below) for applying the ascend-and-prune heuristic before the branch-and-bound procedure starts. During branch-and-bound, we periodically employ our shortest-path heuristic, and the recombine-and-reduce heuristic. Additionally, we use the local heuristics vertex-insertion, key-path exchange, and key-vertex elimination to improve high quality primal solutions. We furthermore use the solution to the current LP-relaxation to guide the shortest-path heuristic—an idea already utilized by other authors, e.g. Koch and Martin (1998). We alter the arc weight as follows: Given an LP solution $y \in \mathbb{Q}^A$ to $BDCut_{FB}$, the (directed version of the) shortest-path heuristic is called with the arc weights $(1 - y(a)) \cdot c(a)$ for all $a \in A$. In this way, arcs with a high LP solution value are more likely to be selected by the heuristic.

Separation After presolving, SCIP-Jack runs the dual-ascent heuristic to select a set of constraints from the *BDCut* formulation to be included into the initial LP. Additionally, we use all $0-1$ constraints. We separate the remaining constraints of the *BDCut* formulation by using a newly implemented maximum-flow algorithm, see Section 6.2.4 for more details. We also separate the flow-balance constraints. Additionally, we use another class of cuts from Koch and Martin (1998):

$$y\left(\delta^-(v) \setminus (w, v)\right) \geqslant y((v, w)), \qquad \text{for all } (v, w) \in \delta^+(v), v \in V \setminus T. \tag{2.81}$$

Although Polzin and Daneshmand (2001a) show that the constraints (2.81) cannot improve the objective value of the *BDCut* LP-relaxation, the constraints usually lead to a speed-up of the branch-and-cut procedure.

Branching Classic variable branching for the *BDCut* formulation often leads to a badly balanced branch-and-bound tree, since the inclusion of an arc has a far larger impact than its exclusion. Thus, a well-known strategy is to branch on vertices instead, see e.g. Hwang et al. (1992): A selected vertex is made a terminal in one branch-and-bound child node, and is removed in its sibling. Such a change is reflected in the IP formulation by adding one additional constraint. We note, however, that branching is rarely required, due to the various powerful algorithms that we apply before. As such, more than 95 percent of the SPG instances considered in this thesis are solved without any branching.

2.7 Computational results

This section provides computational results for our Steiner tree solver SCIP-Jack on a large collection of SPG instances from the literature. We look at the impact of individual components, and furthermore compare SCIP-Jack with the state of the art in SPG solution. An overview of the test-sets is given in Table 2.1. The second column gives the number of instances per test-sets. The third and fourth columns give the range of nodes and edges per test-set. The fifth column states whether for all instances of the test-set optimal solutions are known. We note that this collection covers almost all established test-sets from the literature—including the SteinLib (Koch et al., 2001), as well as the 11th DIMACS and 3rd PACE Challenge instances—except for very easy ones. More details are given in Section 2.7.3. See Section 1.2.1 for hardware details.

2.7.1 Individual components

We start with computational results for individual components of SCIP-Jack. A common way of demonstrating the impact of a single component within an exact solver is to report the performance change when deactivating this component, see e.g. Achterberg (2007b). With a large number of algorithms being combined in a solver—as is the case in this thesis—the question is on the granularity of this approach. We have observed that several algorithms, such as individual heuristics, are reasonably well compensated by another one, so simply deactivating one of them does not show a large performance impact. However, deactivating two complementary algorithms can already have a much larger impact than just deactivating either of them. Thus, we have decided for a hybrid approach. First, we show the individual impact of the three major building blocks of our SPG solver. Second, we take a more in-depth look at the, arguably, most important of them: reduction techniques. In particular, we will demonstrate the impact of the (newly introduced) implied Steiner bottleneck distance techniques.

Name	#	$\lvert V\rvert$	$\lvert E\rvert$	Status	Description
PACE-A	100	53 - 10393	80-204480	solved	Instances from the 3rd PACE Challenge Track A (few terminals).
PACE-B	100	15 - 36415	35-145635	solved	Instances from the 3rd PACE Challenge Track B (small tree-width).
2R	27	2000	11600	solved	3-D cross grid graphs from SteinLib.
VLSI	116	90 - 36711	135 - 68117	solved	Grid graphs with holes (non-geometric) from VLSI design (Koch and Martin, 1998).
vienna-s	85	1991 - 89596	3176 - 148583	solved	Instances derived from telecommunication network design, see Leitner et al. (2014),
vienna-a	85	160 - 34221	237 - 50301	solved	Presolved versions of the above network design instances.
ES10000	1	27019	39407	solved	Originally rectilinear Steiner tree instances. From SteinLib.
TSPFST	76	89-17127	104-27352	solved	Originally rectilinear Steiner tree instances. From TSPLIB (Reinelt, 1991).
GEO-org	23	42481 - 235686	52552 - 366093	solved	Instances derived from telecommunication network design. From Leitner et al. (2014).
GEO-a	23	7565 - 71184	11521 - 113616	solved	Presolved versions of the above *GEO-org* instances.
Cophag14	21	16 - 15473	23 - 38928	solved	Originally obstacle-avoiding rectilinear instances. From 11th DIMACS Challenge.
WRP4	63	110 - 1898	188 - 3060	solved	Instances derived from wire-routing processing problems (Hegde et al., 2014).
WRP3	62	84 - 3168	149 - 6220	solved	
LIN	37	53 - 38418	80 - 71657	solved	Grid graphs with holes (non-geometric) from VLSI design. From SteinLib.
SP	8	6 - 3997	9 - 10278	solved	Constructed hard instances; combination of odd-wheels and odd cycles. From SteinLib.
PUC	50	64 - 4096	192 - 28512	unsolved	Constructed hard instances; hypercubes, and bipartite graphs (Rosseti et al., 2004).
PUCN	13	64 - 4096	192 - 28512	unsolved	Instances with uniform edge weights derived from PUC. From 11th DIMACS Challenge.
ES-R50	15	78754 - 79505	95363 - 97373	unsolved	Originally Euclidean Steiner tree instances with 50 thousand points (Juhl et al., 2018).

Table 2.1: Details on SPG benchmark sets.

The big picture

As elaborated in Section 2.6, the three main building blocks of our SPG solver are reduction techniques, heuristics, and the separation routine (for *BDCut*). Unfortunately, deactivating the separation routine is not feasible, because *BDCut* has exponentially many constraints, and using the equivalent multi-commodity flow formulation *DF* instead leads to out-of-memory aborts on many instances. Thus, we concentrate on reduction methods and heuristics. Due to the long run times and limited computational resources, we have not included all test-sets from Table 2.1 in these experiments. Since we are interested in the big picture, we mostly use aggregated test-sets, to simplify presentation. The test-sets are as follows:

- *Adversarial* (PUC, SP)
- *Group* (WRP3, WRP4)
- *Rectilinear* (ES10000, TSPFST)
- *Telecom* (GEO-org, vienna-s)

– *VLSI*

In the following, we show the performance impact of deactivating: first, all reduction methods; second, all primal heuristics for finding feasible solutions during branch-and-bound; third, all primal heuristics both for branch-and-bound and for reduction techniques. In Table 2.2 we provide the computational results with a time limit of two hours. The first five rows of Table 2.2 list the percentual change in the run time with respected to the shifted geometric mean; the last five rows provide the corresponding percentual change with respect to the arithmetic mean of the run times. In this way, the first column of each row states the test-set to be considered. Ensuing, each of the next three columns provides the result of excluding the solving component specified in the head of the table. We emphasize that positive values signify a favorable impact of the respective algorithmic component on SCIP-Jack.

	Test-set	reduction techniques	B&B primal heuristics	all primal heuristics
sh. geo. mean time	Adversarial	**+75.8**	**+33.3**	**+63.3**
	Group	**+156.0**	**+48.0**	**+112.0**
	Rectilinear	**+750.0**	**+350.0**	**+425.0**
	Telecom	**+22821.1**	-3.5	**+31.6**
	VLSI	**+450.0**	**+25.0**	**+375.0**
arithmetic mean time	Adversarial	**+20.0**	**+2.3**	**+4.2**
	Group	**+294.2**	**+130.2**	**+174.3**
	Rectilinear	**+12613.9**	**+12528.3**	**+12759.4**
	Telecom	**+12508.2**	-5.3	**+8.7**
	VLSI	**+13325.1**	**+85.1**	**+7566.2**

Table 2.2: Each column reports the results of our SPG solver without the specified methods. The values denote the percentual change with respect to the default setting.

Unsurprisingly, the largest impact is achieved by the reduction techniques. Especially the increase in the arithmetic mean time is huge, which reflects the fact that many otherwise easily solvable instances become intractable without reduction methods. But also the primal heuristics have a considerable impact. An exception is the *Telecom* test-set, where the use of primal heuristics during branch-and-bound actually leads to a slight slowdown—for these instances the LP-relaxations are very tight, and optimal primal solutions can usually be found by a simple rounding procedure. Additionally, it can be seen that deactivating primal heuristics also for reduction techniques (as show in the last column) leads to a significant further slowdown. The most prominent example is the *VLSI* test-set, where the impact of deactivating primal heuristics throughout the entire solution process is an order of magnitude larger than the impact of just deactivating primal heuristics during branch-and-bound. This behavior shows the significance of primal heuristics for reduction techniques—where they are indispensable for bound-based methods. In turn, reduction methods are also used in several primal heuristics, which further increases their overall impact.

Reductions

This section demonstrates the strength of the reduction methods implemented in SCIP-Jack. To this end, we use a somewhat more aggressive reduction procedure than in the default version of SCIP-Jack. In this way, we can better convey the strength of the reduction techniques, since the default version of SCIP-Jack aborts the reductions early if the problem can be decomposed into sufficiently small connected components, or if the number of terminals is small. In the remainder of this chapter we will also use a slightly newer version of SCIP-Jack (as compared to the other chapters of this thesis), which include some (SPG-specific) improvements of the implementation of the extended reductions techniques. Table 2.3 shows the arithmetic mean of the percentage of vertices and edges in the presolved problems. Further, we report the shifted geometric mean (see Section 1.2.2) of the run time needed per test-set, with shift $s = 1$.

It can be seen that the considerable effort put into the various algorithms used within presolving pays off. Apart from the constructed, adversarial test-sets PUC, PUCN, and SP, the average size of both the number of vertices and edges is reduced by more than 50 percent on all test-sets, on most even my more than 80 percent. Additionally, many instances are already solved to optimality in presolving.

	average reduced problem size		
Test-set	nodes[%]	edges[%]	mean reduction time [s]
2R	9.9	12.7	1.0
Copenhag14	32.1	29.4	0.8
ES-R50	12.6	16.6	100.6
ES10000FST	15.1	16.8	39.7
GEO-a	21.5	22.5	13.0
GEO-org	5.8	6.5	14.1
LIN	7.6	7.5	2.5
PUCN	78.6	62.2	0.6
PUC	98.4	99.2	0.6
SP	37.5	37.5	0.3
TSPFST	10.2	11.2	0.2
vienna-a	4.8	5.0	2.2
vienna-s	2.0	1.8	2.7
VLSI	0.1	0.1	0.3
WRP3	48.4	48.6	1.1
WRP4	33.5	33.0	0.4

Table 2.3: Average problem sizes after application of reduction algorithms.

Naturally, there is a trade-off between the strength of the reductions (which can be strongly controlled by parameters) and the run time. Thus, it is also difficult to compare the strength of our reduction package with the state-of-the-art implementation by Polzin (2003) and Vahdati Daneshmand (2004). Even more so, because in the updated report Polzin and Vahdati-Daneshmand (2014) no reduction results are given, and a quite different machine is used in Polzin (2003) and Vahdati Daneshmand (2004). Still, we note that at least with regards to the size of the reduced instances, our reduction techniques are competitive with those of Polzin (2003) and Vahdati Daneshmand (2004). For example, on *WRP4* they report an average of 44.2 percent for the remaining edges, while we achieve 33.0 percent. On the other hand, on the *2R* instances Polzin (2003); Vahdati Daneshmand (2004) report only 6.2 percent of remaining edges, compared to 12.7 in our case.

These results also speak for the strength of the new reduction techniques developed in this thesis, since several reduction methods from Polzin (2003) and Vahdati Daneshmand (2004)—such as complete backtracking in extended reductions, or full terminal separator decomposition—have not been implemented in our solver yet.

Next, we concentrate on the impact of a particular class of reduction methods: Those based on the s_p distance. We have decided on this class of reductions, because they generalize the most important reduction concept for SPG, the bottleneck Steiner distance. We use seven benchmark sets from the literature; three from the DIMACS Challenge, three from the SteinLib, and one from Juhl et al. (2018). Table 2.4 shows in the first column the name of the test-set, followed by its number of instances. The next columns show the percentual average number of nodes and edges of the instances after the preprocessing without (column three and four), and with (columns five and six) the s_p based methods. The last two columns report the percentual relative change between the previous results.

It can be seen that the s_p methods allow for a significant additional reduction of the problem size. This behavior is rather remarkable, given the variety of other powerful reduction methods included in SCIP-Jack. Even if the percentage of remaining edges and nodes is already small on average for the base processing (such as for *VLSI*), there are for each of the seven test-sets at least a few instances that are still of large size. These instances can often be significantly reduced by the s_p techniques. While no run times are reported in the table, we note that on each of the seven test-sets the overall run time of the preprocessing (often significantly) decreases when the s_p based methods are used. Furthermore, even for other test-sets where the s_p methods are less (or not at all) successful, one does not observe an increase in the run time of the preprocessing above 10 percent.

		base preprocessing		$+s_p$ techniques		relative change	
Test-set	#	nodes [%]	edges [%]	nodes [%]	edges [%]	nodes [%]	edges [%]
VLSI	116	0.4	0.4	0.1	0.1	**-75.0**	**-75.0**
vienna-s	85	3.3	3.0	2.0	1.8	**-39.4**	**-40.0**
WRP4	63	36.2	36.0	33.5	33.0	**-7.5**	**-8.3**
Copenhag14	21	33.7	32.5	32.1	29.4	**-4.7**	**-9.5**
GEO-org	23	6.7	7.6	5.8	6.5	**-13.4**	**-14.5**
ES10000FST	1	24.1	27.1	15.1	16.8	**-37.3**	**-38.0**
ES-R50	15	17.5	22.8	12.6	16.6	**-28.0**	**-27.2**

Table 2.4: Average remaining nodes and edges after preprocessing.

2.7.2 PACE Challenge 2018

The *Parameterized Algorithms and Computational Experiments Challenge* (PACE) has been initiated to investigate the practical performance of parameterized algorithms. It is sponsored by the University of Amsterdam, Eindhoven University of Technology, Leiden University, and the Center for Mathematics and Computer Science (CWI). The 3rd PACE Challenge (Bonnet and Sikora, 2019), which took place in 2018, was concerned with fixed-parameter tractable (FPT) algorithms for the SPG— recall that for SPG instances with a fixed number of terminals or with a fixed treewidth, polynomial-time algorithms are known. The PACE Challenge 2018 included three tracks, each with 100 instances and a time limit of 30 minutes per instance. Overall, the challenge had 75 submissions.

Although SCIP-Jack does not include any FPT algorithms, Thorsten Koch and the author of this thesis decided to submit it to all three tracks of PACE 2018. Since no commercial solvers were allowed, SoPlex 4.0 was used as LP solver. In Track A (exact solution of instances with few terminals) SCIP-Jack reached 3rd place[12], in Track B (exact solution of instances with small treewidth) SCIP-Jack reached 1st place, and in Track C (heuristic solution of instances with different FPT characteristics) SCIP-Jack reached 2nd place[13].

While the actual instances used for the challenge have not been made available, a training test-set of 100 instances was published for each track (with the results on these test-sets being almost identical to the ones of the actual challenge). For these instances of the exact tracks A and B, we report results of running SCIP-Jack with SoPlex (the configuration used in the actual challenge) as LP solver in Table 2.5. In the actual challenge we used SoPlex 4.0, whereas in this thesis we use SoPlex 5.0, which is the latest version. However, there have been no performance improvements between these two versions. Note that the computational environment used for this thesis is different from the one at the PACE Challenge, which was hosted on the online platform *optil*, see Wasik et al. (2016). We observed that the SCIP-Jack version used at the PACE Challenge runs roughly 10 percent faster on the environment used in this thesis than on *optil*. To provide a reasonable comparison with other participants of the PACE Challenge, we have scaled the run times given in the following accordingly. The available memory was limited to 6 GB and a time limit of 30 minutes (or respectively 1620 seconds because of the scaling) was set, as in the PACE Challenge. The average time is reported as the arithmetic mean—since that was the secondary criterion at the PACE Challenge in case of a tie in the number of solved instances.

While SCIP-Jack-PACE, the version used at the challenge, can solve more than 90 % of all instances in both tracks within the time limit, it is substantially outperformed by SCIP-Jack-NEW, the latest version, which solves 99 of the 100 instances in each track. Also, the average run times are considerable smaller for SCIP-Jack-NEW. Notably, SCIP-Jack-NEW solves 99 of the 100 instances of Track A to optimality, while the winning solver in this track from the PACE Challenge solves 95.

[12] Winning team Track A: Yoichi Iwata, Takuto Shigemura (NII, Japan)

[13] Winning team Track C: Emmanuel Romero Ruiz, Emmanuel Antonio Cuevas, Irwin Enrique Villalobos López, Carlos Segura González (CIMAT, Mexico)

Even better results are achievable when CPLEX is used instead of SoPlex as LP solver. In this case already the PACE version of SCIP-Jack outperforms the best other solver in Track A. Likewise, the current SCIP-Jack can solve all 100 instances within the time-limit in Track A when CPLEX is used, and the average run time in both tracks is more than halved. Finally, we note that also for the heuristic track C, the current version of SCIP-Jack obtains a better score (as used in PACE) than that of the winning PACE solver.

		SCIP-Jack-NEW		SCIP-Jack-PACE		Best other
Track	# instances	solved	avg. time [s]	solved	avg. time [s]	solved
A	100	**99**	**38**	94	111	95
B	100	**99**	**25**	92	132	77

Table 2.5: Computational results for PACE 2018 instances.

2.7.3 SteinLib and beyond: A comprehensive benchmark

For the comparison with the solver by Polzin (2003); Vahdati Daneshmand (2004), we are restricted to the instances used in Polzin and Vahdati-Daneshmand (2014). Still, the experiments in Polzin and Vahdati-Daneshmand (2014) include a large number of test-sets (both the SteinLib and the 11th DIMACS Challenge collection). Thus, we only use test-sets with at least one instance that takes more than 10 seconds to be solved by Polzin and Vahdati-Daneshmand (2014) or our solver. There is one notable exception: We do not consider the test-sets *I320* and *I640* from the SteinLib; for the following reason: Polzin and Vahdati-Daneshmand (2014) use specialized, non-default settings for several test-sets, including *I320* and *I640*, where they use only "(...) fast calculation of bounds (...)" during branch-and-bound. As we aim to give an unbiased picture of the performance of our solver, we only use our default settings throughout this thesis. While we can achieve significant speed-ups on all tests-sets when using specialized settings, the impact is by far strongest on the I instances—more than an order of magnitude for the harder instances. We note, however, that we can match the results from Polzin and Vahdati-Daneshmand (2014) on I320 and I640 if we use dual-ascent bounds during branch-and-bound, instead of LP-based ones[14]. However, using a different algorithm for the base component branch-and-cut constitutes a drastic change of our solver. Consequently, we do not provide comparisons for the *I320* and *I640* test-sets within the table. Other classic benchmark sets, such as the C and D sets from the SteinLib can be considered trivial for our solver (and for

[14] On *I320* the default version of SCIP-Jack solves all instances within 24 hours, and takes a (scaled) mean time of 23.1 seconds. With dual-ascent bounds, SCIP-Jack solves all instances with a mean time of 3.8 seconds. In comparison, the solver by Polzin and Vahdati-Daneshmand (2014) takes a mean time of 4.2 seconds. On *I640* the default version of SCIP-Jack solves 86 instances within 24 hours, and takes a mean time of 92.3 seconds. With dual-ascent bounds, SCIP-Jack solves 95 instances with a mean time of 24.1 seconds. The solver by Polzin and Vahdati-Daneshmand (2014) takes a mean time of 22.0 seconds, and also solves 95 instances to optimality.

that by Polzin and Vahdati-Daneshmand (2014)): All these instances are solved in at most 0.1 seconds by SCIP-Jack.

Since the solver by Polzin (2003); Vahdati Daneshmand (2004) is not publicly available, we give a few remarks concerning the comparison of the computational environments. According to the DIMACS benchmark software (DIMACS, 2015), the machine used in this thesis is 1.59 times faster than the computer used in Polzin and Vahdati-Daneshmand (2014)[15]—just like Polzin and Vahdati-Daneshmand (2014) we used the gcc 4.6.3 compiler for computing the benchmark score, with full-optimization. While the author of this thesis does not have access to the machine used in Polzin and Vahdati-Daneshmand (2014), preliminary experiments on different machines have shown that the DIMACS score is a good estimate for the performance of our solver. Thus, we have scaled the run times reported in the following accordingly. All results were obtained single-threaded. We also note that Polzin and Vahdati-Daneshmand (2014) use CPLEX 12.6 as LP solver, while we use (the latest) CPLEX 12.10 throughout this thesis (for reasons of consistency). However, the difference between the two CPLEX versions for SCIP-Jack is neglectable on the instances within this section: A change in the average run time can only be observed for the (less than five percent of) instances that need branching, and even there the impact is not statistically relevant when several random seeds are used.

We compare the solver by Polzin (2003); Vahdati Daneshmand (2004) and the new solver SCIP-Jack with respect to the mean time, the maximum time, and the number of solved instances. For the mean time we use the shifted geometric mean with a shift of 10. We note that a shift of $s = 1$ would lead to similar relative results of SCIP-Jack compared to Polzin and Vahdati-Daneshmand (2014). We also note that the use of an arithmetic mean would bias strongly in favor of SCIP-Jack, which is especially faster on harder instances. Table 2.6 provides the results for a time-limit of 24 hours, which is the same time-limit as used in Polzin and Vahdati-Daneshmand (2014). The second column shows the number of instances in the test-set. Column three shows the mean time taken by the solver of Polzin (2003); Vahdati Daneshmand (2004), column four shows the mean time of SCIP-Jack. The next column gives the relative speedup of SCIP-Jack. The next three columns provide the same information for the maximum run time, the last two columns give the number of solved instances.

It can be seen that SCIP-Jack consistently outperforms Polzin and Vahdati-Daneshmand (2014)—both with respect to mean and maximum time. Also, SCIP-Jack solves on each test-set at least as many instances as Polzin and Vahdati-Daneshmand (2014). The only test-set where Polzin and Vahdati-Daneshmand (2014) prevail is *VLSI*. On this test-set the results of the extended reductions reported in Polzin (2003) are also stronger, which might be attributed to the use of full-backtracking, which has not yet been implemented in SCIP-Jack.

On the other test-sets, the difference in the run time is especially apparent for the maximum run time. This behavior can be explained by the fact that most test-sets contain many instances that can be solved very fast by both solvers—which brings the mean times closer together. Prominent examples are the *SP* and *Copenhag14* test-sets, for which all instances can be solved by SCIP-Jack within roughly one hour,

[15] Intel Core i7 920, 2.66 GHz

whereas Polzin and Vahdati-Daneshmand (2014) leave several instances unsolved even after 24 hours. Also, the primal-dual gap is significantly smaller for SCIP-Jack: The arithmetic mean on the unsolved PUC instances is 2.3 percent against 3.8 percent in Polzin and Vahdati-Daneshmand (2014).

		# solved		mean time (sh. geo. mean)			maximum time		
Test-set	#	P.&V.	S.-J.	P.&V. [s]	S.-J. [s]	speedup	P.&V. [s]	S.-J. [s]	speedup
VLSI	116	116	116	0.5	0.8	0.63	53.9	83.3	0.65
TSPFST	76	76	76	1.5	1.1	**1.36**	1161.4	263.1	**4.41**
WRP4	62	62	62	3.2	2.4	**1.33**	106.1	90.8	**1.17**
2R	27	27	27	5.0	3.0	**1.67**	43.9	21.1	**2.08**
vienna-a	85	85*	85	7.2	5.2	**1.38**	441.3	59.0	**7.48**
vienna-s	85	85*	85	7.8	6.2	**1.26**	623.5	60.7	**10.27**
WRP3	63	63	63	22.8	13.5	**1.69**	6073.2	4886.4	**1.24**
GEO-a	23	23*	23	158.7	55.3	**2.87**	6476.5	880.9	**7.35**
GEO-org	23	23*	23	145.6	58.5	**2.49**	4385.0	842.1	**5.21**
ES10000	1	1	1	138.0	83.0	**1.66**	138.0	83.0	**1.66**
Cophag14	21	20*	**21**	27.7	13.8	**2.01**	>86400	3845.3	>**22.47**
SP	8	6	**8**	159.4	25.8	**6.18**	>86400	1688.3	>**51.18**
LIN	37	35	**36**	31.3	14.7	**2.13**	>86400	>86400	1.00
PUC	50	17*	**18**	14964.9	11901.1	**1.26**	>86400	>86400	1.00

Table 2.6: Computational comparison of the solver developed for this article (*S.-J.*) and the solver described in Polzin (2003); Vahdati Daneshmand (2004) (*P.&V.*). Times marked by a $\star$ were obtained by P.&V. with (specialized) non-default settings.

As already mentioned, most test-sets in Table 2.6 contain a large number of instances that can be solved by both Polzin and Vahdati-Daneshmand (2014) and our solver in well below one second. To mitigate the impact of such very easy instances on the average times, we group the instances according to their hardness in the following experiment. We use instance groups $[10^k, 86400]$ for $k = -\infty, 0, 1, 2, 3$. Any group $[10^k, 86400]$ contains each instance from Table 2.6 such that Polzin and Vahdati-Daneshmand (2014) or SCIP-Jack solves this instance in not less than 10^k, and at most 86400 seconds. If an instance can be solved by only one solver within the time-limit, we consider the run time of the other solver on this instance as 86400 seconds. Such groupings are commonly used in computational mathematical optimization (also with the time lower bounds being powers of 10), see e.g. Müller et al. (2020); Witzig and Gleixner (2020). In addition to the shifted geometric mean, Table 2.7 also provides the arithmetic mean of the run time for each group. As before, we give the results for both Polzin and Vahdati-Daneshmand (2014) and SCIP-Jack, and report the respective speed-up of SCIP-Jack.

Unsurprisingly, the ratio of the arithmetic mean stays largely unchanged with increasing hardness of the groups. SCIP-Jack is more than a factor of 4 faster than the solver from Polzin and Vahdati-Daneshmand (2014) on all groups. On the other hand, the performance difference with respect to the shifted geometric mean significantly increases with the hardness of the instances. For instances that take more than a thousand seconds to be solved by Polzin and Vahdati-Daneshmand (2014) or SCIP-Jack, the latter is even by a factor of more than 7 faster. This behavior

		shifted geometric mean time			arithmetic mean time		
Group	#	P.&V. [s]	S.-J. [s]	speedup	P.&V. [s]	S.-J. [s]	speedup
$[0, 86400]$	644	12.2	7.9	**1.54**	1235.5	264.9	**4.66**
$[1, 86400]$	342	34.5	19.6	**1.76**	2326.4	498.7	**4.66**
$[10, 86400]$	180	122.5	52.0	**2.36**	4417.1	944.6	**4.68**
$[100, 86400]$	66	1403.2	286.6	**4.90**	11999.0	2546.3	**4.71**
$[1000, 86400]$	30	8035.8	1096.9	**7.33**	25923.1	5435.6	**4.77**

Table 2.7: Computational comparison of the solver developed for this article (*S.-J.*) and the solver described in Polzin (2003); Vahdati Daneshmand (2004) (*P.&V.*), with instance groups ordered by hardness.

can be put down to the fact that we employ a proper branch-and-cut algorithm, whereas the procedure employed by Polzin and Vahdati-Daneshmand (2014) might more accurately be coined *branch-and-reduce*. They employ the information from LP-relaxations mostly to perform more reductions, and often restart the LP solving process. This procedure is certainly advantageous for instances that can be solved completely by reduction techniques. However, for instances that are still of significant size after aggressive application of advanced reduction techniques, the LP separation approach of SCIP-Jack shows its value.

Further results and comparisons

We also note the large performance gap between SCIP-Jack and the best SPG solvers other than Polzin and Vahdati-Daneshmand (2014) described in the literature. For example, the solver by Fischetti et al. (2017), which won the exact SPG category at the 11th DIMACS Challenge, leaves 11 instances of *vienna-a* unsolved at the time-limit of one hour (using a faster machine than Polzin and Vahdati-Daneshmand (2014)), whereas we can solve all these instances within one minute, some even within one second. Furthermore, many of the non-trivial SPG instances that are solved to optimality in Fischetti et al. (2017) can be solved more than two or even three orders of magnitude faster by SCIP-Jack.

We close with results on two test-sets for which Polzin and Vahdati-Daneshmand (2014) do not report results: *EST-50k* and *PUCN*. The *EST-50k* instances can all be solved within five minutes. These instances are originally Euclidean Steiner tree problems; see Section 5.4 for more details and further results. Notably, 7 of the 15 instances from *EST-50k* were solved for the first time to optimality. On the other hand, the state-of-the-art Euclidean Steiner tree solver GeoSteiner cannot solve these instances even after seven days of computation (Juhl et al., 2018). Results on *EST-50k* are also reported in Pajor et al. (2017). However, their solver, which won the SPG heuristics category at the 11th DIMACS Challenge, does not reach the upper bounds from GeoSteiner on any of the *EST-50k* instances.

On the, unweighted, *PUCN* instances, SCIP-Jack also shows a strong performance. It solves 9 of the 13 instances to optimality—all in less than 10 minutes, and all but one within seconds. Indeed, four of the instances were solved for the first time to optimality. The best other results on *PUCN* are reported in Fischetti et al. (2017), who solve five instances within their time-limit of one hour, and in Pajor et al. (2017), who solve the same five instances to optimality. Fischetti et al. (2017) further apply specialized (USPG) heuristics, which they run multiple times with different random seeds. Some of these bound are further improved by Pajor et al. (2017). In Table 2.8 we compare these results with those obtained by SCIP-Jack within a 24 hours run (with default settings), reporting the instances that can be solved for the first time to optimality.

Name	gap [%]	new UB	previous UB
cc10-2n	**opt**	**179**	179
cc3-10n	**opt**	**75**	75
cc3-11n	**opt**	**92**	92
cc3-12n	**opt**	**111**	111

Table 2.8: PUCN instances solved for the first time to optimality (with 24 h time-limit).

Finally, we note that Fischetti et al. (2017) use the vertex-based Formulation 2.5 (TNCut) on the *PUCN* instances. Recall that we have shown in Section 2.2.2 that TNCut has a weaker LP-relaxation than the bidirected cut formulation (used by SCIP-Jack). Thus, the results on *PUCN*, where advanced reduction techniques have little impact, can be seen as a practical affirmation of this theoretical result.

2.8 Conclusion

This chapter has aimed to improve the state of the art in exact SPG solution. The path towards this goal turned out to be rather long and steep. Starting from new theoretical results for well-known IP formulations, we have introduced a wide range of techniques and algorithms to be combined in an exact SPG solver. Notably, we have shown new (and stronger) conditions for the LP-relaxation of the bidirected cut formulation to be tight. Moreover, we have seen that several of the new algorithms and concepts provably dominate well-known results from the literature, such as the bottleneck Steiner distance. Finally, the integration of the various components into a branch-and-cut algorithm has given way to an exact SPG solver that outperforms the formerly undisputed state-of-the-art method established by Polzin (2003); Vahdati Daneshmand (2004) almost 20 years ago. Moreover, several SPG benchmark instances have been solved for the first time to optimality.

Interestingly, the strong performance of the new SPG solver is achieved despite several important algorithms and techniques from Polzin (2003); Vahdati Daneshmand (2004) not being implemented in the solver yet. Furthermore, several new techniques introduced in this thesis have also not yet been implemented in full scope. Thus, there is still a high potential for further improvement. We provide more detail on possible further improvements and research directions in Chapter 7.

Chapter 3

A relative: The maximum-weight connected subgraph problem

This chapter is concerned with a relative of the SPG, the *maximum-weight connected subgraph problem* (MWCSP). While at first glance this problem may appear to be rather different from the SPG, this chapter shows that the two are in fact closely related. Aiming for faster exact solution, we show how several algorithmic techniques from the previous chapter can be extended for MWCSP. However, to decisively outperform state-of-the-art MWCSP solvers, fully independent MWCSP algorithms and techniques need to be developed. As before, the trajectory will be from theory to practice—with a special emphasis on the theoretical strength of the employed IP formulations and their polyhedral properties.

3.1 Introduction

The past ten years have witnessed a surge of research articles dealing with the MWCSP. As practitioners, for instance in computational biology, have become more aware of this problem and its practical potential, their work has in turn (re-)fueled the interest of mathematicians and computer scientists. The source of this symbiotic interplay is a rather plain looking problem: Given an undirected graph $G = (V, E)$ and vertex weights $p : V \to \mathbb{Q}$, the task is to find a connected subgraph $S = (V(S), E(S)) \subseteq G$ such that

$$P(S) := \sum_{v \in V(S)} p(v) \tag{3.1}$$

is maximized. While computational biology, see e.g. Alcaraz et al. (2014); Dittrich et al. (2008); Klimm et al. (2020); Loboda et al. (2016), seems to be the most prominent application field for the MWCSP, one also encounters the problem in other, disparate, areas such as wildlife conservation, e.g. Dilkina and Gomes (2010), and computer vision, e.g. Chen and Grauman (2012), or robotics, e.g. Banfi (2018).

The MWCSP is $\mathcal{NP}$-hard, see e.g. Johnson (1985). It is even $\mathcal{NP}$-hard to approximate the MWCSP within any constant factor as shown in Álvarez-Miranda et al. (2013a). Furthermore, the MWCSP is fixed-parameter tractable in both the number of positive vertices, see Section 4.2.2, and the number of non-positive vertices, see Buchanan et al. (2018). Various articles discuss theoretical aspects of the MWCSP, such as the strength of (mixed) integer-programming formulations, e.g. Álvarez-Miranda et al. (2013b); Carvajal et al. (2013), polyhedral descriptions, e.g. Biha et al. (2015); Wang et al. (2017), or complexity, e.g. Álvarez-Miranda et al. (2013a); Buchanan et al. (2018). Practical exact algorithms for MWCSP can for example be found in Álvarez-Miranda et al. (2013a); Backes et al. (2011); Fischetti et al. (2017); Leitner et al. (2018a).

The MWCSP can also be seen as fundamental model for optimization problems based on *induced connectivity*. I.e., one looks for a subsets of vertices, such that the subgraph *induced* by these vertices is connected. Which edges are selected to obtain connectivity is not relevant. This problem type can be found in many clustering and network analysis applications. In addition to the above mentioned areas, *induced connectivity* problems are found in social network analysis, see Moody and White (2003), political districting, see Garfinkel and Nemhauser (1970), wireless sensor network design, see Buchanan et al. (2015). Also the unweighted (as well as uniformly weighted) Steiner tree problem in graphs is based on induced connectivity: Any solution (i.e. Steiner tree) consisting of n nodes will be of weight $n-1$; it does not matter which $n-1$ edges are selected as long as they connect the given nodes. As we will see in the following, the MWCSP can be regarded a generalization of the unweighted SPG.

3.1.1 Preliminaries and additional notation

Throughout the algorithmic part of this chapter—starting with Section 3.3—it will be assumed that in each MWCSP at least one vertex is assigned a negative and one a positive weight. In the case of only non-negative vertex weights, the MWCSP reduces to finding a connected component of maximum vertex weight; in the case of only non-positive vertex weights, the empty set constitutes an optimal solution. Moreover, for most algorithms it will be assumed that an MWCSP instance $I_{MW} = (V, E, p)$ is given such that the underlying graph (V, E) is connected. The latter assumption does not limit the generality, as one can optimize each connected components of a non-connected MWCSP separately. We define $T_p := \{v \in V \mid p(v) > 0\}$, and occasionally write for the sake of simplicity $V = \{v_1, v_2, ..., v_n\}$ as well as $T_p = \{t_1, t_2, ..., t_k\}$, with $k = |T_p|$.

A close relative of the MWCSP is the *rooted maximum-weight connected subgraph problem* (RMWCSP), see e.g. Álvarez-Miranda et al. (2013b), which incorporates the additional condition that a non-empty set $T_f \subseteq V$ needs to be part of any feasible solution. For simplicity, we usually assume that $p(t) = 0$ for all $t \in T_f$. The unweighted SPG be formulated as a RMWCSP by assigning each non-terminal vertex a weight of -1.

We introduce the distance function $\overline{d} : V \times V \mapsto \mathbb{Q} \cup \{-\infty\}$ defined as

$$\overline{d}(v_i, v_j) := \sup\{P(Q) \mid Q \text{ is a } (v_i, v_j)\text{-path and } (V(Q) \setminus \{v_i, v_j\}) \cap T_p = \emptyset\} \quad (3.2)$$

for any $v_i, v_j \in V$. In particular, $\overline{d}(v_i, v_j) = \overline{d}(v_j, v_i)$ and $\overline{d}(v_i, v_i) = p(v_i)$. Also, with the convention $\sup \emptyset = -\infty$, one observes that $\overline{d}(v_i, v_j) = -\infty$ if and only if there is no path between v_i and v_j without intermediary positive vertices. Given a vertex v_0 and two additional vertices $v_i, v_j \in V \setminus \{v_0\}$, it will be said that for v_0 vertex v_i is $\overline{d}$-*nearer* than vertex v_j if $\overline{d}(v_0, v_i) \geqslant \overline{d}(v_0, v_j)$. For each vertex v_i the k $\overline{d}$-nearest vertices of positive weight (if existent) are denoted by $\overline{v}_{i,1}, \overline{v}_{i,2}, ..., \overline{v}_{i,k}$. In Duin (1993) a similar distance function is defined for the Steiner tree problem in graphs that looks for paths of minimum edge weight without intermediary terminals—thus, we have chosen a corresponding notation.

We define the *neighborhood* of a vertex set $U \subseteq V$ as

$$N(U) := \big\{ v \in V \setminus U \mid \exists u \in U, \{u, v\} \in \delta(U) \big\}.$$

For a single $v \in V$ we set $N(v) := N(\{v\})$. For directed graphs we define

$$N^+(U) := \big\{ v \in V \setminus U \mid \exists u \in U, (u, v) \in \delta^+(U) \big\}.$$

Finally, given an r-t flow f, we denote its *net flow value* by $v(f) := f(\delta^+(r)) - f(\delta^-(r))$. See Chapter 1 for general notation and concepts.

3.1.2 Contribution and structure

This chapter aims at further enhancing exact MWCSP solution-based on integer programming. First, Section 3.2 analyzes integer and mixed integer formulations for MWCSP and RMWCSP. In particular, node-based formulations, which have gained notable attention in the recent literature, are compared with edge-based ones. It will be shown that the latter prevail with respect to the strength of their LP-relaxations. Furthermore, polyhedral results are given, including a (compact extended) description of the connected subgraph polytope—the convex hull of subsets of vertices that induce a connected subgraph—for all graphs with no four independent vertices.

The remainder of the chapter is concerned with the (practical) exact solution of MWCSP based on the strongest of the previously studied IP formulation. We proceed by combining several approaches:

- Section 3.3 introduces several new MWCSP reduction techniques. We show that some of these techniques require to solve $\mathcal{NP}$-hard subproblems. However, the underlying concepts allow us to design empirically powerful approximations.
- Section 3.4 links the preceding two sections. It introduces transformations of MWCSP and RMWCSP to SAP, and discusses the use of dual-ascent on these SAPs. In this way, further reduction techniques can be applied, and we can also generate strong initial cutting planes for the IP formulations used in this chapter.
- Section 3.5 turns from dual to primal bounds. We introduce several new primal heuristics for MWCSP, which not only make use of the concepts introduced in the previous sections, but can also be used to strengthen them.
- Section 3.6 discusses the incorporation of the new techniques introduced in this chapter into an exact MWCSP solver. Furthermore, the practical performance of this solver is compared with previous results from the literature.

The new MWCSP solver significantly outperforms previous solvers from the literature—being often orders of magnitude faster, and solving more instances to optimality. As a result, several benchmark instances from the 11th DIMACS Challenge can be solved for the first time to optimality, and the best known solution for other ones can be improved.

3.2 (M)IP formulations and the connected subgraph polytope

In this section, we use $\mathbb{R}$ instead of $\mathbb{Q}$ for the vertex weights, because we will not be concerned with complexity, but rather polyhedral results.

3.2.1 Rooted maximum-weight connected subgraphs

This section discusses the directed variant of the RMWCSP, see Álvarez-Miranda et al. (2013b): Given a directed graph $D = (V, A)$, vertex weights $p : V \to \mathbb{R}$, a non-empty set $T_f \subseteq V$ and an $r \in T_f$, find a connected subgraph $S \subseteq D$ containing T_f such that any $v \in V(S)$ can be reached from r on a directed path in S, and such that $p(V(S))$ is maximized. Any undirected RMWCSP can be formulated in directed form by choosing an arbitrary $r \in T_f$ and replacing each edge by two anti-parallel arcs.

Note that any solution to the directed RMWCSP can be represented as an arborescence. This observation leads to the following IP formulation, see e.g. Álvarez-Miranda et al. (2013b), based on a well-known formulation for SAP, see e.g. Goemans and Myung (1993). Define for each $v \in V$ a variable $x(v) \in \{0, 1\}$ that is equal to 1 if and only if vertex v is part of the solution. Analogously, define for each $a \in A$ a variable $y(a) \in \{0, 1\}$.

Formulation 3.1. *Rooted Steiner Arborescence Formulation (RSA)*

$$
\begin{aligned}
&max\ p^T x && && (3.3)\\
&s.t.\ y(\delta^-(v)) = x(v) && \text{for all } v \in V \setminus \{r\} && (3.4)\\
&\quad y(\delta^-(U)) \geqslant x(v) && \text{for all } U \subseteq V \setminus \{r\},\ v \in U && (3.5)\\
&\quad x(t) = 1 && \text{for all } t \in T_f && (3.6)\\
&\quad x \in \{0,1\}^V && && (3.7)\\
&\quad y \in \{0,1\}^A. && && (3.8)
\end{aligned}
$$

In Álvarez-Miranda et al. (2013b) a new formulation for the directed RPCSTP-based on node-separators is introduced. Note that the use of node-separators for modeling connectivity is already suggested in Fügenschuh and Fügenschuh (2008).

Formulation 3.2. *Rooted Node Separator Formulation (RNCut)*

$$\begin{aligned}
\max \quad & p^T x && && (3.9)\\
\text{s.t. } x(C) &\geqslant x(v) && \text{for all } v \in V \setminus (\{r\} \cup N^+(r)),\ C \in \mathcal{C}(r,v) && (3.10)\\
x(v) &= 1 && \text{for all } v \in T_f && (3.11)\\
x &\in \{0,1\}^A. && && (3.12)
\end{aligned}$$

Besides the two IP models introduced above, several other formulations for RMWCSP (sometimes including a budget constraint) have been introduced in the literature, see e.g. Álvarez-Miranda et al. (2013b); Dilkina and Gomes (2010). However, one can show that these formulations are weaker with respect to the LP-relaxation than both of the above models, see Álvarez-Miranda et al. (2013b) for some such results. Another example is the formulation from Conrad et al. (2007) that is based on single-flow. However, also this formulation can be shown to be weaker than Formulation 3.1 by using max-flow/min-cut arguments—similarly to corresponding results for minimum spanning tree or Steiner tree problems, which can be found for example in Magnanti and Wolsey (1995).

In Álvarez-Miranda et al. (2013b) it is stated that the LP-relaxations of the RNCut and RSA model yield the same optimal value. Unfortunately, this claim is not correct, as the following proposition shows. Appendix A.2.2 discusses the error in the line of argumentation in Álvarez-Miranda et al. (2013b)—and furthermore provides some insight on how the node separator constraints miss to capture structures accurately described by edge cut constraints.

Proposition 3.3. *It holds that $proj_x(\mathcal{P}_{LP}(RSA)) \subset \mathcal{P}_{LP}(RNCut)$ and the inclusion can be strict.*

Proof. The inclusion is essentially proven in Álvarez-Miranda et al. (2013b). An example for a strict inclusion is given in Appendix A.2.2. □

One can strengthen the RSA formulation by the inequalities

$$y(\delta^-(v)) \leqslant y(\delta^+(v)) \qquad \text{for all } v \in V \setminus (T_f \cup T_p), \tag{3.13}$$

which are similar to the flow-balance constraints used in Section 2.2. However, these constraints depend on the objective vector, so they cannot (directly) be used for polyhedral results. We refer to the strengthened formulation as RSA_{FB}. One readily obtains the following result from Lemma 2.3.

Lemma 3.4. *If $|T_p \cup T_f| \leqslant 3$, then $v_{LP}(RSA_{FB}) = v(RSA_{FB})$.*

Proof. Let I be an RMWCSP instance with $|T_p \cup T_f| \leqslant 3$. Define an SAP $I' = (V', A', T', c')$ on an extended graph (V', A'). Initially, set $V' := V$, $A' := A$, and $T' := T_f$. For each arc $a = (v,w) \in A$ set $c'(a) := \max\{-p(w), 0\}$. For each $t \in T_p$

we add a new terminal t' to T', and arcs (r, t') of weight $p(t)$ and (t, t') of weight 0 to A'. It holds that

$$p(T_p) - v(DCut_{FB}(I')) = v(RSA_{FB}(I)); \tag{3.14}$$

recall that we assume $T_p \cap T_f = \emptyset$. Any optimal LP solution (y, x) to RSA_{FB} can be extended to a feasible LP solution y' to $DCut_{FB}$ defined by $y'(t, t') = x(t)$, $y'(r, t') = 1 - x(t)$ for all $t \in T_p$, as well as $y'(a) := y(a)$ for all $a \in A$. Thus,

$$p(T_p) - v_{LP}(DCut_{FB}(I')) \geqslant v_{LP}(RSA_{FB}(I)) \geqslant v(RSA_{FB}(I)). \tag{3.15}$$

Because I' has at most three terminals, Lemma 2.3 guarantees that

$$v_{LP}(DCut_{FB}(I')) = v(DCut_{FB}(I')).$$

Thus, (3.15) implies that the inequalities (3.15) are satisfied with equality. Consequently, we have $v_{LP}(RSA_{FB}(I)) = v(RSA_{FB}(I))$. □

3.2.2 Node based formulations for non-rooted connected subgraphs

From this section on we consider the undirected MWCSP. Some of the following results can also be extended to the directed case. However, the undirected MWCSP is the more common (and, arguably, also more natural) problem.

This section considers formulations for MWCSP that use only node variables. The probably best known one, see e.g. Wang et al. (2017), is given below.

Formulation 3.5. *Node Separator Formulation (NCut)*

$$\begin{aligned} max \quad & p^T x & & (3.16)\\ s.t.\ x(v) + v(w) - x(C) \quad & \leqslant \quad 1 & \textit{for all } v, w \in V, v \neq w, C \in \mathcal{C}(v, w) \quad & (3.17)\\ x(v) \quad & \in \quad \{0, 1\} & \textit{for all } v \in V. \quad & (3.18) \end{aligned}$$

The contraction of neighboring positive weight vertices drastically reduces the size of many real-world MWCSP instances, as for example shown in Rehfeldt et al. (2019). Note that when contracting adjacent vertices $t, u \in T_p$ into a new vertex t', we set $p(t') := p(t) + p(u)$. The following result (which we will need later on), describes the impact of this operation on the LP-relaxation of $NCut$.

Proposition 3.6. *$v_{LP}(NCut)$ is invariant under the contraction of adjacent vertices of positive weight.*

Proof. Let I be an MWCSP instance with an edge $\{t, u\} \in E$ such that $t, u \in T_p$. Let $I' = (V', E', p')$ be the instance obtained from I be contracting $\{t, u\}$ into a new vertex t'. It holds that $v_{LP}(NCut(I')) \leqslant v_{LP}(NCut(I))$, because any $x' \in \mathcal{P}_{LP}(NCut(I'))$ can be mapped to a $x \in \mathcal{P}_{LP}(NCut(I))$ with $p^T x = {p'}^T x'$ defined by $x(v) := x'(v)$ for all $v \in V \cap V'$, and $x(t) := x(u) := x'(t')$. The opposite case is somewhat more involved.

Let x be an optimal LP solution to $NCut(I)$. The optimality of x, and the fact that $\{t, u\} \in E$ imply

$$x(t) = x(u). \tag{3.19}$$

Define $x' \in \mathbb{R}^{V'}$ by $x'(v) := x(v)$ for all $v \in V' \setminus \{t'\}$, and $x'(t') := x(t)$. Assume that $x'(t') \in (0, 1)$—otherwise, the proof is already complete. It remains to be shown that $x' \in \mathcal{P}_{LP}(NCut(I'))$. Suppose this is not the case. Then there are $a, b \in V'$ and an a-b separator $C'_{ab} \subset V'$ such that

$$x'(a) + x'(b) - x'(C'_{ab}) > 1. \tag{3.20}$$

Because x is feasible, $t' \in C'_{ab}$. Thus, we obtain from (3.20) that

$$x(a) + x(b) - x(t) = x'(a) + x'(b) - x'(t') > 1, \tag{3.21}$$

and therefore

$$\min\{x(a), x(b)\} > x(t). \tag{3.22}$$

Now we return to the original instance I. Because x is optimal, and $x(t) = x(u) < 1$, there is a $q \in V \setminus \{t, u\}$ and a $C_{qt} \in \mathcal{C}(q, t)$ such that

$$x(t) + x(q) - x(C_{qt}) = 1. \tag{3.23}$$

Similarly, there is a $s \in V \setminus \{t, u\}$ and a $C_{su} \in \mathcal{C}(s, u)$ with $x(u) + x(s) - x(C_{su}) = 1$. At least one such combination q, C_{qt}, or s, C_{su} satisfies $u \notin C_{qt}$ or $t \notin C_{su}$, otherwise we could increase $x(u)$ and $x(t)$. Assume w.l.o.g. $u \notin C_{qt}$. Further, observe that (3.23) implies

$$x(C_{qt}) \leqslant \min\{x(t), x(q)\}. \tag{3.24}$$

Thus, (3.23) and (3.22) imply $a, b \notin C_{qt}$. One notes that $C_{qt} \notin \mathcal{C}(a, q)$, because (3.22) and (3.23) imply

$$x(a) + x(q) - x(C_{qt}) > 1. \tag{3.25}$$

Likewise, $C_{qt} \notin \mathcal{C}(b, q)$. Consequently, any path from $\{t, u\}$ to a or b needs to cross C_{qt}; otherwise, the latter would not separate q and t. Therefore, $\tilde{C}_{ab} := (C'_{ab} \setminus \{t'\}) \cup C_{qt}$ separates a and b (in the original graph). However, from (3.20) and (3.24) we obtain

$$1 < x(a) + x(b) - x'(C'_{ab}) \leqslant x(a) + x(b) - x(\tilde{C}_{ab}), \tag{3.26}$$

which contradicts the feasibility of x. □

Furthermore, one obtains the following optimality criterion:

Proposition 3.7. *If $|T_p| \leqslant 2$, then $v_{LP}(NCut) = v(NCut)$.*

Proof. Consider an MWCSP $I = (G, p)$ with $|T_p| \leqslant 2$. The case $|T_p| \leqslant 1$ is clear. Let $\{a, b\} := T_p$ and assume $p(a) \geqslant p(b)$. Thus, there is a minimal optimal LP solution x such that $x(a) = 1$. Let (V, A) be the bidirected equivalent of G. Create a new directed graph (V', A') by replacing each node $v \in V \setminus \{a, b\}$ by two nodes v_1, v_2 and arcs $(v_1, v_2), (v_2, v_1)$. Further, all ingoing arcs of v become ingoing arcs of v_1, and all outgoing arcs of v are now outgoing arcs of v_2. Define arc capacities k for each pair of these new arcs by $x(v)$; for any (remaining) arc $e \in A$ set $k(e) := \infty$.[16] By the max-flow/min-cut theorem there is an a-b flow f with $v(f) = x(b)$ in this extended network. Define the directed MWCSP $I_r := ((V, A), T_f, r, p)$ with $T_f := \{a\}$ and $r := a$, and set $y := f \restriction_A$. Because of the optimality and minimality of x it holds that $(x, y) \in \mathcal{P}_{LP}(RSA(I_r))$. Thus, $v_{LP}(NCut(I)) \leqslant v_{LP}(RSA(I_r))$. Furthermore, y satisfies constraints (3.13). Because of $v(NCut(I)) = v(RSA(I_r))$, Lemma 3.4 implies that $v_{LP}(NCut(I)) = v(NCut(I))$. □

Figure 3.1 shows an MWCSP instance with $|T_p| = 3$ and $v_{LP}(NCut) \neq v(NCut)$. It holds that $v(NCut) = 1$, but $v_{LP}(NCut) = 1.5$ (set the values of all negative weight node variables to 0.5 and the remainder to 1).

Finally, by combining the previous two propositions we obtain a significantly shorter proof of a main result from Wang et al. (2017). Recall that $\alpha(G)$ denotes the independence number of graph G.

Theorem 3.8. *If $\alpha(G) \leqslant 2$, then $\mathcal{P}_{LP}(NCut)$ is integral.*

Proof. Let $p \in \mathbb{R}^V$. If $\alpha(G) \leqslant 2$, then Proposition 3.6 implies that the MWCSP (G, p) can be transformed to an MWCSP with at most two positive weight vertices without changing $v_{LP}(NCut)$. Now, Proposition 3.7 gives $v_{LP}(NCut) = v(NCut)$. Because p can be chosen arbitrarily, $\mathcal{P}_{LP}(NCut)$ is integral. □

Wang et al. (2017) also show that $\mathcal{P}_{LP}(NCut)$ is integral only if $\alpha(G) \leqslant 2$.

Indegree constraints

Given an undirected graph $G = (V, E)$, a $d \in \mathbb{Z}^E$ is an *indegree vector* if there is an orientation $D = (V, A)$ of G such that $d_v = |\delta_D^-(v)|$ for all $v \in V$. For each indegree vector d the corresponding *indegree inequality* is given as

$$\sum_{v \in V} (1 - d_v) x(v) \leqslant 1, \tag{3.27}$$

where $x \in \mathbb{R}^V_{\geqslant 0}$ are the node variables. Korte et al. (2012) show that the indegree inequalities describe the connected subgraph polytope if G is a tree. Furthermore, Wang et al. (2017) show conditions for (3.27) to be facet inducing and show that the constraints can be separated in linear time. It is further shown that the constraints (3.27) (for suitable choices of indegree vectors) can strengthen the $NCut$ formulation.

[16] Such kind of flow network transformations are well-known in algorithmic graph theory; see also Appendix A.2.2.

3.2.3 Edge based formulations for non-rooted connected subgraphs

An edge-based formulation for the directed MWCSP is introduced in Álvarez-Miranda et al. (2013a), based on a transformation to the prize-collecting SPG. We will use essentially the same formulation for the undirected MWCSP, but without the transformation to the prize-collecting SPG, and thus with a different objective function. Consider the bidirected equivalent $D = (V, A)$ to the given undirected graph. Let (V_r, A_r) be the directed graph defined as follows with an additional node r:

$$V_r := V \cup \{r\},$$

and

$$A_r := A \cup \{(r, v) \mid v \in V\}.$$

Define the following extended MWCSP formulation based on the new graph (V_r, A_r).

Formulation 3.9. *Extended Steiner Arborescence Formulation (ESA)*

$$\begin{aligned}
&\mathit{max}\ p^T x && && (3.28)\\
&\mathit{s.t.}\ y(\delta^-(v)) = x(v) && \text{for all } v \in V && (3.29)\\
&y(\delta^-(U)) \geqslant x(v) && \text{for all } U \subseteq V,\ v \in U && (3.30)\\
&y(\delta^+(r)) \leqslant 1 && && (3.31)\\
&x \in \{0,1\}^V && && (3.32)\\
&y \in \{0,1\}^{A_r}. && && (3.33)
\end{aligned}$$

The remainder this section aims to prove an integrality condition for the polytope $proj_x(\mathcal{P}_{LP}(ESA))$ based on the independence number. Our approach can be divided in two parts. First, we show that for any MWCSP instance with $1 \leqslant |T_p| \leqslant 3$ there is an optimal LP solution (x, y) with $x(v) = 1$ for a $v \in T_p$. In the second part (following Lemma 3.13), we use this v as a root node and apply the same principal ideas already used in Section 2.2.2 for the USPG: I.e. we show the invariance of the integrality gap under edge contraction and reduce any MWCSP instance with bounded independence number to an MWCSP instance with bounded number of positive vertices. We start with an easy technical result.

Lemma 3.10. *Let (x, y) be an optimal LP solution (x, y) to ESA, and let $v \in V$. There is a $\tilde{y} \in \mathbb{R}^{A_r}$ with $\tilde{y}((r, v)) = x(v)$ such that $(x, \tilde{y})$ is an optimal LP solution to ESA.*

Proof. Assume there is an optimal LP solution (x, y) with $\kappa := y(\delta^-_D(v)) > 0$ for a $v \in V$. Because of (3.30), there is an r-v flow $f_\kappa \leqslant y$ with $v(f_\kappa) = \kappa$ and $f_\kappa((r, v)) = 0$. Define a new solution $(x, \tilde{y})$ with

$$\tilde{y}((u, w)) := \begin{cases} y((u, w)) - f_\kappa((u, w)) + f_\kappa((w, u)), & (u, w) \in A \\ y((u, w)) - f_\kappa((u, w)), & (u, w) \in \delta^+(r) \setminus \{(r, v)\} \\ y((u, w)) + \kappa, & (u, w) = (r, v). \end{cases}$$

For (3.29), first let $u \in V \setminus \{v\}$. It holds that

$$\tilde{y}(\delta^-(u)) = y(\delta^-(u)) + f_\kappa(\delta^+_D(u)) - f_\kappa(\delta^-_D(u)) - f_\kappa((r,u)) = y(\delta^-(u)) = x(u).$$

Similarly, because of $\tilde{y}((r,v)) = y((r,v)) + \kappa$ and $f_\kappa(\delta^-_D(v)) + f_\kappa(\delta^+_D(v)) = \kappa$ it holds that $\tilde{y}(\delta^-(v)) = x(v)$.

For (3.30), consider a $U \subseteq V$, and a $u \in U$. First, assume $v \in U$. Because of $(r,v) \in \delta^-(U)$, and $f_\kappa(\delta^-_D(U)) + f_\kappa(\delta^+_D(U)) = \kappa$ we obtain $\tilde{y}(\delta^-(U)) = y(\delta^-(U))$. Second, assume $v \notin U$. In this case, flow conservation of f_κ implies that

$$\tilde{y}(\delta^-(U)) = y(\delta^-(U)) + f_\kappa(\delta^+(U)) - f_\kappa(\delta^-(U)) = y(\delta^-(U)),$$

which concludes the proof. □

Note that for finding an optimal solution, ESA only requires arcs $(r,v) \in \delta^+(r)$ with $v \in T_p$. Furthermore, only constraints (3.30) for vertices $v \in U$ with $v \in T_p$ need to be enforced. We will refer to this modified formulation as ESA^+. Further, we define

$$A_r^+ := A \cup \{(r,t) \mid t \in T_p\}.$$

In practice, it is advisable to add additional $|T_p|$ symmetry breaking constraints similar to those from Fischetti et al. (2017) to ESA^+. As to the LP-relaxation of ESA^+, one obtains the following result.

Lemma 3.11. *Let (x, y^+) be an optimal LP solution to ESA^+. Then $(x,y) \in \mathbb{R}^{V+A_r}$ with $y(a) := y^+(a)$ for $a \in A_r^+$ and $y(a) := 0$ for $a \in A_r \setminus A_r^+$ is an optimal LP solution to ESA.*

Proof. Let ESA' be the reduced version of ESA where constraints (3.30) are only enforced for vertices $v \in U$ with $v \in T_p$. Note that

$$v_{LP}(ESA) \leqslant v_{LP}(ESA') \leqslant v_{LP}(ESA^+). \tag{3.34}$$

In this proof we only consider minimal optimal LP solutions, i.e., solutions for which no entry can be reduced without losing either feasibility or optimality.

First, we show that any optimal LP solution to ESA^+ is also optimal for ESA'. To this end, we show the existence of an optimal LP solution (x', y') to ESA' such that $y'((r,v)) = 0$ for all $v \in V \setminus T_p$. Assume there is an optimal LP solution (x', y') to ESA' with $y'((r,v)) > 0$ for a $v \in V \setminus T_p$. Because (x', y') is optimal, there is a r-t flow f^t with $f^t \leqslant y'$ for a $t \in T_p$ with $v(f^t) = y'((r,v))$. We can now proceed as in Lemma 3.10 to revert the flow going to t. The resulting optimal solution $(\tilde{x}, \tilde{y})$ satisfies $\tilde{y}((r,v)) = 0$ and $\tilde{y}((r,u)) \leqslant y'((r,u))$ for all $u \in V \setminus \{t\}$.

Second, we show that any optimal LP solution (x', y') to ESA' with $y'((r,v)) = 0$ for all $v \in V \setminus T_p$ satisfies constraints (3.30) also for $v \in U$ with $v \notin T_p$. We follow essentially the same line of argumentation used in Goemans and Myung (1993) for the SPG bidirected cut formulation. Suppose there is a $U \subseteq V$ and a $u \in U$ with

$$x'(u) > y'(\delta^-(U)). \tag{3.35}$$

Choose such a U with $|U|$ as small as possible. Because of (3.35), there is an $e \in \delta^-(u) \setminus \delta^-(U)$ such that $y'(e) > 0$. Because of the minimality of (x', y'), there is a $W \subseteq V$ and a $t \in W \cap T_p$ such that $e \in \delta^-(W)$ and

$$y'(\delta^-(W)) = x'(t). \tag{3.36}$$

Because of $e \subseteq U$ and $|e \cap W| = 1$, one obtains $|U \cap W| < |U|$. We will show that $U \cap W$ satisfies (3.35), which contradicts the minimality of $|U|$. By standard graph theory we have that

$$y'((\delta^-(U)) + y'(\delta^-(W)) \geqslant y'(\delta^-(U \cap W)) + y'(\delta^-(U \cup W)).$$

Together with (3.36), it follows that $y'((\delta^-(U)) \geqslant y'(\delta^-(U \cup W))$, which leads to the sought for contradiction. □

Corollary 3.12. $v_{LP}(ESA) = v_{LP}(ESA^+)$.

Further, we require the following result.

Lemma 3.13. *If $|T_p| \leqslant 3$, then there is an optimal LP solution (x, y) to ESA such that $x(t) \in \{0, 1\}$ for all $t \in T_p$.*

Proof. As before, let $D = (V, A)$ be the bidirected equivalent to the given undirected graph. Also, we assume any optimal solution to be minimal. By Lemma 3.11 we can consider ESA^+ instead of ESA to show the required result. Thus, throughout this proof we consider an optimal LP solution (x, y) to ESA^+.

The case $|T_p| \leqslant 1$ is clear. Assume $|T_p| = 2$, and let $\{a, b\} := T_p$ such that $p(a) \geqslant p(b)$. By Lemma 3.10 we can assume that $y(\delta_D^-(a)) = 0$. Thus, also $y(\delta_D^+(b)) = 0$. If $y(\delta^+(a)) = 0$, either $x(a) = 1$ and $x(b) = 0$, or vice versa. If $y(\delta^+(a)) > 0$, the minimality of (x, y) implies

$$\sum_{v \in V \setminus \{a\}} p(v) y(\delta_D^-(v)) > 0, \tag{3.37}$$

which implies also $\beta := y(\delta_D^-(b)) > 0$. Let $\kappa := \frac{1}{\beta}$. Define $\tilde{y} \in \mathbb{R}^{A_r^+}$ by $\tilde{y}((r, a)) := 1$, $\tilde{y}((r, b)) := 0$, and $\tilde{y}(e) := \kappa y(e)$ for all $e \in A$. Define $\tilde{x}(v) := \tilde{y}(\delta^-(v))$ for all $v \in V$. One notes that $(\tilde{x}, \tilde{y})$ is feasible, and satisfies $\tilde{x}(a) = \tilde{x}(b) = 1$. Furthermore, $p^T \tilde{x} \geqslant p^T x$ because of $\kappa \geqslant 1$.

In the remainder of this proof we consider an MWCSP instance I with $|T_p| = 3$.

Claim 1. There is an optimal LP solution (x, y) to $ESA^+(I)$ such that $x(t) = 1$ for a $t \in T_p$.

Proof. Let $\{a, b, c\} := T_p$ such that $p(a) \geqslant \max\{p(b), p(c)\}$. Again, assume $y((r, a)) = x(a)$. Thus, also $y(\delta_D^-(a)) = 0$. Suppose that $y((r, t)) < 1$ for all $t \in T_p$. Note that $y((r, b)) > 0$ or $y((r, c)) > 0$ (otherwise $y((r, a)) = 1$). Assume w.l.o.g. $y((r, b)) > 0$.

Because of $p(a) \geqslant p(c)$ and $y(\delta_D^-(a)) = 0$, we can assume by a flow argument similar to that of Lemma 3.10 that $y((r,c)) = 0$ holds. Similarly, we can assume that there is a flow f_b^c from r to c with $v(f_b^c) = f_b^c((r,b)) = y((r,b))$ and $f_b^c \leqslant y$—otherwise, we decrease $y((r,b))$ and increase $y((r,a))$. Let f_a^b and f_a^c be maximum flows from a to b and c with $f_a^b \leqslant y$ and $f_a^c \leqslant y$. If $v(f_a^b) = v(f_a^c) = 0$, we are effectively in the case $|T_p| \leqslant 2$, since we can restrict the problem to the support graph of (x, y).

So assume $v(f_a^b) > 0$ or $v(f_a^c) > 0$. Thus, $x(a) > 0$. Suppose $x(b) < 1$ and $x(c) < 1$. Note that either $v(f_a^b) = 0$, or both $v(f_a^b) > 0$ and $v(f_a^c) > 0$. First, suppose $v(f_a^b) = 0$. Thus, $y(\delta^-(b)) = 0$ and

$$\sum_{v \in V \setminus \{a,b\}} p(v) y(\delta_D^-(v)) > 0. \tag{3.38}$$

Define $\kappa := \frac{x(a)}{v(f_a^c)}$. Define $\tilde{y} \in \mathbb{R}^{A_r^+}$ by

$$\tilde{y}(e) := \max\{y(e), \kappa f_a^c(e)\}, \tag{3.39}$$

for all $e \in A_r^+$, and define $\tilde{x} \in \mathbb{R}^V$ accordingly. We have $\kappa x(c) = \tilde{x}(c)$. Thus, (3.38) implies $p^T \tilde{x} > p^T x$.

Second, suppose $v(f_a^b) > 0$ and $v(f_a^c) > 0$. In this case, (3.37) holds. Furthermore, $y(\delta_D^-(b)) = f_a^b(\delta_D^-(b))$. Thus, we can proceed as before and multiply both f_a^b and f_a^c by some $\kappa > 1$ to get a better LP solution. Overall, we have shown that $x(b) = 1$ or $x(c) = 1$. □

(Proof of Lemma 3.13 continued.) Let $\{a, b, t\} := T_p$ such that $x(t) = 1$ (which we can assume by Claim 1). Assume $y((r,t)) = x(t)$. In the following we mostly ignore the arcs $\delta^+(r)$ and concentrate on the bidirected graph $D = (V, A)$. We need to show that $x(a), x(b) \in \{0, 1\}$.

Suppose $x(a) \in (0,1)$ or $x(b) \in (0,1)$. Let f^a and f^b be maximum flows from t to a and b with capacity $y(e)$ for each arc $e \in A$. Note that $x(a) = 1$ or $x(b) = 1$; otherwise we could increase f^a, f^b, and y as in the proof of Claim 1. Assume w.l.o.g. $x(a) = 1$. Let $\overline{k} \in \mathbb{R}^A$ with $\overline{k}(e) := \max\{0, f^a(e) - f^b(e)\}$ for all $e \in A$. Let $\check{f}^a$ be a maximum t-a flow with $\check{f}^a \leqslant \overline{k}$.

Claim 2. It holds that $v(\check{f}^a) > 0$.

Proof. Suppose $v(\check{f}^a) = 0$. Let $\overline{V}_t \subset V$ be the set of vertices that can be reached by a directed path from t on the support graph induced by $\overline{k}$ (i.e., via arcs $e \in A$ with $\overline{k}(e) > 0$). Because of $v(\check{f}^a) = 0$, we have $a \notin \overline{V}_t$. Because (x, y) is optimal, we have $b \notin \overline{V}_t$. Because of $a \notin \overline{V}_t$ and $x(a) = 1$ we obtain

$$f^a(\delta^+(\overline{V}_t)) - f^a(\delta^-(\overline{V}_t)) = 1. \tag{3.40}$$

From the definition of $\overline{V}_t$ we get

$$f^b(\delta^+(\overline{V}_t)) \geqslant f^a(\delta^+(\overline{V}_t)). \tag{3.41}$$

Finally, because of $x(b) < 1$ we have

$$f^b(\delta^+(\overline{V}_t)) - f^b(\delta^-(\overline{V}_t)) < 1. \tag{3.42}$$

From (3.40),(3.41), and (3.42) we get

$$f^b(\delta^-(\overline{V}_t)) > f^a(\delta^-(\overline{V}_t)). \tag{3.43}$$

Thus, there is a $u \in \overline{V}_t \setminus \{t\}$ and $e_0 \in \delta^-(u)$ with $f^b(e_0) > f^a(e_0)$; note that $y(e_0) = f^b(e_0)$. By definition of $\overline{V}_t$, there is a directed path P from t to u such that $f^a(e) > f^b(e)$ for all $e \in A(P)$. Let $U \subset V$ with $e_0 \in \delta^-(U)$. If $b \in U$, the existence of P implies $y(\delta^-(U)) > x(b)$ (since we can increase f^b along P). If $a \in U$, we obtain from $y(e_0) > f^a(e_0)$ that $y(\delta^-(U)) > x(a)$. Thus, we can decrease $y(e_0)$ while staying feasible—in contradiction to the minimality or optimality of (x, y). □

(Proof of Lemma 3.13 continued.) We assume $v(\check{f}^a) < 1$. Otherwise the proof would already be complete, since the support graphs of f^a and f^b would be arc disjoint. Let $\hat{f}^a := f^a - \check{f}^a$. Further, for all $e \in A$ define $\tilde{f}^b(e) := \max\{0, f^b(e) - f^a(e)\}$. Note that $\hat{f}^a$ is a flow, but $\tilde{f}^b$ in general not. We further have

$$\sum_{v\in V} p(v)(\hat{f}^a + \check{f}^a + \tilde{f}^b)(\delta^-_D(v)) = \sum_{v\in V\setminus\{t\}} p(v)x(v). \tag{3.44}$$

Claim 3. It holds that

$$\frac{1}{v(\hat{f}^a)} \sum_{v\in V} p(v)(\hat{f}^a + \tilde{f}^b)(\delta^-_D(v)) = \frac{1}{v(\check{f}^a)} \sum_{v\in V} p(v)\check{f}^a(\delta^-_D(v)). \tag{3.45}$$

Proof. Because (x, y) is optimal and $f^a = \hat{f}^a + \check{f}^a$, we obtain from (3.44) that for any sufficiently small $\varepsilon > 0$:

$$\big(1+\frac{\varepsilon}{v(\hat{f}^a)}\big) \sum_{v\in V} p(v)(\hat{f}^a+\tilde{f}^b)(\delta^-_D(v))+\big(1-\frac{\varepsilon}{v(\check{f}^a)}\big) \sum_{v\in V} p(v)\check{f}^a(\delta^-_D(v)) \leqslant \sum_{v\in V\setminus\{t\}} p(v)x(v), \tag{3.46}$$

and

$$\big(1-\frac{\varepsilon}{v(\hat{f}^a)}\big) \sum_{v\in V} p(v)(\hat{f}^a+\tilde{f}^b)(\delta^-_D(v))+\big(1+\frac{\varepsilon}{v(\check{f}^a)}\big) \sum_{v\in V} p(v)\check{f}^a(\delta^-_D(v)) \leqslant \sum_{v\in V\setminus\{t\}} p(v)x(v). \tag{3.47}$$

Note that on the right hand sides of the previous two inequalities we have essentially shifted an ε of the flow f^a from $\check{f}^a$ to $\hat{f}^a$, and vice-versa. Finally,

$$0 \overset{(3.46)}{\leqslant} \frac{\varepsilon}{v(\hat{f}^a)} \sum_{v\in V} p(v)(\hat{f}^a + \tilde{f}^b)(\delta^-_D(v)) - \frac{\varepsilon}{v(\check{f}^a)} \sum_{v\in V} p(v)\check{f}^a(\delta^-_D(v)) \overset{(3.45)}{\leqslant} 0, \tag{3.48}$$

which proves (3.45). □

(Proof of Lemma 3.13 continued.) Set

$$\kappa := \big(1 + \frac{v(\check{f}^a)}{v(\hat{f}^a)}\big). \tag{3.49}$$

Because of (3.45), and (3.44) we obtain

$$\kappa \sum_{v \in V} p(v)(\hat{f}^a + \tilde{f}^b)(\delta^-_D(v)) = \sum_{v \in V \setminus \{t\}} p(v)x(v). \tag{3.50}$$

Define $\tilde{y} \in \mathbb{R}^{A_r^+}$ by

$$\tilde{y}(e) := \max\{y(e), \kappa \hat{f}^a(e), \kappa f^b(e)\}, \tag{3.51}$$

for all $e \in A_r^+$, and define $\tilde{x} \in \mathbb{R}^V$ accordingly. It holds that $p^T\tilde{x} = p^T x$, and $\tilde{x}(c) = 1$. □

As the last piece, we have the now familiar contraction result.

Proposition 3.14. *$v_{LP}(ESA)$ is invariant under the contraction of adjacent vertices of non-negative weight.*

Proof. Let $I = (V, E, p)$ be a MWCSP instance with adjacent $t, u \in V$ such that $p(t) \geqslant 0$, and $p(u) \geqslant 0$. Let $I' = (V', E', p')$ be the MWCSP obtained from contracting t and u. Denote the resulting vertex by t'. I.e., $V' = (V \setminus \{t, u\}) \cup \{t'\}$. Recall that by definition $p'(t') = p(t) + p(u)$. Let $D' := (V'_r, A'_r)$ be the directed graph on which $ESA(I')$ is defined. Let $D = (V_r, A_r)$ be the corresponding graph for $ESA(I)$.

First, we show that $v_{LP}(ESA(I)) \geqslant v_{LP}(ESA(I'))$. One readily verifies that there is an optimal LP solution (x, y) to $v_{LP}(ESA(I))$ such that $x(t) = x(u)$. Due to Lemma 3.10, we can assume that $y((r, t)) = x(t)$. By a similar flow argument, we can further assume that $y((r, u)) = 0$ and $y((t, u)) = x(t)$. Construct an LP solution (x', y') to $ESA(I')$: First, set $y'(a) := y(a)$ for all $a \in A'_r \cap A_r$, and $y'(a) := 0$ for all $a \in \delta^-_{D'}(t') \setminus \{(r, t')\}$. For any $a = (t', v) \in \delta^+_{D'}(t')$ proceed as follows. If $(t, v), (u, v) \in A_r$, set $y'(a) := y((t, v)) + y((u, v))$; if $(t, v) \notin A_r$, set $y'(a) = y((u, v))$; otherwise, set $y'(a) = y((t, v))$. Because of $y(\delta^-(v)) \leqslant 1$, we have in any case that $y'(a) \leqslant 1$. Finally, set $y'((r, t')) := x(t)$. Define $x'(v) := y'(\delta^-_{D'}(v))$ for all $v \in V'$. Note that $x'(v) = x(v)$ for all $v \in V \setminus \{t, u\}$, and $x'(t') = x(t)$; thus, $p^T x' = p^T x$. The feasibility of (x', y') can be seen by noting that any flow $f_q \leqslant y$ in D from either t or u to any $q \in V \setminus \{t, u\}$ can be transformed to a flow $f'_q \leqslant y'$ from t' to q in D' such that $v(f_q) = v(f'_q)$.

Finally, we show that $v_{LP}(ESA(I)) \leqslant v_{LP}(ESA(I'))$. Given an optimal LP solution (x', y') to $ESA(I')$ with $y'((r', t')) = x'(t')$, we define a corresponding LP solution (x, y) to $ESA(I)$. First, $y((t, u)) := x'(t'), y((u, t)) := 0$. Second, $y(a) := y'(a)$ for all $a \in A'_r \cap A_r$, and $y(a) := 0$ for all $a \in \delta^-(\{t, u\}) \setminus \{(r, t)\}$, and $y((r, t)) = x'(t')$. Next, consider the remaining edges $\delta^+(\{t, u\})$. If $(t, v), (u, v) \in A$ set $y((t, v)) := y'(t', v)$, $y((u, v)) := 0$; otherwise, for $a = (t, v)$ or $a = (u, v)$ set $y(a) = y'((t', v))$. □

We now reach the main result of this section.

Theorem 3.15. *If $\alpha(G) \leqslant 3$, then $proj_x(\mathcal{P}_{LP}(ESA))$ is integral.*

Proof. Let $I = (G, p)$ be an MWCSP with $\alpha(G) \leqslant 3$. Let $I' = (V', E', p')$ be the MWCSP obtained from I by contracting all adjacent vertices of non-negative weight. Let A' be the bidirected equivalent of E'. Proposition 3.14 implies that $v_{LP}(ESA(I)) = v_{LP}(ESA(I'))$. Also, I' satisfies $|T'_p| \leqslant 3$ and the vertices T'_p are independent. By Lemma 3.13 and Lemma 3.10 there is an optimal LP solution $(\tilde{x}, \tilde{y})$ to $ESA(I')$ such that $x(u) \in \{0, 1\}$ for all $u \in T'_p$, and $y((r,t)) = 1$ for one $t \in T'_p$. Consider the RMWCSP $I'_t = (V', E', T'_f, p')$ with $T'_f := \{t\}$. For simplicity, we deviate from the assumption that fixed terminals have 0 weight. It holds that $v(ESA(I')) = v(RSA(I'_t))$ and $v_{LP}(ESA(I')) = v_{LP}(RSA(I'_t))$. We will show that

$$v_{LP}(RSA(I'_t)) = v(RSA(I'_t)), \tag{3.52}$$

which concludes the proof. Let (x, y) be the restriction of $(\tilde{x}, \tilde{y})$ to (V', A'). Note that (x, y) is an optimal LP solution to $RSA(I'_t)$. Suppose that (3.52) does not hold. Thus, by Lemma 3.4 there is a $v \in V' \setminus (T'_p \cup T'_f)$ with

$$y(\delta^+(v)) < y(\delta^-(v)). \tag{3.53}$$

The case $|T'_p| < 3$ can be readily ruled out by a flow argument. So assume $|T'_p| = 3$. Because of $\alpha(G) \leqslant 3$, at least one vertex $u \in T'_p$ is adjacent to v. Recall that $x(u) \in \{0, 1\}$. If $x(u) = 0$, we reduce the problem to the support graph of (x, y), which corresponds to the case $|T'_p| < 3$. So assume $x(u) = 1$. If $u \neq t$, define the RMWCSP $I'_u = (V', E', T''_f, p')$ with $T''_f := \{t, u\}$. Further, construct an optimal solution $(x, \tilde{y})$ to I'_u with root u analogously to Lemma 3.10. In this way, $\tilde{y}(\delta^+(v)) < \tilde{y}(\delta^-(v))$ holds again (for the same v as above). In the following, assume $u = t$. Define a new LP solution (x', y') from y as follows. For $a_0 := (t, v)$ set $y'(a_0) := y(\delta^+(v))$. For any $a \in \delta^-(v) \setminus \{a_0\}$ set $y'(a) := 0$. For all $a \in A' \setminus \delta^-(v)$ set $y'(a) := y(a)$. Set $x'(v) := y(\delta^+(v))$, and $x'(w) := x(w)$ for all $w \in V \setminus \{v\}$. By construction of I'_t it holds that $p(v) < 0$ (otherwise, v would have been contracted into u). Thus, $p'^T x' > p'^T x$. The feasibility of (x', y') can be seen as in the proof of Theorem 2.7. □

Note that there are graphs with $a(G) = 4$, such that $proj_x(\mathcal{P}_{LP}(ESA))$ is not integral. For an example, extend the graph in Figure 3.1 as follows. Add a new vertex v and edges between v and the (three) vertices of negative weight shown in the figure.

3.2.4 Comparison of the formulations

A result from Álvarez-Miranda et al. (2013a) states that the directed equivalents of ESA and (a slight generalization of) $NCut$ induce the same polyhedral relaxation of the directed connected subgraph polytope. This result suggest that the same relation holds for the undirected case. Unfortunately, the result from Álvarez-Miranda et al. (2013a) is not correct (the proof suffers from a similar problem as that discussed in Appendix A.2.2 for the rooted case). The strict inclusion result given in the next proposition can indeed also be extended to the directed case.

Proposition 3.16. *The following inclusion holds and can be strict:*

$$proj_x(\mathcal{P}_{LP}(ESA)) \subset \mathcal{P}_{LP}(NCut). \tag{3.54}$$

Proof. Let $(x, y) \in \mathcal{P}_{LP}(ESA)$ and let $a, b \in V$, $a \neq b$. Let $C \in \mathcal{C}(a, b)$ and let U_a be the connected component in the graph $(V \setminus C, E[V \setminus C])$ with $a \in U_a$. Define $\bar{U}_b := V \setminus U_a$ and $\bar{U}_a := U_a \cup C$. Because of $\bar{U}_a \cap \bar{U}_b = C$, one obtains

$$y(\delta^-(\bar{U}_a)) + y(\delta^-(\bar{U}_b)) = y(\delta^-(\bar{U}_a \cup \bar{U}_b)) + y(\delta^-(C)), \tag{3.55}$$

where we use $\delta^- := \delta^-_{D_r}$. Thus,

$$\begin{aligned} x(a) + x(b) &\overset{(3.30)}{\leqslant} y(\delta^-(\bar{U}_a)) + y(\delta^-(\bar{U}_b)) && (3.56)\\ &\overset{(3.55)}{=} y(\delta^+(r)) + y(\delta^-(C)) && (3.57)\\ &\overset{(3.31)}{\leqslant} 1 + x(C). && (3.58) \end{aligned}$$

An example for a strict inclusion is given in Figure 3.1. E.g., consider the following point that is in $\mathcal{P}_{LP}(NCut)$, but not in $proj_x(\mathcal{P}_{LP}(ESA))$: Set the values of all negative weight node variables to 0.5 and the remainder to 1. $\square$

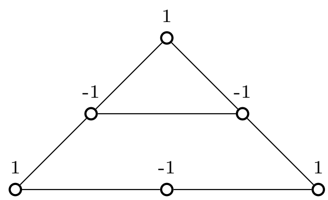


Figure 3.1: MWCSP instance with given node weights.

Next, we consider the indegree constraints. Following Wang et al. (2017), we define

$$\mathcal{Q}' := \{x \in \mathbb{R}^V_{\geqslant 0} \mid \text{x satisfies all indegree constraints}\}. \tag{3.59}$$

While $\mathcal{Q}' \nsubseteq \mathcal{P}_{LP}(NCut)$ and $\mathcal{P}_{LP}(NCut) \nsubseteq \mathcal{Q}'$, see e.g. Wang et al. (2017), the indegree constraints cannot improve the ESA formulation, as the following proposition shows.

Proposition 3.17. *The following inclusion holds and can be strict:*

$$proj_x(\mathcal{P}_{LP}(ESA)) \subset \mathcal{Q}'. \tag{3.60}$$

Proof. Consider an undirected graph G, and let D be its bidirected equivalent. Furthermore, let D_r be the extended, directed graph on which ESA is defined. Let

$(x, y) \in \mathcal{P}_{LP}(ESA)$. First, note that constraints (3.29) and (3.30) imply for all $\{v, w\} \in E$ that

$$\min\{y(\delta^-_{D_r}(v)), y(\delta^-_{D_r}(w))\} \geqslant y((v, w)) + y((w, v)). \tag{3.61}$$

Let d be an indegree vector. It holds that

$$\begin{aligned}
\sum_{v \in V} x(v) &= \sum_{a \in A_r} y(a) \\
&\leqslant \sum_{a \in A} y(a) + 1 \\
&= \sum_{\{v,w\} \in E} \big(y((v, w)) + y((w, v))\big) + 1 \\
&\overset{(3.61)}{\leqslant} \sum_{\{v,w\} \in E} \min\{y(\delta^-_{D_r}(v)), y(\delta^-_{D_r}(w))\} + 1 \\
&\leqslant \sum_{v \in V} d_v x(v) + 1,
\end{aligned}$$

which implies that (3.27) is satisfied by x; thus, $x \in \mathcal{Q}'$. For a strict inclusion consider the graph in Figure 3.1 and the point x as defined in the proof of Proposition 3.16. □

Summarizing the results of this section, one obtains:

Theorem 3.18. *It holds that*

$$proj_x(\mathcal{P}_{LP}(ESA)) \subset \mathcal{Q}' \cap \mathcal{P}_{LP}(NCut), \tag{3.62}$$

and the inclusion can be strict.

Finally, note that by using one flow for each vertex, similar to the DF formulation, it is also possible to obtain a compact extended formulation for the connected subgraph polytope that is equivalent to ESA—and thus (strictly) stronger than the combined node-separator and indegree formulation.

3.3 Reduction techniques

Reduction techniques are a vital component for practical exact solution of MWCSP, see El-Kebir and Klau (2014); Leitner et al. (2018a), and also for many other combinatorial optimization problems, such as SPG, see Section 2.3 or maximum clique, see Verma et al. (2015). Still, reduction techniques for the MWCSP have only been recently addressed in the literature. The first ground was broken in the course of the 11th DIMACS Challenge, with two articles Althaus and Blumenstock (2014); El-Kebir and Klau (2014) containing reduction techniques as part of an exact solving approach. Later, a dual-ascent-based branch-and-bound algorithm with strong reduction properties was described in Leitner et al. (2018a). Also, several other authors, e.g. Loboda et al. (2016); Álvarez Miranda and Sinnl (2017); Wang et al. (2017), have used simple reduction techniques such as the contraction of adjacent vertices of positive weight.

Note that an optimal solution may consist of a single vertex. Thus, care needs to be taken to avoid the deletion or modification of an optimal positive weight vertex. Indeed, one finds wrong reduction tests in literature that disregard this observation, as detailed in Rehfeldt et al. (2019). An example is the contraction of a positive weight vertex of degree 1 into its (negative weight) neighbor. Also, just remembering a maximum-weight vertex before the start of the reduction techniques, as suggested in the literature, is not sufficient: A single-vertex solution might be created during the reduction process on a reduced problem. Indeed, guarding measures during the reduction process can hardly be avoided due to the following proposition. It can be readily proven by a reduction from the decision variant of MWCSP.

Proposition 3.19. *Deciding whether no single-vertex maximum-weight connected subgraph exists is $\mathcal{NP}$-hard.*

In the following, we introduce a number of MWCSP reduction techniques that are both theoretically and practically stronger than those described by other authors in the literature so far. These techniques can also be applied to RMWCSP if sufficiently high weights for each fixed terminal are used. We note that a reduction technique not implied by the methods introduced in the following is the contraction of chains of non-positive vertices, see El-Kebir and Klau (2014).

To render the proof techniques more perspicuous, throughout this section it will without loss of generality be assumed that each solution to I_{MW} is given as a tree (and not as an arbitrary connected subgraph).

3.3.1 Bound-based reductions

The term *bound-based reductions* describes preprocessing methods that identify edges and vertices for elimination by examining whether they induce an upper bound that is lower than a given lower bound (or vice versa), see e.g. Duin (1993); Hwang et al. (1992). In the following, we will introduce a new bound-based MWCSP reduction technique. This technique can be seen as an adaptation of the terminal-regions decomposition method for SPG that we developed in Section 2.3.2.

The base of the reduction technique is the following, new, concept: a *positive-vertex decomposition* of I_{MW}—with underlying graph (V, E)—is a partition $H = \{H_{t_i} \subseteq V \mid T_p \cap H_{t_i} = \{t_i\}\}$ of V such that for each $t_i \in T_p$ the subgraph $(H_{t_i}, E[H_{t_i}])$ is connected. Each of the H_{t_i} is called *region* with *center* t_i. Furthermore, a vertex $v_j \in H_{t_i}$ adjacent to a vertex $v_k \notin H_{t_i}$ is called *boundary vertex* of region H_{t_i}; the set of all such vertices to a region H_{t_i} will be denoted by $B(H_{t_i})$. Additionally, an edge $\{v_i, v_j\}$ with v_i and v_j in different regions will be called *H-boundary edge*.

To set the stage for the computation of an upper bound, define for all $t_i \in T_p$ the *positive-vertex decomposition radii*:

$$r_H(t_i) := \max\{\overline{d}(t_i, v_k) \mid v_k \in B(H_{t_i})\} \tag{3.63}$$

and

$$r_H^+(t_i) := \max\{r_H(t_i), 0\}. \tag{3.64}$$

Definition (3.64) allows us to establish three bound-based reduction criteria presented in the following. An important observation underlying all these criteria is that for each positive vertex t_i that is part of an optimal solution S with $|V(S) \cap T_p| \geqslant 2$ there needs to be a path in $V(S) \cap H_{t_i}$ from t_i to a vertex in $B(H_{t_i})$—and the weight of this path is bounded by $r_H(t_i)$. Since t_i does not have to be in $V(S)$, one cannot use $r_H(t_i)$ to obtain a bound on the weight of S; however, one can use $r_H^+(t_i)$ instead. Moreover, one can observe that if a negative vertex v_i is part of an optimal solution, there need to be two paths in S connecting v_i to positive vertices and having no vertices but v_i in common. These two observations lead to:

Proposition 3.20. *Let H be a positive-vertex decomposition of I_{MW} and assume that $|T_p| \geqslant 2$. Furthermore, let $v_i \in V \setminus T_p$ and assume that for each optimal solution S to I_{MW} it holds that $v_i \in V(S)$. Finally, let*

$$U_2 := \sum_{t \in T_p} r_H^+(t) - \min\{r_H^+(t) + r_H^+(t') \mid t, t' \in T_p,\ t \neq t'\}. \tag{3.65}$$

Thereupon,

$$U := U_2 + \overline{d}(v_i, \overline{v}_{i,1}) + \overline{d}(v_i, \overline{v}_{i,2}) - p(v_i) \tag{3.66}$$

is an upper bound on the weight of S.

Proof. Let S be an optimal solution to I_{MW} such that $v_i \in V(S)$. As before, it is assumed (without limiting generality) that S is a tree. Denote the (unique) path in S between v_i and a $t_j \in V(S) \cap T_p$ by Q_j and set $\mathcal{Q} := \{Q_j \mid t_j \in V(S)\}$. First, note that $|\mathcal{Q}| \geqslant 2$, because if $\mathcal{Q}$ just contained one path, say Q_j, it would follow for $S' := \{t_j\}$ that $v_i \notin S'$ and $P(S') \geqslant P(S)$ (which contradicts the assumptions of the proposition). Second, if a vertex v_k is contained in two distinct paths of $\mathcal{Q}$, the subpaths of these two paths between v_i and v_k coincide. Otherwise there would need to be a cycle in S. Additionally, there are at least two (distinct) paths $Q_k, Q_l \in \mathcal{Q}$ such that $V(Q_k) \cap V(Q_l) = \{v_i\}$. Otherwise, due to the precedent observation, all paths in $\mathcal{Q}$ would have one edge $\{v_i, v_i'\}$ in common, which could be discarded to obtain a tree S' with $v_i \notin V(S')$ and $P(S') \geqslant P(S)$.

Now, choose two distinct paths $Q_k \in \mathcal{Q}$ and $Q_l \in \mathcal{Q}$ with minimum number of combined H-boundary edges and $V(Q_k) \cap V(Q_l) = \{v_i\}$. Further, define $\mathcal{Q}^- := \mathcal{Q} \setminus \{Q_k, Q_l\}$. For all $Q_r \in \mathcal{Q}^-$, denote by Q_r' the subpath of Q_r from t_r up to the last vertex still in H_{t_r}. Suppose that Q_k has a vertex $v_q \in V(S)$ in common with a Q_r'. Consequently, $Q_l \cap Q_r = \{v_i\}$, because S is cycle-free. Furthermore, according to the preceding observations, Q_k and Q_r have to contain a joint subpath including v_i and v_q. But this implies that Q_k contains at least one additional H-boundary edge (in order to be able to reach t_k, which is by definition not in H_{t_r}). Therefore, and due to $V(Q_l) \cap V(Q_r) = \{v_i\}$, the path Q_r would have initially been selected instead of Q_k.

Following the same line of argumentation, one validates that likewise Q_l has no vertex in common with any Q'_r. Conclusively, the paths Q_k, Q_l have only the vertex v_i in common and all paths Q'_r are vertex disjoint and also do not have any vertex in common with both Q_k, Q_l. Using their combined weight, one can obtain an upper bound on the weight of S by:

$$
\begin{aligned}
P(S) = &\sum_{v \in V(S)} p(v) \\
&\leqslant \Big(\sum_{Q_r \in \mathcal{Q}^-} P(Q'_r) \Big) + P(Q_k) + P(Q_l) - p(v_i) \\
&\leqslant \sum_{t \in T_p} r^+_H(t) - \min\{r^+_H(t) + r^+_H(t') \mid t, t' \in T_p,\ t \neq t'\} + P(Q_k) + P(Q_l) - p(v_i) \\
&\leqslant \sum_{t \in T_p} r^+_H(t) - \min\{r^+_H(t) + r^+_H(t') \mid t, t' \in T_p,\ t \neq t'\} + \overline{d}(v_i, \overline{v}_{i,1}) + \overline{d}(v_i, \overline{v}_{i,2}) \\
&\quad - p(v_i).
\end{aligned}
$$

The first inequality follows from above discussed properties of the paths in Q^- and the paths Q_k and Q_l. The second inequality uses the fact that the weight of each path $Q_r \in Q^-$ can be bounded from above by $r^+_H(t_r)$. Finally, the third inequality exploits that the paths Q_k and Q_l do not contain any intermediate vertices of positive weight and that there weight can therefore be bounded by using the distance function $\overline{d}$. Consequently, the proposition is proven. □

It follows from the proposition that vertex of non-positive weight can be eliminated if the associated upper bound U in (3.66) is smaller than a known lower bound (e.g. the weight of a given feasible solution). An application of the proposition is exemplified in Figure 3.2 for a simple MWCSP instance with positive vertices t_1, t_2, t_3—the weights are given next to the corresponding vertices. With H being the positive-vertex decomposition marked by the dotted ellipses it holds that $r^+_H(t_1) = 1$, $r^+_H(t_2) = 2$, and $r^+_H(t_3) = 0.5$. Consequently, for Proposition 3.20—with v_i as labeled in the figure—it holds that $U_2 = 2$ and $U = 2.5$. Therefore, v_i can be eliminated if a lower bound higher than 2.5 is given.

The following proposition can be used to moreover eliminate vertices of positive weight. It can be proven similarly to Proposition 3.20 (see Appendix A.2.1).

Proposition 3.21. *Let H be a positive-vertex decomposition of I_{MW} and assume that $|T_p| \geqslant 2$. Furthermore, let $v_i \in T_p$ and assume that an optimal solution S exists such that $v_i \in V(S)$ and $|V(S) \cap T_p| \geqslant 2$. Define*

$$U_1 := \sum_{t \in T_p \setminus \{v_i\}} r^+_H(t) - \min\{r^+_H(t) \mid t \in T_p \setminus \{v_i\}\}. \tag{3.67}$$

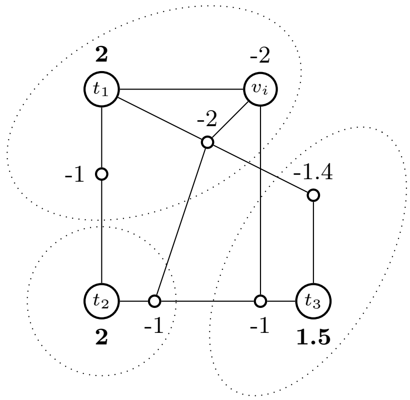


Figure 3.2: A positive-vertex decomposition of an MWCSP instance with regions marked by dotted ellipses.

Then

$$U := U_1 + \overline{d}(v_i, \overline{v}_{i,1}) \tag{3.68}$$

is an upper bound on the weight of S.

The positive vertex decomposition concept does not only allow for direct elimination of vertices, but can furthermore be used for a criterion that guarantees that a vertex cannot be of degree higher than 2 in any optimal solution. This information will be utilized in Section 3.3.2.

Proposition 3.22. *Let H be a positive-vertex decomposition of I_{MW} and assume that $|T_p| \geqslant 3$. Furthermore, let $v_i \in V \setminus T_p$ and assume that for an optimal solution S to I_{MW} it holds that $|\delta_S(v_i)| \geqslant 3$. Finally, let*

$$\begin{aligned} U_3 := &\sum_{t \in T_p} r_H^+(t) \\ &- \min\{r_H^+(t_j) + r_H^+(t_k) + r_H^+(t_l) \mid t_j, t_k, t_l \in T_p;\ t_j, t_k, t_l \text{ disjoint}\}. \end{aligned} \tag{3.69}$$

Thereupon,

$$U := U_3 + \overline{d}(v_i, \overline{v}_{i,1}) + \overline{d}(v_i, \overline{v}_{i,2}) + \overline{d}(v_i, \overline{v}_{i,3}) - 2p(v_i) \tag{3.70}$$

is an upper bound on the weight of S.

To efficiently apply Proposition 3.20, one would like to minimize (3.65)—and for Proposition 3.21 and Proposition 3.22 to minimize (3.67) and (3.69), respectively. Unfortunately, this problem turns out to be $\mathcal{NP}$-hard.

The decision variant of the problem can be stated as follows. Let $\alpha \in \mathbb{N}_0$ and let $G_0 = (V_0, E_0)$ be an undirected, non-empty graph. Furthermore, let $p_0 : V_0 \to \mathbb{Z}$, set $T_0 := \{v \in V_0 \mid p(v) > 0\}$, and assume that $\alpha < |T_0|$. For each positive-vertex decomposition H_0 of G_0 choose $T_0' \subsetneq T_0$ such that $|T_0'| = \alpha$ and $r^+_{H_0}(t') \leqslant r^+_{H_0}(t)$ for all $t' \in T_0'$ and $t \in T_0 \setminus T_0'$. Let:

$$C_{H_0} := \sum_{t \in T_0 \setminus T_0'} r^+_{H_0}(t). \tag{3.71}$$

We now define the α-positive-vertex decomposition problem as follows: Given a $k \in \mathbb{N}$, is there a positive-vertex decomposition H_0 such that $C_{H_0} \leqslant k$? In the following proposition it is shown that this problem is $\mathcal{NP}$-complete, which forthwith establishes the $\mathcal{NP}$-hardness of finding a positive-vertex decomposition that minimizes (3.65), (3.67), or (3.69)—which corresponds to $\alpha = 2$, $\alpha = 1$, and $\alpha = 3$, respectively.

Proposition 3.23. *For each $\alpha \in \mathbb{N}_0$ the α-positive-vertex decomposition problem is $\mathcal{NP}$-complete.*

Proof. Given a positive-vertex decomposition H_0 it can be tested in polynomial time whether the associated C_{H_0} is less than or equal to k. This can be be done for instance as follows: Consider the set of (directed) arcs $A' := \{(v, w) \in V_0 \times V_0 \mid \{v, w\} \in E\}$ and define edge costs $c' : A' \to \mathbb{Z}_{\geqslant 0}$ such that for $a = (v_i, v_j) \in A'$:

$$c'(a) = \begin{cases} -p_0(v_j), & \text{if } p_0(v_j) < 0 \\ 0, & \text{otherwise} \end{cases}$$

Thereupon, C_{H_0} can be computed by running (the directed version of) Dijkstra's algorithm for each subgraph $(H_{t_i}, A'[H_{t_i}])$, starting from t_i and using the arcs costs c'. Consequently, the positive-vertex decomposition problem is in $\mathcal{NP}$.

Next, it will be shown that the, $\mathcal{NP}$-complete (Garey and Johnson, 1979), independent set problem can be reduced to the positive-vertex decomposition problem. To this end, let $G_{ind} = (V_{ind}, E_{ind})$ be an undirected, non-empty graph and $k \in \mathbb{N}$. The problem is to determine whether an independent set in G_{ind} of cardinality at least k exists. Without loss of generality it will be assumed that G_{ind} does not include any vertices of degree 0.

To establish the reduction, construct a graph G_0 from G_{ind} as follows. Initially, set $G_0 = (V_0, E_0) := G_{ind}$ and define vertex weights $p_0(v_i) := 1$ for all $v_i \in V_0$. Next, extend G_0 by replacing each edge $e_l = \{v_i, v_j\} \in E_0$ with a vertex v_l' of weight $p_0(v_l') = -1$ and the two edges $\{v_i, v_l'\}$ and $\{v_j, v_l'\}$. Finally, if $\alpha > 0$, choose an arbitrary $v_0 \in V_0 \cap V_{ind}$ and add vertices v_j' of weight -1 for $j = 0, ..., \alpha$ and vertices v_j'' of weight 1 for $j = 1, ..., \alpha$ to G_0. Additionally, add an edge $\{v_0, v_0'\}$. Finally, add edges $\{v_0', v_j'\}$ and $\{v_j', v_j''\}$ for $j = 1, ..., \alpha$.

First, one observes that the size $|V_0| + |E_0|$ of the new graph G_0 is a polynomial in the size $|V_{ind}| + |E_{ind}|$ of G_{ind}. Next, $r^+_{H_0}(v_i) = 0$ holds for a vertex $v_i \in G_0 \cap G_{ind}$ if and only if H_{v_i} contains all (newly inserted) adjacent vertices of v_i in G_0. The latter condition implies that for each adjacent vertex v_k of v_i in $G_0 \cap G_{ind}$ it holds that $r^+_{H_0}(v_k) = 1$. Moreover, in a positive-vertex decomposition for (G_0, p_0) of minimum cost C_{H_0}, it holds that $r^+_{H_0}(v''_j) = 0$ for $j = 1, ..., \alpha$. Hence, there is an independent set in G_{ind} of cardinality at least k if and only if there is a positive-vertex decomposition H_0 for (G_0, p_0) such that

$$C_{H_0} \leqslant |V_{ind}| - k.$$

This proves the proposition. □

Since attempting to find an exact polynomial time algorithm for minimizing (3.65) seems to be overly optimistic, a greedy heuristic based on Dijkstra's algorithm will instead be used. Moreover, a local search heuristic has been developed to improve the decomposition found by the greedy approach. The combined algorithm runs in $O(m \log n)$, which also gives the whole bound-based reduction test a worst-case complexity of $O(m \log n)$—if a lower bound is already available. This reduction test will be referred to as *Positive Vertex Decomposition* (PVD) test; the computation of a lower bound will be discussed in Section 3.5.

3.3.2 Alternative-based reductions

This section covers several *exclusion tests* (Duin, 1993): reduction methods that attempt to prove that a specified part of the problem graph—usually a single vertex or edge—is not contained in at least one optimal solution. The usual procedure is to show that for each solution that contains this specified subgraph there is another, alternative, solution of equal or better objective value that does not. A simple example for an exclusion test is the following. Delete any edge $\{v, w\} \in E$, such that there is a $t \in V$ with $p(t) \geqslant 0$, and edges $\{v, t\}, \{w, t\} \in E$. The reasoning is as follows: For any connected subgraph that contains $\{v, w\}$, we can remove $\{v, w\}$ and add both $\{v, t\}, \{w, t\}$. The result is a connected subgraph of at least the same weight. See also Rehfeldt et al. (2019).

Constrained walks

In the following, we again apply our new concept of walk-based distance measures from the previous chapter. Let $v, w \in V$. A finite walk $W = (v_1, e_1, v_2, e_2, ..., e_{r-1}, v_r)$ with $v_1 = v$ and $v_r = w$ will be called *positive-weight constrained* (v,w)*-walk* if no $u \in T_p \cup \{v, w\}$ is contained more than once in W. For any $k, l \in \mathbb{N}$ with $1 \leqslant k \leqslant l \leqslant r$ define the subwalk $W(v_k, v_l) := (v_k, e_k, v_{k+1}, e_{k+1}, ..., e_{l-1}, v_l)$. In the following, let W be a positive-weight constrained (v, w)-walk. Define the *interior cost* of W as:

$$C^-(W) := \sum_{u \in V(W) \setminus \{v,w\}} p(u), \tag{3.72}$$

where the convention that the empty sum equals 0 is assumed, so the interior cost of an edge is likewise 0. Furthermore, define the *positive-weight constrained length* of W as:

$$l_{pw}(W) := \min\{C^-(W(v_k, v_l)) \mid 1 \leqslant k \leqslant l \leqslant r,\ v_k, v_l \in T_p \cup \{v, w\}\}. \tag{3.73}$$

Note that $l_{pw}(W) \leqslant 0$ holds, because the interior cost of an edge is 0. For a motivation of the positive-weight constrained length consider two vertex-disjoint solutions S_1 and S_2 to I_{MW}, and a walk W that joins S_1 and S_2. Thereupon, one can obtain a new solution S_3 out of S_1, S_2, and W such that

$$P(S_1) + P(S_2) + l_{pw}(W) \leqslant P(S_3). \tag{3.74}$$

Denote the set of all positive-weight constrained (v, w)-walks by $\mathcal{W}_{pw}(v, w)$ and define the *positive-weight constrained distance* between v and w as

$$d_{pw}(v, w) := \max\{l_{pw}(W) \mid W \in \mathcal{W}_{pw}(v, w)\}. \tag{3.75}$$

For the next results, let G be the underlying graph of I_{MW}, and consider the distance network $D_G(U, -d_{pw})$ for a given set $U \subseteq V$. For simplicity, we write $D(U, -d_{pw}) := D_G(U, -d_{pw})$ in the following. See Section 1.1.2 for the definition of a distance network. Because $d_{pw}(v, w) \leqslant 0$ for any $v, w \in V$, the distance network $D(U, -d_{pw})$ has non-negative edge weights. Based on these concepts, we introduce two new edge elimination criteria in the following.

Proposition 3.24. *Let $e = \{v, w\} \in E$ with $p(v) \leqslant 0$. Define $\Delta := (T_p \cup \{w\}) \cap N(v)$, and*

$$\mathcal{U} = \big\{U \subseteq N(v) \mid |U| \geqslant 2, U \supseteq \Delta\big\}.$$

If for all $U \in \mathcal{U}$ the weight of a minimum spanning tree on $D(U, -d_{pw})$ is smaller than $-p(v)$, then at least one optimal solution does not contain edge e.

Proof. Let S be a connected subgraph with $e \in E(S)$. We will show that there is a connected subgraph S' with $e \notin E(S')$ such that

$$P(S) \leqslant P(S'). \tag{3.76}$$

We can assume that S is a tree, and that $\Delta \subseteq N_S(v)$. Otherwise, we can modify S to satisfy these conditions and still contain e without decreasing the weight of S. Note that if v is of degree 1 in S, we can simply delete edge e to obtain the desired S'. So assume $|N_S(v)| \geqslant 2$.

Let $U := N_S(v)$. Let $\hat{S}$ be the subgraph obtained from S by removing vertex v and all incident edges. Let $S^{(1)}, ..., S^{(k)}$ be the (inclusion-wise maximal) connected components of $\hat{S}$. Note that $k = |U|$. Let F_U be a minimum spanning tree on $D(U, -d_{pw})$, and denote its weight by $-C_U$ (recall that d_{pw} is non-positive). By the assumption of the proposition it holds that

$$C_U - p(v) > 0. \tag{3.77}$$

Assume that the $S^{(i)}$ are ordered such that for each $i \in \{2, ..., k\}$ there are vertices $q^{(i)} \in V(\bigcup_{h \leqslant i} S^{(h)}) \cap U$ and $r^{(i)} \in V(S^{(i+1)}) \cap U$ such that there is a $(q^{(i)}, r^{(i)})$-walk $W^{(i)}$ in (V, E) corresponding to an edge in the spanning tree F_U. Note that $l_{pw}(W^{(i)}) = d_{pw}(q^{(i)}, r^{(i)})$. Set $\hat{S}^{(1)} := S^{(1)}$ and proceed for $i = 2, ..., k$ as follows.

First, observe that $v \notin V(W^{(i)})$ due to the assumptions of the proposition. Let $v_1, v_2, ..., v_s$ be the vertices encountered (in this order) when traversing $W^{(i)}$ from $q^{(i)}$ to $r^{(i)}$. So in particular $v_1 = q^{(i)}$ and $v_s = r^{(i)}$. Let b be the minimum number such that $v_b \in V(S^{(i)})$. Further, let a be the largest number in $\{1, 2, .., b\}$ such that $v_a \in V(\hat{S}^{(i-1)})$. Further, define $x := \max\{j \in \{1, ..., a\} \mid v_j \in T_p \cup \{v_1\}\}$ and $y := \min\{j \in \{b, ..., s\} \mid v_j \in T_p \cup \{v_s\}\}$. By definition, $x \leqslant a < b \leqslant y$ and furthermore:

$$C^-(W^{(i)}(v_a, v_b)) \geqslant C^-(W^{(i)}(v_x, v_y)) \geqslant l_{pw}(W^{(i)}) = d_{pw}(q^{(i)}, r^{(i)}). \tag{3.78}$$

Define $\hat{S}^{(i)} := \hat{S}^{(i-1)} \cup S^{(i)} \cup W^{(i)}(v_a, v_b)$, with a slight abuse of notation $W^{(i)}(v_a, v_b)$ is considered as a subgraph here. Ultimately, $S' := \hat{S}^{(k)}$ is a connected subgraph and it holds that

$$P(S') \overset{(3.78)}{\geqslant} P(\hat{S}) + C_U = P(S) + C_U - p(v) \overset{(3.77)}{>} P(S). \tag{3.79}$$

Because $v \notin V(S')$ implies $e \notin E(S')$, the proposition is proven. □

Another reduction test can be obtained by splitting the neighborhood of the edge considered for elimination, as detailed in the following proposition.

Proposition 3.25. *Let $e = \{v, w\} \in E$. Assume that $p(v) + p(w) \leqslant 0$. Define $\Delta := (T_p \cap N(e))$. Further, define*

$$\mathcal{U} = \big\{U \subseteq N(e) \mid U \supseteq \Delta, U \cap (N(v) \setminus \{w\}) \neq \emptyset, U \cap (N(w) \setminus \{v\}) \neq \emptyset\big\}.$$

If for all $U \in \mathcal{U}$ the weight of a minimum spanning tree on $D(U, -d_{pw})$ is smaller than $-(p(v) + p(w))$, then no optimal solution contains edge e.

The proposition can be proved in a similar way to the previous one. Note that both propositions can be extended to the case of equality if the walks corresponding to positive-weight constrained distances do not contain edge e. The reduction test obtained from the previous two propositions strictly dominates shortest-path-based reduction methods described in the literature (El-Kebir and Klau, 2014; Leitner et al., 2018a).

We also note that the above two propositions can be strengthened for vertices that have been shown to be of degree at most 2 in an optimal solution. Such information can for example be obtained by using Proposition 3.22. We give more details in Rehfeldt and Koch (2019).

In this thesis, heuristics are employed to compute lower bounds on the positive-weight constrained distance. To justify the use of heuristics, it will be demonstrated that computing the positive-weight constrained distance is $\mathcal{NP}$-hard. This pessimistic worst-case complexity does not come as a surprise, since already a weaker corresponding concept for the prize-collecting Steiner tree problem is $\mathcal{NP}$-hard, as shown in Uchoa (2006). See Chapter 4 for more details.

First, the decision variant of the positive-weight constrained distance is defined. Let $G_0 = (V_0, E_0)$ be an undirected and connected graph with $|V_0| \geqslant 2$. Furthermore, let $p_0 : V_0 \to \mathbb{Z}$. Given two distinct vertices $v, w \in V_0$ and a $k \in \mathbb{Z}_{\leqslant 0}$, the positive-weight constrained distance problem is to determine whether $d_{pw}(v, w) \geqslant k$. The $\mathcal{NP}$-hardness of the problem can be shown by a reduction from the Hamiltonian path problem—as in the $\mathcal{NP}$-hardness proof of the bottleneck Steiner distance for the prize-collecting Steiner tree problem (Uchoa, 2006).

Proposition 3.26. *The positive-weight constrained distance problem is $\mathcal{NP}$-complete.*

Proof. First, note that the positive-weight constrained length of a given path Q can be computed in $O(|V_0(Q)|^2)$; hence, the positive-weight constrained distance problem is in $\mathcal{NP}$. Next, let $G_{Ham} = (V_{Ham}, E_{Ham})$ be an undirected, connected graph with two distinct vertices v, w. The Hamiltonian path problem asks whether a (simple) path between v and w exists that contains all vertices. This problem can be reduced to the positive-weight constrained distance problem as follows. Initially, set $G_0 := (V_0, E_0) := G_{Ham}$ and define $p_0(v) := 1$ for all $v \in V_0$. Next, extend G_0 by adding vertices v', v'' with weights $p_0(v') = -|V_{Ham}|$, $p_0(v'') = 0$ and vertices w', w'' with weights $p_0(w') = -|V_{Ham}|$, $p_0(w'') = 0$ to V_0. Finally, add edges $\{v, v'\}$, $\{v', v''\}$ and $\{w, w'\}$, $\{w', w''\}$ to E_0. Thereupon, G_{Ham} contains an Hamiltonian path between v and w if and only if $d_{pw}(v'', w'') \geqslant -|V_{Ham}|$ on (V_0, E_0, p_0). $\square$

Notwithstanding its $\mathcal{NP}$-hardness, the positive-weight constrained distance can be approximated by heuristics well enough to allow for a strong practical performance of the associated reduction tests.

Dominating connected sets

Besides paths, one can also use general connected subgraphs for alternative-based reductions tests. This chapter introduces the concept of *dominating connected sets* for the MWCSP: Let $X \subset V$ such that $(X, E[X])$ is connected and let $U \subseteq V \setminus X$. Then X will be said to *MWCS-dominate* U if

$$N(U) \subseteq N(X) \cup X$$

Importantly, one can remove U from any feasible solution and reconnect the resulting components by using only vertices of X. In the following, additional conditions will be formulated that allow to remove U, or parts of it, without reducing the weight of at least one optimal solution. The first such condition is stated in the following proposition. This proposition also generalizes a result from El-Kebir and Klau (2014), which states that a vertex v with $p(v) < 0$ can be deleted if there is a $w \in V \setminus \{v\}$ such that $N(v) = N(w)$ and $p(v) \leqslant p(w)$.

Proposition 3.27. *Let $U \subseteq V \setminus T_p$ and $X \subseteq V \setminus U$ such that X MWCS-dominates U and assume*

$$\sum_{u \in U} p(u) \leqslant \sum_{u \in X : p(u) < 0} p(u). \tag{3.80}$$

Then there exists an optimal solution S such that $U \nsubseteq V(S)$. The set X will be said to all-weights MWCS-dominate *U.*

Proof. Let S be a feasible solution with $U \subseteq V(S)$. Note that by construction $p(w) \leqslant 0$ for all $w \in U$. Define

$$\Delta_S := \big\{v \in V(S) \setminus U \mid \exists \{v, w\} \in E(S), w \in U\big\}.$$

Next, remove U from S. In this way one obtains a new (possibly empty) subgraph S' that contains at most $|\Delta_S|$ many (inclusion-wise maximal) connected components. If S' is connected, no further discussion is necessary. Otherwise, note that each connected component of S' contains a vertex $v \in \Delta_S$. Therefore, these components can be reconnected as follows. First, add $X \setminus V(S')$ to $V(S')$ to obtain a new subgraph S''. Second, because X MWCS-dominates U and because each connected component contains a $v \in \Delta_S$, there exists a set of edges $\tilde{E}_{S''} \subseteq E[V(S'')]$ that reconnects S''. Adding $\tilde{E}_{S''}$ to S'', one obtains a, finally connected, subgraph S'''. Finally, the construction of S''' implies:

$$\sum_{u \in V(S''')} p(u) \geqslant \sum_{u \in V(S)} p(u) - \sum_{u \in U} p(u) + \sum_{u \in X : p(u) < 0} p(u) \overset{(3.80)}{\geqslant} \sum_{u \in V(S)} p(u).$$

This concludes the proof. $\square$

While Proposition 3.27 guarantees that set U is not part of at least one optimal solution, the same may not be true for subsets of U. Therefore, one cannot just eliminate U in general. However, in the case of $|U| = 1$ one can forthwith eliminate U, and in the case of $|U| = 2$ with $U = \{v, w\} \in E$ one can eliminate the edge $\{v, w\}$. Figure 3.3 shows an MWCSP instance for which an edge can be eliminated by means of the criterion formulated in Proposition 3.27. The vertices of the dashed edge have a summed weight of -4.3, smaller than the weight of the (sole) negative vertex in the MWCS-dominating set X marked by the upper dotted ellipse (which is -3.5).

In contrast to Proposition 3.27, the following proposition allows to eliminate non-trivial (i.e. larger than single-vertex or single-edge) subgraphs of $(V \setminus T_p, E[V \setminus T_p])$—but also involves a more restricting test condition.

Proposition 3.28. *Let $U \subseteq V \setminus T_p$ and $X \subseteq V \setminus U$ such that X MWCS-dominates U and assume*

$$\max_{w \in U} p(w) \leqslant \sum_{u \in X : p(u) < 0} p(u). \tag{3.81}$$

Then there exists an optimal solution S such that $U \cap V(S) = \emptyset$. The set X will be said to max-weight MWCS-dominate *U.*

Proof. Let S be a feasible solution with $U \cap V(S) \neq \emptyset$. Further, define Δ_S as in the proof of Proposition 3.27. Remove $U \cap V(S)$ from S to obtain a new (possibly empty)

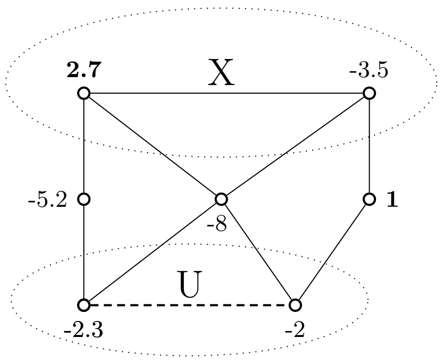


Figure 3.3: An MWCSP instance. Considering the vertices enclosed by the upper dotted ellipse as the set X and those enclosed by the lower one as U, one can verify with Proposition 3.27 that the dashed edge can be deleted.

subgraph S' that contains at most $|\Delta_S|$ many (inclusion-wise maximal) connected components. Assume that there are at least two connected components. Each of these components contains a vertex $v \in \Delta_S$. These components can therefore be reconnected as in the proof of Proposition 3.27 to obtain a connected subgraph S''' with $U \cap V(S''') = \emptyset$. Because of (3.81) it holds for the resulting connected subgraph S''' that $\sum_{v \in V(S''')} p(v) \geqslant \sum_{v \in V(S)} p(v)$. $\square$

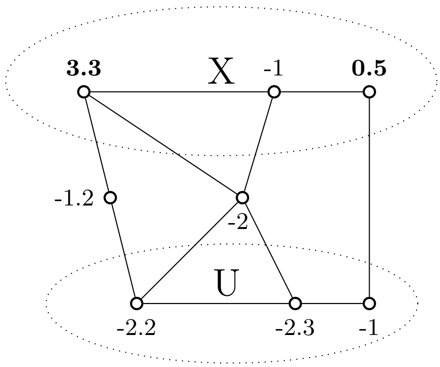


Figure 3.4: An MWCSP instance. Considering the vertices enclosed by the upper dotted ellipse as set X and the ones enclosed by the lower one as U, one can verify with Proposition 3.28 that all vertices of U (and incident edges) can be deleted.

Figure 3.4 shows an MWCSP instance that can be reduced by using Proposition 3.28.

For the special case of $|U| = 1$ a vertex set X max-weight MWCS-dominates a vertex set U if and only if X all-weights MWCS-dominates U. Therefore, such a set will be called *single-weight MWCS-dominating*. As will be shown in the following, already this special case is $\mathcal{NP}$-hard. Let $G_0 = (V_0, E_0)$ be an undirected, non-empty graph. Furthermore, let $p_0 : V_0 \to \mathbb{Z}$. Given a vertex $v \in V_0$ with $p_0(v) \leqslant 0$ the single-weight MWCS-domination problem is to determine whether a subset of $V \setminus \{v\}$ exists that single-weight MWCS-dominates v.

Proposition 3.29. *The single-weight MWCS-domination problem is $\mathcal{NP}$-complete.*

Proof. Given a vertex subset X it can be verified with worst-case complexity of $O(|E_0| + |V_0|)$ whether this is an MWCS-dominating set to v. Hence, the single-weight MWCS-dominating decision problem is in $\mathcal{NP}$.

In the following it will be demonstrated that the, $\mathcal{NP}$-complete (Garey and Johnson, 1979), vertex cover problem can be reduced to the single-weight MWCS-domination problem. Let $G_{cov} = (V_{cov}, E_{cov})$ be an undirected, non-empty graph and $k \in \mathbb{N}$. Thereupon, for the vertex cover problem it has to be determined whether a set in V_{cov} of cardinality at most k exists that is incident to all edges E_{cov}.

To establish the reduction, construct a graph G_0 from G_{cov} as follows: Start with $G_0 = (V_0, E_0) := G_{cov}$ and extend this graph as follows: First, define vertex weights $p_0(v) := -1$ for all $v \in V_0$. In the next step replace each edge $e_l = \{v, w\} \in E_0$ by a vertex v'_l of weight $p_0(v'_l) := -(k+1)$ and the two edges $\{v, v'_l\}$ and $\{w, v'_l\}$. Moreover, add edges $\{v, w\}$ for each pair of distinct vertices $v, w \in V_0 \cap V_{cov}$ to E_0. Due to the previous step, this procedure does not lead to multi-edges. Finally, add a vertex $v_0^\star$ of weight $p_0(v_0^\star) := -k$ to V_0 and add edges $\{v_0^\star, v\}$ for all $v \in V_0 \setminus (V_{cov} \cup \{v_0^\star\})$.

Scrutinizing the graphs G_0 and G_{cov}, one can verify that a single-weight MWCS-dominating set X to $v_0^\star$ exists if and only if to each (newly added) vertex $v \in V_0 \setminus (V_{cov} \cup \{v_0^\star\})$ there is an adjacent vertex $w \in V_0 \cap V_{cov}$ with $w \in X$. The latter condition is satisfied if and only if there is a vertex cover in G_{cov} of cardinality at most k. ☐

3.3.3 Combining dominating sets and constrained distances

Despite both being $\mathcal{NP}$-hard, the MWCS-domination and the constrained walk distance concept can be merged into a powerful additional reduction test. The stage for this combined routine is set by the following:

Proposition 3.30. *Let $U \subseteq V \setminus T_p$ and define*

$$\Delta := \big\{v \in V \setminus U \mid \exists \{v, w\} \in E, w \in U\big\}$$

If $\Delta = \emptyset$, then no optimal solution to I_{MW} contains any vertex of U. Otherwise, let $X \subseteq V \setminus U$ such that

$$\Delta_1 := \Delta \cap \Big(\big\{v \in V \setminus X \mid \exists \{v, w\} \in E, w \in X\big\} \cup X\Big)$$

is non-empty and $(X, E[X])$ is connected. Define

$$C_1 := \sum_{u \in X : p(u) < 0} p(u). \tag{3.82}$$

Further, let $\Delta_2 := \Delta \setminus \Delta_1$ and choose for each $v_k \in \Delta_2$ an, arbitrary, $v'_k \in X$. Define

$$C_2 := \sum_{v_k \in \Delta_2} d_{pw}(v_k, v'_k). \tag{3.83}$$

If

$$C := C_1 + C_2 > \sum_{u \in U} p(u), \tag{3.84}$$

then each optimal solution S to I_{MW} satisfies $U \not\subseteq V(S)$.

Proof. Let S be a feasible solution with $U \subsetneq V(S)$. Note that both $C_1 \leqslant 0$ and $C_2 \leqslant 0$. Define

$$\Delta_1^S := \Delta_1 \cap V(S)$$

and

$$\Delta_2^S := \Delta_2 \cap V(S).$$

In the following it will be demonstrated how to construct a connected subgraph S''' that does not contain all vertices of U and satisfies $P(S''') \geqslant P(S)$.

Let S' be the subgraph obtained from S by removing U and all incident edges. Note that each maximal connected component of S' contains at least one vertex of $\Delta_1^S \cup \Delta_2^S$. Furthermore, it holds that

$$P(S') = P(S) - \sum_{u \in U} p(u). \tag{3.85}$$

If $\Delta_1^S \neq \emptyset$, let S'' be the vertex-induced subgraph of $X \cup V(S')$. Otherwise set $S'' := S'$. In both cases, it holds for S'' that

$$P(S'') \geqslant P(S') + C_1 \overset{(3.85)}{=} P(S) - \sum_{u \in U} p(u) + C_1. \tag{3.86}$$

Moreover, all vertices of Δ_1^S are part of one connected component of S''.

Set $S''' := S''$. Consider each $v_k \in \Delta_2^S \setminus V(S''')$ consecutively and choose a (v_k, v'_k)-walk W^k (with v'_k as defined in the statement of this proposition) such that $l_{pw}(W^k) = d_{pw}(v_k, v'_k)$. If v_k and v'_k are in different connected components of S''', there exist $v_q \in V(W^k)$ in the connected components of v_k and $v'_q \in V(W^k)$ in the connected component of v'_k such that $V(W^k(v_q, v'_q)) \cap V(S''') = \{v_q, v'_q\}$. Add (the subgraph corresponding to) $W^k(v_q, v'_q)$ to S'''. Because of condition (3.84) there is at least on vertex of U that is not contained in any of these newly added paths—otherwise it would hold that $C_2 \leqslant \sum_{u \in U} p(u)$ and therefore also $C \leqslant \sum_{u \in U} p(u)$. Moreover, because of condition (3.83) the overall procedure reduces the weight of S'' by at most $|C_2|$. Hence, it holds for the new (now connected) subgraph S''' that

$$P(S''') \geqslant P(S'') + C_2 \overset{(3.86)}{\geqslant} P(S) - \sum_{u \in U} p(u) + C_1 + C_2. \tag{3.87}$$

Finally, $U \not\subseteq V(S''')$ holds and due to (3.84) it follows from (3.87) that

$$P(S''') > P(S). \tag{3.88}$$

Hence the proposition is proven. $\square$

Corollary 3.31. *Assume that the conditions of Proposition 3.30 hold, but instead of* (3.84) *assume*

$$C_1 + C_2 > \max_{u \in U} p(u). \tag{3.89}$$

Then each optimal solution S to I_{MW} satisfies $U \cap V(S) = \emptyset$.

Proof. Let S be a feasible solution. Further, let S''' be a connected subgraph created from S by the procedure described in the proof of Proposition 3.30. S''' is connected and it holds that $P(S''') > P(S)$, so only the equation $U \cap V(S''') = \emptyset$ needs to be verified. By construction all vertices of S''' are in one of the three sets: $(V(S) \setminus U)$, X, and the set of vertices that are part of a (v_k, v'_k)-walk W^k with $l_{pw}(W^k) = d_{pw}(v_k, v'_k)$ and $v_k \in \Delta_2^S$. By definition the first two of these sets cannot contain any vertices of U. Furthermore, because of (3.89), none of the walks W^k can contain a vertex of U since otherwise it would hold that $l_{pw}(W^k) \leqslant \max_{u \in U} p(u)$—which is a contradiction because of $C_1 + C_2 \leqslant C_2 \leqslant l_{pw}(W^k)$. Thus, $U \cap V(S) = \emptyset$. $\square$

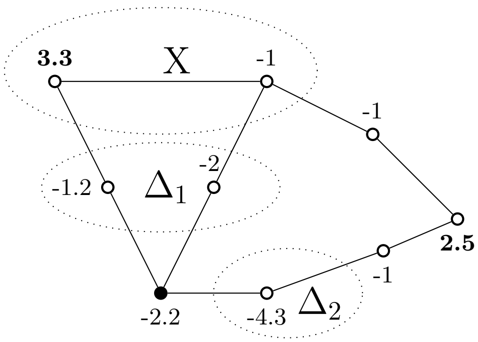


Figure 3.5: An MWCSP instance. Consider the vertices enclosed by the upper dotted ellipse as the set X, the lower left one as Δ_1, the lower right ones as Δ_2, and let U be the set that only contains the bottom left (filled) vertex. One can verify with Proposition 3.30 that the bottom left vertex can be deleted.

Once again, for the special case of $|U| = 1$, corollary and proposition coincide. Figure 3.5 shows an MWCSP for which a vertex can be deleted by means of this special case. Consider the upper two encircled vertices as the set X. The right neighbor (forming the set Δ_2) of the filled vertex can be connected by a walk of positive-weight constrained length -1 to X, so $C_2 \geqslant -1$. Since $C_1 = -1$ and all other neighbors (Δ_1) of the filled vertex are also neighbors of X, one can delete the vertex.

3.4 From dual-ascent to exact solving

An important (bound-based) SPG reduction technique can be derived from a dual-ascent algorithm by Wong (1984) for Formulation 1.1 (*DCut*), see e.g. Duin (1993). In the following, we shortly describe this dual-ascent algorithm, before discussing its applicability for MWCSP.

Consider an SAP (V, A, T, r, c). Let $\mathcal{W} := \{W \subset V \mid r \notin W, T \cap W \neq \emptyset\}$. Further, consider the dual of *DCut*:

$$\max \sum_{W \in \mathcal{W}} \mu_W \tag{3.90}$$

$$\text{s.t.} \sum_{W \in \mathcal{W} | a \in \delta^-(W)} \mu_W \leqslant c(a) \qquad \text{for all } a \in A, \tag{3.91}$$

$$\mu \geqslant 0. \tag{3.92}$$

Given a dual solution μ, let $A_\mu \subseteq A$ be the set of arcs for which (3.91) is tight. For each $t \in T \setminus \{r\}$, define the *root component* U_t of t as the set of vertices $v \in V$ such that there exists a directed v-t path in A_μ. A root component U_t is *active* if $T \cap U_t = \{t\}$. Initially, the dual-ascent algorithm sets $\mu := 0$. In each iteration an active root component U_t is chosen and μ is increased until (3.91) becomes tight for at least one $a \in \delta^-(U_t)$. This increase can be done implicitly by just adapting the reduced costs. The algorithm terminates when no active root component is left. The algorithm runs in $O(|A| \min\{|V||T|, |A|\})$, but is usually much faster in practice.

At termination, dual-ascent provides a dual solution to the LP-relaxation of Formulation 1.1, involving directed paths along arcs of reduced cost 0 from the root to each additional terminal. This information can be used to facilitate the solving process for an MWCSP, as will be show in the following. To apply the dual-ascent algorithm to MWCSP, we first devise a transformation from MWCSP to SAP. We note that Leitner et al. (2018a), independently from our work, developed an extension of the dual-ascent algorithm that can also be applied to MWCSP (as well as to the prize-collecting Steiner tree problem). However, their algorithm essentially requires a root, and thus needs to run several times to obtain valid bounds. In contrast, our approach requires just a single execution of dual-ascent.

The underlying idea of the transformation is to treat the MWCSP vertices of positive weight as terminals in the SAP. However, since all terminals need to be part of any feasible SAP solution, several vertices (including a root) and arcs are added to the SAP that allow us to model the exclusion of positive vertex from a feasible solution. Recall that we assume I_{MW} to contain at least one vertex of positive, and one of negative weight.

Transformation 3.32 (MWCSP to SAP).
  Input: *An MWCSP* $I_{MW} = (V, E, p)$
  Output: *An SAP* $P' = (V', A', T', c', r')$

1. *Set* $V' := V$*,* $A' := \{(v, w) \in V' \times V' \mid \{v, w\} \in E\}$*.*

2. *Set* $c' : A' \to \mathbb{Q}_{\geqslant 0}$ *such that for* $a = (v, w) \in A'$*:*
$c'(a) = \begin{cases} -p(w), & \text{if } p(w) < 0 \\ 0, & \text{otherwise} \end{cases}$
3. *Add two vertices* r' *and* v'_0 *to* V'*.*
4. *Denote the set of all* $v \in V$ *with* $p(v) > 0$ *by* $T_p = \{t_1, ..., t_s\}$ *and define* $M := \sum_{t \in T_p} p(t)$*.*
5. *For each* $i \in \{1, ..., s\}$*:*
   (a) *Add an arc* (r', t_i) *of weight* M *to* A'*.*
   (b) *Add a new node* t'_i *to* V'*.*
   (c) *Add arcs* (t_i, v'_0) *and* (t_i, t'_i) *to* A'*, both being of weight* 0*.*
   (d) *Add an arc* (v'_0, t'_i) *of weight* $p(t_i)$ *to* A'*.*
6. *Define the set of terminals* $T' := \{t'_1, ..., t'_s\} \cup \{r'\}$*.*
7. ***Return*** (V', A', T', c', r')*.*

The following proposition establishes the relation between the SAP resulting from the above transformation, and the original MWCSP.

Proposition 3.33 (MWCSP to SAP)**.** *Let* $I_{MW} = (V, E, p)$ *be an MWCSP and* $I' = (V', A', T', c', r')$ *an SAP obtained from* I_{MW} *by Transformation 3.32. Let* $S' \subseteq (V', A')$ *be an optimal solution to* I'*. The set* $S \subseteq (V, E)$ *defined by*

$$V(S) := \{v \in V \mid v \in V'(S')\}, \tag{3.93}$$
$$E(S) := \{\{v, w\} \in E \mid (v, w) \in A'(S') \text{ or } (w, v) \in A'(S')\}. \tag{3.94}$$

is an optimal to I_{MW}*. Further, it holds that:*

$$P(S) = \sum_{v \in V : p(v) > 0} p(v) - \sum_{a \in A'(S')} c'(a) + M. \tag{3.95}$$

Proof. Let S' be an optimal solution to I'. One readily verifies that the S defined by (3.93) and (3.94) is connected, and thus feasible for I_{MW}. We first show that (3.95) holds for S and S'. Further, one verifies that $\delta^+_{S'}(r') = 1$. Define $A := \{(v, w) \in A' \mid \{v, w\} \in E\}$. First, one observes that for each $v \in V(S)$ such that $p(v) \leqslant 0$ there is exactly one incoming arc $a \in A(S')$, so:

$$\sum_{v \in V(S) : p(v) \leqslant 0} p(v) = -\sum_{a \in A(S')} c'(a). \tag{3.96}$$

Second:

$$\sum_{v \in V(S) : p(v) > 0} p(v) = \sum_{v \in V : p(v) > 0} p(v) - \sum_{v \in V \setminus V(S) : p(v) > 0} p(v) \tag{3.97}$$
$$= \sum_{v \in V : p(v) > 0} p(v) - \sum_{a \in A'(S') \setminus A(S')} c'(a) + M. \tag{3.98}$$

Finally, by combining (3.96) and (3.97) the equation:

$$\sum_{v \in V(S)} p(v) = \sum_{v \in V: p(v) > 0} p(v) - \sum_{a \in A'(S')} c'(a) + M \tag{3.99}$$

is obtained, which coincides with (3.95).

Finally, suppose that S is not optimal. I.e., there is a solution $\tilde{S}$ to I_{MW} such that $P(\tilde{S}) > P(S)$. Since we have presupposed that at least one vertex of V has positive weight, we can assume that there is an $u \in V(\tilde{S}) \cap T_p$. We build a solution $\tilde{S}'$ to the SAP I' as follows: First, we define $V'(\tilde{S}') := \{r', u, v_0'\}$, $A'(\tilde{S}') := \{(r', u), (u, v_0')\}$, and add to $A'(\tilde{S}')$ all arcs reachable from u through forward arcs (v, w) such that $\{v, w\} \in E(\tilde{S})$. Concomitantly, we add all vertices corresponding to arcs in $A'(\tilde{S}')$ to $V'(\tilde{S}')$. Second, we add for each $t_i \in T_p$ contained in $V'(\tilde{S}')$ the arc (t_i, t_i'), which is of cost 0, to $A'(\tilde{S}')$. For all $t_i \in T_p$ not connected we add the arc (v_0', t_i'), which is of cost $p(t_i)$, to $A'(\tilde{S}')$. Finally, we add all $t_i' \in T'$ to $V'(\tilde{S}')$. Consequently, all $t_i' \in T'$ are reachable from r' through forwards arcs and, being cycle-free and connected, $\tilde{S}'$ is a solution to P'. Furthermore, because S and S' satisfy (3.95), it holds that:

$$\sum_{a \in A'(\tilde{S}')} c'(a) - M = \sum_{v \in V: p(v) > 0} p(v) - P(\tilde{S}) \tag{3.100}$$

$$< \sum_{v \in V: p(v) > 0} p(v) - P(S) \tag{3.101}$$

$$= \sum_{a \in A'(S')} c'(a) - M, \tag{3.102}$$

which contradicts the assumption that S' is an optimal solution to I'. Therefore, S is an optimal solution to I_{MW}. □

Note that the proposition is just concerned with the correspondence of optimal solutions. See Rehfeldt and Koch (2018a) for a (slightly more involved) map of each feasible solution to I' to a feasible solution to I_{MW}. The additional technicalities of the latter result are not relevant in the following, and therefore no further details are given.

Similar to Section 3.2.4, one can show that applying the *DCut* formulation on the SAP I' from the above transformation yields (after a constant shift of the objective) the same optimal LP value as ESA. Furthermore, one notes that the constraints (1.3) for I' corresponding to all non-zero μ_W can be readily transformed to constraints (3.30) for ESA^+. These can be used as initial constraints for a branch-and-cut algorithm.

In practice, one tries to only increase small root components in the dual-ascent algorithm. Moreover, it is advantageous to only update the currently used root component and rebuild U_t by a BFS or DFS after each change of t, see Pajor et al. (2017). For I' one notices that for distinct terminals t_i, t_j with $v_0' \in U_{t_i}$ and $v_0' \in U_{t_j}$ it holds that $U_{t_i} \setminus \{t_i\} = U_{t_j} \setminus \{t_j\}$. However, due to the structure of I' all root components will remain active until the end of the algorithm. Thus, a simple, but sometimes highly effective modification of I' is to make v_0' a terminal. In this way, any root component U_t ceases to be active as soon as $v_0' \in U_t$. Still, the final reduced costs remain the same.

As already noted, an important application of dual-ascent is within reduction techniques. Consider an SAP (V, A, T, c, r). Let $v \in V \setminus T$, let $S^\star$ be an optimal solution, and let L_{DA} be the lower bound obtained by dual-ascent. If $S^\star$ contains v, the weight of $S^\star$ can be bounded from below by L_{DA} plus the length (with respect to the reduced costs provided by dual-ascent) of a shortest path from the root to v and the length of a shortest path from v to a nearest terminal (other than the root) to v. Hence, v can be deleted if the just defined bound exceeds a known upper bound U. An analogous test can be stated for the elimination of arcs. The above deliberations forthwith set the stage for an MWCSP reduction technique: Whenever a vertex can be deleted in the SAP, the same is true for its counterpart in the analogous MWCSP. Whenever two anti-parallel arcs in the SAP can be deleted, the corresponding edge can be deleted in the MWCSP.

Transformation 3.32 can additionally be used to show that a vertex $t_i \in T_p$ is part of at least one optimal solution. If the reduced cost of an arc (r', t_i') is higher than $U - L_{DA}$, it can be deduced that the vertex t_i is part of at least one optimal solution to I_{MW}. If at least one (positive) vertex can be shown to be part an optimal solution, the MWCSP can be solved as an RMWCSP. Note that a disadvantage of both ESA^+ and the above IP resulting from Transformation 3.32 is the existence of symmetric solutions (for a solution S, there are $|V(S) \cap T_p| - 1$ many). For the RMWCSP one can instead apply the following (new) transformation, which gives way to a problem that is not burdened with symmetric solutions. The transformation will be provided in a more general setting, namely for the directed variant of the RMWCSP (described in Section 3.2.1). As before, it will be assumed that all fixed terminals T_f have 0-weight.

Transformation 3.34 (Directed RMWCSP to SAP)**.**
Input: *A directed RMWCSP* $I_{RMW} = (V, A, T_f, p, r)$
Output: *An SAP* $I' = (V', A', T', c', r')$

1. *Set* $V' := V$*,* $A' := A$*,* $r' := r$*.*
2. *Set* $c' : A' \to \mathbb{Q}_{\geqslant 0}$ *such that for* $a = (v, w) \in A'$*:*
$$c'(a) = \begin{cases} -p(w), & \text{if } p(w) < 0 \\ 0, & \text{otherwise} \end{cases}$$
3. *Denote the set of all* $v \in V \setminus T_f$ *with* $p(v) > 0$ *by* $T_p = \{t_1, ..., t_s\}$
4. *For each* $i \in \{1, ..., s\}$*:*
   (a) *Add a new node* t_i' *to* V'*.*
   (b) *Add an arc* (r', t_i') *of weight* $p(t_i)$ *to* A'*.*
   (c) *Add an arc* (t_i, t_i') *of weight* 0 *to* A'*.*
5. *Define the set of terminals* $T' := \{t_1', ..., t_s'\} \cup \{T_f\}$*.*

6. ***Return*** (V', A', T', c', r').

The transformation is illustrated in Figure 3.6. Moreover, the correspondence between a directed RMWCSP and the SAP resulting from Transformation 3.34 is established by the following proposition.

(a) Directed RMWCSP instance (b) Transformed SAP instance

Figure 3.6: Illustration of a directed RMWCSP instance with root r (left) and the equivalent SAP obtained by Transformation 3.34 (right). Terminals are drawn as squares.

Proposition 3.35 (Directed RMWCSP to SAP). *Let $I' = (V', A', T', c', r')$ be an SAP obtained from a directed RMWCSP $I_{RMW} = (V, A, T_f, p, r)$ by applying Transformation 3.34. Each solution S' to I' can be mapped to a solution S to I_{RMW} defined by:*

$$V(S) := V \cap V'(S'), \tag{3.103}$$

$$A(S) := A \cap A'(S'). \tag{3.104}$$

If S' is an optimal solution to I', then S is an optimal solution to I_{RMW} and their objective values satisfy:

$$P(S) = \sum_{v \in V : p(v) > 0} p(v) - \sum_{a \in A'(S')} c'(a). \tag{3.105}$$

The proposition can be proved in a similar way as Proposition 3.33. In the following, the *DCut* formulation for the SAP (V', A', T', c', r') obtained from an directed RMWCSP by performing Transformation 3.34 will be referred to as $TransCut$. Note that $TransCut$ can also be used for an undirected RMWCSP by using as simple transformation to a bidirected graph. The objective value of a solution $y \in \{0,1\}^{|A'|}$ to $TransCut$ is defined as:

$$v(TransCut) := \sum_{v \in V : p(v) > 0} p(v) - {c'}^T y. \tag{3.106}$$

One can strengthen the formulation by adding the flow-balance constraints (2.9). The SAP resulting from Transformation 3.34 displays two immediate advantages as compared to the one from Transformation 3.32. First, the number of arcs is reduced by

$2|T_p|$. Second, while for each (LP) solution to the *DCut* formulation of the SAP from Transformation 3.32 there can be up to $|T_p| - 1$ equivalent solutions, this symmetry has vanished in the *TransCut* formulation. In addition to these advantages, the new SAP can be solved by the separation algorithm of SCIP-Jack without any alterations.

In Rehfeldt and Koch (2019) we additionally show that the *TransCut* formulation has an LP-relaxation that is strictly stronger than that of Formulation 3.2 (*RNCut*), which uses only node variables. I.e., we have $v_{LP}(TransCut) \leqslant v_{LP}(RNCut)$, and the inequality can be strict. The proof in Rehfeldt and Koch (2019) is rather long and tedious, and thus omitted here. We note, however, that a shorter proof can be obtained by adapting the results from Section 3.2.3.

One might argue that despite its inferior LP-relaxation the *RNCut* formulation is preferable in practice since it leads to a problem with far fewer variables: *RNCut* only considers nodes, and *TransCut* moreover requires additional arcs for each $t \in T_p$. However, preprocessed MWCSP instances are in practice sparse and include only a small amount of positive-weight vertices (Rehfeldt et al., 2019). In particular, in a preprocessed MWCSP instance there are no adjacent vertices of positive weight.

3.5 Primal heuristics

Having discussed the computation of dual bounds for the MWCSP in the previous section, we now turn to the primal side. Several primal heuristics for the MWCSP have been described in the literature. In Álvarez-Miranda et al. (2013a), for example, a breadth-first-search-based heuristic that makes use of the reduced cost obtained during a branch-and-cut algorithm was suggested. In Leitner et al. (2018a) several variants of a dual-ascent-based heuristic for the rooted prize-collecting Steiner tree problem were suggested, and were also applied for the MWCSP by using a transformation initially introduced in Dittrich et al. (2008). For other MWCSP heuristics see Álvarez Miranda and Sinnl (2017); Fu and Hao (2017b); El-Kebir and Klau (2014).

In the following, we introduce several new MWCSP primal heuristics.

3.5.1 Constructive heuristics

As the name suggests, constructive heuristics build up a new solution from scratch.

A greedy approach

The first heuristic is similar to the classic shortest paths heuristic for the SPG, see Section 2.5, and is conceptually straightforward: Starting with a single vertex, the heuristic iteratively connects the current subtree to vertices of positive weight (as compared to terminals for SPG). In the case of the SPG a natural choice for the connection of a terminal is a shortest path, but in the case of MWCSP the choice is less clear. The following algorithm chooses paths that also take intermediary vertices of positive weight into account.

Let vertex $v_r \in V$ be the start vertex. Initially, define for all $v \in V \setminus \{v_r\}$: $p^+(v) := \max\{p(v), 0\}$, $p^-(v) := \max\{-p(v), 0\}$, $\tilde{d}(v) := \infty$. For v_r set all these values to 0. Define a predecessor $pred(v) := null$ for each $v \in V$. Define the initial tree S as $V(S) := \{v_r\}, E(S) := \emptyset$, and set $Q := \{v\}$. While $Q \neq \emptyset$ proceed as follows. Choose a $v = \arg\min_{u \in Q} \tilde{d}(u)$ and remove v from Q. If $p(v) > \tilde{d}(v)$, add the path P between S and v marked by $pred$ to S. Further, set for each $w \in V(P)$: $\tilde{d}(w) := p^+(w) := 0$, and add w to Q.

In any case, define for each $\{v, w\} \in \delta(v)$

$$\tilde{d}_{vw} := \tilde{d}(v) + p^-(w) - \min\{p^-(w), p^+(v)\}. \tag{3.107}$$

If $\tilde{d}_{vw} < \tilde{d}(w)$, add w to Q and set $\tilde{d}(w) := \tilde{d}_{vw}$, $pred(w) := v$. Note that in (3.107) we cannot just subtract $p^+(v)$, because otherwise the algorithm might cycle. Once Q is empty, we compute a spanning tree on the graph induced by $V(S)$, while trying to have vertices $V(S) \setminus T_p$ as leaves. Then, we use a linear-time dynamic programming algorithm described in Magnanti and Wolsey (1995) to compute a maximum-weight connected subgraph on this tree.

If an LP solution to the MWCSP is available, modified vertex weight p' can be used for the heuristic. For instance, assume an LP solution (x, y) to ESA is given. Set $p'(v) := x(v)p(v)$ for all $v \in V$. Moreover, the heuristic is started from a (constant) number of distinct vertices. To this end, vertices $v \in V$ with highest value $x(v)$ in the incumbent LP solution are chosen as starting points. In case of ties, vertices with higher weight p are preferred.

In the following, the above algorithm is referred to as *Greedy Construction* (GC) heuristic. It should be noted that the idea of reinserting vertices into the priority queue of Dijkstra's algorithm was already used in a heuristic for the SPG (de Aragão and Werneck, 2002). Furthermore, the concept of using LP solutions to guide primal heuristics for combinatorial optimization problems is widely used, see for example Koch and Martin (1998).

A reduction-based approach

The first of two reduction-based approaches described in this chapter builds on a concept introduced as *prune* in Polzin and Daneshmand (2001b) for SPG. By virtue of the PVD method introduced in Section 3.3, this approach can now be used for the MWCSP as well. While for the original PVD test an upper bound is provided by the weight of a given solution, in the prune heuristic the bound is chosen such that in each iteration a certain number of vertices is eliminated. Thereupon, all exact reductions methods are executed on the reduced graph, motivated by the assumption that the (possibly inexact) eliminations performed by the bound-based method will allow for further (exact) reductions. To avoid infeasibility, initially a feasible solution is computed (by using GC and the subsequently described local-search heuristics) of which no vertices or edges are allowed to be deleted by the (inexact) bound-based method.

The second reduction-based heuristic approach is borne from the combination of the prune heuristic and dual-ascent: the *ascend-reduce* method (based on an approach originally suggested in Wong (1984) for the Steiner tree problem in graphs). Let I_{MW} be the original MWCSP and I' the SAP resulting from Transformation 3.32. The ascend-reduce heuristic attempts to find a good solution on the subproblem $\tilde{I}_{MW}$ constituted by the undirected edges of I_{MW} corresponding to zero-reduced-cost paths in I' from the root to all additional terminals—in this chapter a solution is computed by employing reduction techniques and heuristics. This approach is motivated by the assumption that notable similarities exist between an optimal (or near-optimal) MWCSP solution and the LP solution corresponding to the reduced costs provided by dual-ascent.

3.5.2 Local search heuristics

Given a solution S to a problem, a local search algorithm examines a *neighborhood* of S, i.e. a set of solutions obtainable from S by performing a predefined set of operations. Consequently, all heuristics described in this section assume that a feasible solution S is given.

Greedy extension

The *Greedy Extension* (GE) heuristic works in two phases: In phase one all vertices $V(S)$ are inserted into the priority queue of Dijkstra's algorithm with their distance values set to 0. Thereupon, the GC heuristic is executed. However, the GC algorithm is only stopped when all vertices of positive weight have been scanned (the criterion for including vertices to the current solution remains unchanged). Furthermore, the heuristic saves (a constant number) $\alpha \in N$ (or as many as exist) vertices $t'_k \in T_p \setminus V(S), k = 1, ..., \alpha$ such that $p(t'_k) - \tilde{d}_{vw}(t'_k) \geqslant p(t) - \tilde{d}_{vw}(t)$ for all other vertices $t \in T_p \setminus V(S)$. In phase two, α iterations $k = 1, ..., \alpha$ are performed; in each iteration vertex t'_k is connected to S (by using the paths computed by the GC heuristic) to obtain a solution S_k. Next, the GC heuristic is executed from S_k and updates S in case a better solution could be found. Analogously to the GC heuristic, GE can be executed with altered vertex weights if an LP solution is available.

Vertex inclusion and vertex exclusion

The idea of the *Vertex Inclusion* (VI) heuristic is to add a vertex to a given solution such that other negative-weight vertices of the solution can be discarded. First, compute a spanning tree S_{span} on S. Next, iterate through all neighboring vertices v_i of S: Let $\delta_{S,i}$ be the set of all edges between v_i and $V(S)$. If $|\delta_{S,i}| \leqslant 1$, continue. Otherwise, add an (arbitrary) edge $a'_0 \in \delta_{S,i}$ to S_{span}. Afterwards, iteratively add each edge $a'_j \in \delta_{S,i} \setminus \{a'_0\}$ to S_{span}. Whenever a new edge has been added, search for a minimum-weight sequence of vertices of degree 2 (with respect to S_{span}) on the newly created cycle. If such a sequence being of negative weight exists (including single vertices), remove it from S_{span}, otherwise remove a'_j. When all edges $\delta_{S,i}$ have been checked and if the weight of the removed vertices is smaller than that of v_i, leave S_{span} in its modified form, otherwise restore it. In the implementation of the heuristic (linear) *link-cut trees* (Sleator and Tarjan, 1983) are used. This data structure allows to easily dis- and reconnect trees. A similar heuristic is known for the SPG (Uchoa and Werneck, 2010), but it only eliminates single edges instead of chains.

A complementary approach is taken by the *Vertex Exclusion* (VE) heuristic: it aims to remove vertices of S. Consider the connected subgraph

$$G_S := (V(S), E[V(S)]).$$

Thereupon, the heuristic employs reduction techniques from Section 3.3 on G_S to obtain a new graph $\tilde{G_S}$. On this new graph the GE heuristic is used to obtain a solution $\tilde{S}$. Finally, this solution is retransformed to a solution S' to the original problem.

3.6 Solving to optimality

With several algorithmic components introduced and discussed two central questions remain: How to assemble these threads and weave them into a coherent exact solver, and, equally important, how does the resulting algorithm perform?

3.6.1 A full-fledged exact solver

The exact solver described in this chapter is realized within our Steiner tree problem solver SCIP-Jack. All MWCSP algorithms described so far are integrated into a branch-and-cut algorithm, as detailed in the following. The IP formulation used by the solver is (essentially) ESA^+_{FB} for MWCSP, and $TransCut_{FB}$ for RMWCSP. For both formulations we use separation. To this end, we use a specialized maximum-flow algorithm with warm-start capabilities (see Section 6.2.4) to detect violated inequalities.

Reduction techniques can be found in three components of our branch-and-cut algorithm: In preprocessing, in domain propagation, and within primal heuristics. Preprocessing is performed in several rounds—as long as a predefined percentage of edges has been eliminated during the previous round. During preprocessing we also try to transform any MWCSP instance to RMWCSP. For domain propagation during branch-and-bound we proceed as follows: Instead of deleting edges or vertices, the corresponding variables are fixed to 0 in the IP formulation. Additionally, instead of the reduced-costs from dual-ascent, we use the reduced-costs provided by the LP-relaxation for further reductions.

Another component instrumental for empirically successful exact solving is constituted by the primal heuristics described in Section 3.5. In the implementation for this chapter, ascend-reduce uses the prune heuristic to find a solution on the graph obtained by dual-ascent and employs GE, VI and VE to improve it. In turn, the prune heuristic calls GC to obtain an initial feasible solution, calls the local search heuristics GE, VI to improve it, and employs several reduction methods. Finally, to improve the solution obtained by ascend-reduce, all local search heuristics are used. This heuristic package is repeatedly used during preprocessing. Furthermore, the heuristics GC, GE, and VI are used during branch-and-bound.

Finally, branching is performed on vertices—by assigning the vertex v_i to branch on weight $p(v_i) = \infty$ in one branch-and-bound child node and removing it in the other—which seems to be the natural choice for the MWCSP.

A look beyond the surface of the new exact MWCSP solver reveals an intricate synergy of the three major solving components introduced in this chapter: First, heuristics and reduction techniques are deeply intertwined. Reduction methods are crucial for the success of both prune and ascend-reduce, while the quality of the primal bound obtained by these heuristics determines the effectiveness of the dual-ascent reduction method. Indeed, the prune heuristic could only be realized due to the newly introduced PVD concept. Furthermore, only the combination of dual-bound and reduced costs obtained by dual-ascent, and the primal bound provided by ascend-reduce consistently gives rise to the transformation of MWCSP to RMWCSP and the subsequent application of the $TransCut$ formulation. In turn, on the SAP obtained from this transformation one can again execute the dual-ascent reduction method.

3.6.2 Computational results

For details on the hardware used in this thesis see Section 1.2.1. The computational evaluation for this chapter has been performed on the six test-sets described in Table 3.1. The ACTMOD and JMPALMK instances were all solved to optimality in the course of the 11th DIMACS Challenge DIMACS (2015), whereas the SHINY test-set Loboda et al. (2016) was introduced later. The two test-sets with the largest instances, HANDBI and HANDBD, are from Hegde et al. (2014), and were originally formulated as prize-collecting Steiner tree problems. However, the instances have uniform edge weights and can therefore be transformed to MWCSP. Most of these large instances proved to be intractable for the solvers participating in the 11th DIMACS Challenge, and could only recently be solved to optimality by Leitner et al. (2018a). Still, three instances of these two test-sets have remained unsolved. Finally, the PUC test-set contains instances that were designed to defy known solution techniques. Five of the 18 PUC instances have remained unsolved.

Name	Instances	$\|V\|$	$\|E\|$	Status	Description
JMPALMK	72	500-1500	2597-20527	solved	Euclidean, randomly generated instances from Álvarez-Miranda et al. (2013a).
SHINY	39	232-3828	202-4494	solved	Instances from network enrichment analysis in computational biology (Loboda et al., 2016).
ACTMOD	8	2034-5226	3335-93394	solved	Instances from integrative biological network analysis (Dittrich et al., 2008).
HANDS	20	39600 - 42500	78704 - 84475	solved	Images of hand-written text from a signal processing problem (Hegde et al., 2014).
HANDB	28	158400 - 169800	315808 - 338551	unsolved	
PUCNU	18	64-4096	192-24574	unsolved	Adversarial instances from the 11th DIMACS Challenge (DIMACS, 2015).

Table 3.1: Classes of MWCSP instances.

Reduction results

The impact of the individual of the algorithmic components is difficult to measure due to their strong interaction. For example, deactivating the primal heuristics also has a large effect on the reduction methods, since heuristics are heavily used for the bound-based reductions. Vice-versa, reductions techniques are also a central ingredient of several primal heuristics. For computational results on the impact of the individual components we refer to Rehfeldt and Koch (2019), but note that the knowledge gain from these results is limited. Essentially, with any of the three main building blocks, reduction techniques, primal heuristics, graph transformations and IP formulation being deactivated, the solving behavior drastically deteriorates.

Here, we merely provide results on the strength of the reduction methods when used for presolving. Note that these methods also make use of the primal and dual heuristics, as well as of the new graph transformations (for applying dual-ascent). Table 3.2 shows the arithmetic mean of the percentage of vertices and edges in the presolved problems. Further, we report the shifted geometric mean (see Section 1.2.2) of the run-time needed per test-set, with shift $s = 1$. It can be seen that the considerable effort put into the various algorithms used within presolving pays off. Apart from PUCNU, the average size of both the number of vertices and edges is reduced by more than 95 percent on all test-sets. Most instances are even solved to optimality. Given that some of the instances contain more than 300 thousand edges, the run-times are also tolerable: Even for the, large-scale, HANDB instances, the shifted geometric mean of the run-times is less than three seconds. We also note that the instances in PUCNU were designed to defy known solution techniques such as reduction methods. Against this backdrop, the obtained reductions are still notable.

	average reduced problem size		
Test-set	vertices[%]	edges[%]	mean reduction time [s]
SHINY	0.2	0.2	0.0
JMPALMK	0.1	0.0	0.0
ACTMOD	0.0	0.0	0.1
HANDS	0.4	0.2	0.3
HANDB	3.5	3.4	2.6
PUCNU	73.1	64.5	0.4

Table 3.2: Average problem sizes after application of reduction algorithms.

Exact solution and comparison

Table 3.3 provides aggregated results on the exact solution of the benchmark sets from Table 3.1. The first column shows the test-set considered in the current row. Columns two shows the shifted geometric mean of the run-time of SCIP-Jack. Since most of the instances can be handled very quickly, we have chosen a shift of $s = 1$. The next column provides the maximum run-time per test-set, and the last column the number of solved instances.

The results for the first three test-sets—JMPALMK, SHINY, and ACTMOD—reveal a strong performance of SCIP-Jack. All instances are solved in less than 0.3 seconds, and all but three instances are even solved within 0.1 seconds. To the best of the author's knowledge, SCIP-Jack significantly outperforms all other solvers from the literature on these instances. For example, most instances of SHINY are solved two orders of magnitude faster than the solver described in Loboda et al. (2016) (results for the other test-sets are not given). Furthermore, the maximum run-time in Loboda et al. (2016) on the SHINY test-set is more than a thousand seconds (with a few instances remaining unsolved), while it is less than 0.1 seconds for SCIP-Jack. The computational environment for the experiments in Loboda et al. (2016) is described as AMD Opteron 6380 CPUs with 2.5 GHz. Fischetti et al. (2017), whose solver won the MWCSP category at the 11th DIMACS Challenge, reports results for JMPALMK and ACTOMOD, using a machine that is roughly 1.4 times slower than ours, according to the DIMACS benchmark score. Taking the different machine into account, Fischetti et al. (2017) is between one and two orders of magnitude slower on almost all JMPALMK and ACTOMOD instances. The difference is even larger for other MWCSP solvers competing at the DIMACS Challenge, such as Althaus and Blumenstock (2014); El-Kebir and Klau (2014).

The best other results for the ACTMOD instances are achieved by the solver from Leitner et al. (2018a). Still, the solver is about an order of magnitude slower than SCIP-Jack on these instances (with both using the same machine). Other MWCSP solvers introduced after the 11th DIMACS Challenge, such as Álvarez Miranda and Sinnl (2017), are outperformed by Leitner et al. (2018a).

Also on the two HAND test-sets, SCIP-Jack is notably faster than other solvers from the literature. Already for the easier HANDS test-set, the solver from Fischetti et al. (2017) fails to solve several instances within the one hour time limit, see Leitner et al. (2018a). In contrast to at most two seconds taken by SCIP-Jack. Again, the best other results are achieved by Leitner et al. (2018a). The run-time of this solver is within two orders of magnitude of the run-time of SCIP-Jack on most HAND instances. Still, Leitner et al. (2018a) is always at least around one order of magnitude slower. Further, SCIP-Jack can solve one more instance to optimality (or even two if the time-limit is increased). This instance, *handbd04*, is also solved for the first time to optimality—in less than half a minute.

Test-set	# instances	# solved	mean time [s]	maximum time [s]
JMPALMK	72	72	0.0	0.0
SHINY	39	39	0.0	0.1
ACTMOD	8	8	0.1	0.3
HANDS	20	20	0.3	1.6
HANDB	28	26	4.5	>7200
PUCNU	18	13	55.6	>7200

Table 3.3: Computational results of the MWCSP solver described in this chapter.

Finally, the PUCNU test-set proves to be the hardest for SCIP-Jack, with only 13 out of 18 instances being solved within two hours. The solver from Leitner et al. (2018a) solves 7 instances within the same time. Fischetti et al. (2017) perform better, and solve 10 instances to optimality. Also from the PUCNU test-set, several instances are solved for the first time to optimality by SCIP-Jack—as detailed below.

Progress on unsolved instances

Name	gap [%]	new UB	previous UB
handbd04	**opt**	**3202.18574**	3202.710021
handbd13	**opt**	**13.184261**	13.187615
cc7-3nu	**opt**	**270**	271
cc10-2nu	**opt**	**167**	168
handbi13	0.1	4.24964	4.260670
cc11-2nu	0.8	303	304
cc12-2nu	0.7	563	565

Table 3.4: Improvements on unsolved DIMACS instances in a long run (24 h).

Table 3.4 shows results obtained with a time limit of 24 hours on previously unsolved instances from the 11th DIMACS Challenge. We ran the experiments with two different random seeds. The impact of using different random seeds was limited, but we were at least able to further improve one primal bound in this way. We provide the final optimality gap (column two), the best primal objective value (column three), and the known primal objective value from the literature (column four). Overall, four instances can be solved for the first time to optimality.

3.7 Conclusion

This chapter has set about to improve the state of the art in exact MWCSP solution. It started with analyses and comparisons of IP and MIP formulations with respect to the strength of their LP-relaxations. Along the way, we have also provided a tighter (compact) description of the connected subgraph polytope—the convex hull of subsets of vertices that induce a connected subgraph. Furthermore, we have given a (compact) complete description of the connected subgraph polytope for graphs with no four independent vertices. As an important conclusion of the theoretical study, we have seen that the considered edge-based formulations are consistently stronger than the node-based ones.

Based on the strongest of the considered (M)IP formulations, this chapter has continued with various new algorithms for exact MWCSP solution. Three central components are reduction techniques, graph transformations, and heuristics. The—surprisingly symbiotic—synergy of all three components gives rise to a powerful branch-and-cut algorithm that outperforms previous approaches by a large margin. Furthermore, several benchmark instances from the 11th DIMACS Challenge can be solved for the first time to optimality, and the best known primal solution of other ones can be improved.

Still, the end of the road is certainly not reached yet. For example, several powerful algorithms introduced in Chapter 2 for SPG, such as the extended reduction techniques or the FPT dynamic programming algorithm, could be extended to MWCSP. On the theoretical side, it seems well worthwhile to study new IP formulations for the MWCSP to further improve the strength of the LP-relaxation, and to obtain a tighter polyhedral description. For example, it might be possible to strengthen the node-based formulations by additional constraints to match their edge-based counterparts. Improvements of MWCSP complexity results, which were not covered in this chapter, will be introduced as part of the more general prize-collecting Steiner tree problem in the following.

Chapter 4

A generalization: The prize-collecting Steiner tree problem

This chapter is concerned with a well-known generalization of both problems considered in the previous two chapters: The *prize-collecting Steiner tree problem* (PCSTP). As in the previous chapter, we combine extensions of already introduced algorithms with conceptually new ones. However, since the PCSTP generalizes MWCSP and SPG, some new results for PCSTP can also be applied to these problems.

4.1 Introduction

The prize-collecting Steiner tree problem (PCSTP) can be stated as follows: Given an undirected graph $G = (V, E)$, edge weights $c : E \to \mathbb{Q}_{>0}$, and node weights (or *prizes*) $p : V \to \mathbb{Q}_{\geqslant 0}$, a tree $S = (V(S), E(S)) \subseteq G$ is required such that

$$C(S) := \sum_{e \in E(S)} c(e) + \sum_{v \in V \setminus V(S)} p(v) \tag{4.1}$$

is minimized. By setting sufficiently high node weights for its terminals, each SPG instance can be transformed to a PCSTP. Moreover, PCSTP can also be considered a generalization of MWCSP: As we will see in Section 4.2.2, any MWCSP instance is essentially a PCSTP with unit edge weights. PCSTP has occasionally also been formulated in maximization form, see e.g. Johnson et al. (2000), where the objective function

$$\sum_{v \in V(S)} p(v) - \sum_{e \in E(S)} c(e) \tag{4.2}$$

is to be maximized. While the minimization and maximization formulations are equivalent for exact solution, for approximation algorithms the two formulations are fundamentally different. PCSTP in its minimization form can be approximated within a factor of 2 in polynomial time, see e.g. Goemans and Williamson (1995), but it is $\mathcal{NP}$-hard to approximate the maximization form within any constant factor, see e.g. Feigenbaum et al. (2001).

The relevance of the PCSTP can at least partly be attributed to its large number of practical applications. These applications can be found in various areas, for instance in the design of telecommunication networks (Ljubic, 2004), electricity planning (Bolukbasi and Kocaman, 2018), geophysics (Schmidt et al., 2015), and even machine learning (Hidayati et al., 2019). Moreover, PCSTP is a popular tool in computational biology, see e.g. Akhmedov et al. (2018); Ideker et al. (2002); Tuncbag et al. (2016). Recently, PCSTP has also been employed for data mining, see e.g. Gionis et al. (2017).

The PCSTP is notably younger than the SPG (albeit still older than the author of this thesis): It was introduced around 15 years after the SPG by Segev (1987)[17]. However, since then, the PCSTP has been extensively discussed in the literature, both from theoretical and practical perspectives. The first approximation algorithm was introduced by Bienstock et al. (1993), and achieved a factor 3 approximation. This factor was later improved by Goemans and Williamson (1995), Johnson et al. (2000), and Feofiloff et al. (2007). The latter achieve a $(2 - \frac{2}{|V|})$-approximation. Finally, Archer et al. (2011) proposed a $(2 - \varepsilon)$ approximation; with $0.03 < \varepsilon < 0.04$. For approximation results on planar graphs see Bateni et al. (2011). Moreover, a large number of heuristic algorithms for PCSTP have been suggested, see e.g. Canuto et al. (2001); da Cunha et al. (2009); Fu and Hao (2017a).

As to (practical) exact solving, the sophisticated branch-and-cut algorithm by Ljubic et al. (2006) was an early milestone. A survey on the developments before 2006 is given by Costa et al. (2006). Later, the PCSTP attracted considerable interest in the wake of the 11th DIMACS Challenge (DIMACS, 2015)—dedicated to Steiner tree problems—where the PCSTP categories could boast the most participants by far. Furthermore, in the recent years a considerable number of additional solvers for the PCSTP have been introduced, see e.g. Akhmedov et al. (2016); Braunstein and Muntoni (2016); Fischetti et al. (2017); Fu and Hao (2017a); Gamrath et al. (2017); Leitner et al. (2018a); Ming et al. (2018); Sun et al. (2019). Some of these solvers, in particular Leitner et al. (2018a), drastically improve on the best results achieved at the DIMACS Challenge—being able to not only solve many instances orders of magnitude faster, but also to solve a number of instances for the first time to optimality. Exact approaches for PCSTP are usually based on branch-and-bound or branch-and-cut (Fischetti et al., 2017; Gamrath et al., 2017), include specialized (primal and sometimes dual) heuristics (Klau et al., 2004; Leitner et al., 2018a), and make use of various preprocessing methods to reduce the problem size (Ljubic et al., 2006; Leitner et al., 2018a). For more details on the PCSTP literature see the recent survey Ljubic (2020).

4.1.1 Preliminaries and additional notation

Throughout this chapter it will be presupposed that a PCSTP instance $I_{PC} = (V, E, c, p)$ is given such that (V, E) is connected; otherwise, one can optimize each connected component separately. We call $T_p := \{v \in V \mid p(v) > 0\}$ the set of *potential terminals* (Leitner et al., 2018a). It will be assumed that $T_p \neq \emptyset$. For ease of presentation we use $\{t_1, t_2, ..., t_s\} := T_p$, so in particular $s := |T_p|$. A $t \in T_p$ will

[17] By the name *node-weighted Steiner tree problem.*

be called *proper potential terminal* if

$$p(t) > \min_{e \in \delta(t)} c(e). \tag{4.3}$$

The set of all proper potential terminals will be denoted by $T_p^+ = \{t_1^+, t_2^+, ..., t_{s^+}^+\}$, with $s^+ := |T_p^+|$. Accordingly, define $T_p^- := T_p \setminus T_p^+$. The distinction of proper and non-proper potential terminals was already made in Uchoa (2006), where it was noted that non-proper potential terminals allow for additional presolving methods. This distinction can also be found in Fischetti et al. (2017); Leitner et al. (2018a).

We will call any PCSTP solution that consists of just one vertex *trivial*. If $T_p^+ = \emptyset$, then there exists a trivial optimal solution. In general, there exists an optimal solution whose leaves are a subset of T_p^+, or there exists at least one trivial optimal solution.

Finally, we define a variation of the PCSTP, the *rooted prize-collecting Steiner tree problem* (RPCSTP)[18]. The RPCSTP incorporates the additional condition that a non-empty set $T_f \subseteq V$ of *fixed terminals* needs to be part of all feasible solutions. We assume w.l.o.g. that $p(t) = 0$ for all $t \in T_f$.

4.1.2 Contribution and structure

This chapter introduces and analyses new techniques and algorithms for PCSTP that ultimately aim for efficient exact solution. Most of the techniques are based on, or result in reductions (or transformations) of the PCSTP to equivalent problems—these problems can be PCSTPs itself, but can also be from different problem classes. The reductions can for example decrease the problem size or allow us to obtain a stronger IP formulation. Moreover, several of the new methods provably dominate previous results from the literature. While several of the new techniques require to solve $\mathcal{NP}$-hard subproblems, the underlying concepts allow us to design empirically efficient heuristics. Furthermore, we provide complexity results for the exact solution of PCSTP (and related problems), which underpin the design of most algorithms in this chapter. Also these complexity results are based on problem transformations.

A salient feature of this chapter is the intricate interaction of the individual algorithmic components and their wide applicability within a branch-and-cut framework—from preprocessing and probing, to IP formulation and separation methods, to heuristics, domain propagation, and branching. The integration of the new methods into an exact solver also brings significant computational advancements: The new solver is significantly faster than current state-of-the-art competitors, and furthermore solves 24 benchmark instances for the first time to optimality.

The remainder of this chapter is structured as follows.

- Section 4.2 shows that PCSTP is fixed-parameter tractable (FPT) in $|T_p^+|$. Furthermore, we discuss (known and new) transformations from the node-weighted Steiner tree and maximum-weight connected subgraph problem to PCSTP, which directly lead to FPT results. Also, we show that non-proper potential

[18] Note that in the literature it is more common to denote only problems with exactly one fixed terminal as rooted prize-collecting Steiner tree problem, e.g. in Ljubic et al. (2006)

terminals naturally arise from these transformations. Overall, Section 4.2 provides a strong theoretical motivation for distinguishing between proper and non-proper potential terminals within PCSTP algorithms, which will be a dominating theme throughout this chapter.

– Section 4.3 introduces several new reduction techniques for PCSTP. Most importantly, a new distance function based on so-called *prize-constrained walks* is introduced. By using this distance function, we introduce for example a new edge elimination criterion. The new techniques are also compared with previous methods from the literature.

– Section 4.4 makes further use of the concept of prize-constrained walks. By a combination with the reduced-costs of a particular LP relaxation, prize-constrained walks allow us to find vertices that need to be part of any optimal solution. We further show how this information leads to a better IP formulation.

– Section 4.5 shows how to integrate the newly developed algorithms within a branch-and-cut framework. Furthermore, computational results are given, as well as comparisons with state-of-the-art PCSTP solvers.

– Finally, Section 4.6 offers a conclusion, and suggestions on possible future research.

4.2 Proper potential terminals and complexity

In a number of (real-world) PCSTP instances from the literature $|T_p^-|$ is considerably larger than $|T_p^+|$, so it seems well-worthwhile to algorithmically distinguish between proper and non-proper potential terminals. This section provides also a theoretical foundation for such a distinction. Namely, by showing how proper and non-proper potential terminals arise from problems related to PCSTP and by showing how the complexity of PCSTP depends on the number of proper potential terminals.

4.2.1 On the complexity of PCSTP

In the following we demonstrate that for the complexity of PCSTP the number $s^+ = |T_p^+|$ is the crucial parameter. One observes throughout this chapter that the complexity of several new PCSTP algorithms is likewise governed by s^+. We first show the following.

Theorem 4.1. *The PCSTP is fixed-parameter tractable for the parameter s^+. It can be solved in time $O(3^{s^+} n + 2^{s^+} n^2 + n^2 \log n + mn)$.*

Due to its technical nature, a detailed proof is given in Appendix A.3.1 only. In the following we describe the main building blocks. Consider a RPCSTP $I_f = (G, T_f, c, p)$ with $T_p^+ = \emptyset$. By extending the well-known dynamic programming algorithm from Dreyfus and Wagner (1971), we obtain the following result.

Proposition 4.2. *An optimal solution to I_f can be found in time $O(3^{|T_f|}n + 2^{|T_f|}n^2 + n^2 \log n + mn)$.*

Now we return to PCSTP. It will be assumed that no trivial solution exists for PCSTP (otherwise one needs to compare the solution found in the following with the best trivial solution). The following describes how to transform any PCSTP to an equivalent RPCSTP instance that has no proper potential terminals and satisfies $|T_f| = s^+ + 1$.

Transformation 4.3 (PCSTP to RPCSTP)**.**
Input: *PCSTP (V, E, c, p) with $T_p^+ \neq \emptyset$*
Output: *RPCSTP (V', E', T_f', c', p')*

1. *Initially, set $V' := V$, $E' := E$, $c' := c$; define $M := \sum_{t \in T_p^+} p(t)$.*
2. *Define $p' : V' \to \mathbb{Q}_{\geqslant 0}$ for all $v \in V'$ by*

$$p'(v) := \begin{cases} p(v) & \text{if } v \in T_p^-, \\ 0 & \text{otherwise.} \end{cases}$$

3. *Let $j \in \{1, ..., s^+\}$ such that $p(t_j) = \min_{t \in T_p^+} p(t)$.*
4. *Add vertex t_0' to V'.*
5. *For each $i \in \{1, ..., s^+\}$:*
   (a) *add node t_i' with $p(t_i') := 0$ to V';*
   (b) *add edges $\{t_0', t_i\}$ and $\{t_i, t_i'\}$ to E', both of weight M.*
6. *For each $i \in \{1, ..., s^+\} \setminus \{j\}$:*
   (a) *add edge $\{t_i, t_j'\}$ of weight $M + p(t_j)$ to E';*
   (b) *add edge $\{t_i', t_j'\}$ of weight $M + p(t_i)$ to E'.*
7. *Define fixed terminals $T_f' := \{t_1', ..., t_{s^+}'\} \cup \{t_0'\}$.*
8. ***Return*** *(V', E', T_f', c', p').*

Instead of a correctness proof, we give an informal description of the transformation. Let I be a PCSTP. Let I' be the RPCSTP resulting from Transformation 4.3, let S' be an optimal solution to I', and let $S := S' \cap (V, E)$. One observes that S' needs to contain at least one t_i in order to connect t_0' with the remainder of the fixed terminals. Because of the choice of M, one further observes that for each fixed terminal of I' exactly one incident edge is in S'. Each fixed terminal t_i' with $i \neq j$ is either connected via t_i or via t_j'. In the first case, $t_i \in V(S)$ holds, and in the second, $t_i \notin V(S)$. The edge weights are chosen such that the objective value of S' corresponds to that of S in both cases. The fixed terminal t_j' has a special status, as it is used to connect any fixed terminal t'_i with $t_i \notin V(S)$. Thus, t'_j is connected to all t_i. Note that the corresponding t_j needs to be of minimum prize among all proper potential terminals for the transformation to be correct. Overall, S is an optimal solution to I, and one obtains the following relation

$$C(S) = C(S') - |T'_f|M.$$

By combining Proposition 4.2 and Transformation 4.3, one obtains Theorem 4.1.

Interestingly, one can also easily extend Transformation 4.3 such that the result is an SPG with $s+1$ terminals (and at most $2n+1$ vertices). To do so, one needs to essentially consider all potential terminals as proper potential terminals. There has been no transformation in the literature so far from SPG to PCSTP.

However, the structure of the resulting SPG does not lend itself well to an efficient practical solution by state-of-the-art SPG algorithms. Still, one can use this transformation to directly derive further complexity results for PCSTP from SPG. E.g., Vygen (2011) shows that an SPG with k terminals can be solved in time $O(nk2^{k+\log_2 k \log_2 n})$, which translates to $O(ns2^{s+\log_2 s \log_2 n + \log_2 s})$ for PCSTP. One could also extend the result from Vygen (2011) (by verifying them for I_f similarly to Proposition 4.2) to show that the same bound holds for s^+.

Having demonstrated that PCSTP is tractable if the number of proper potential terminals is bounded from above, we now turn to the opposite case. The SPG is well-known to be fixed-parameter tractable in $n - |T|$, which can be shown by enumeration of the non-terminal vertices (Hakimi, 1971). For node-weighted Steiner tree and maximum-weight connected subgraph problems one can show similar results (Buchanan et al., 2018). However, the situation for PCSTP with respect to $n - s^+$ is different, as the following proposition shows.

Proposition 4.4. *PCSTP is $\mathcal{NP}$-hard even if $s^+ = n$.*

Proof. We show that the—$\mathcal{NP}$-complete (Garey and Johnson, 1979)—vertex cover problem can be reduced to the decision variant of PCSTP, such that the resulting instance satisfies $s^+ = n$. Let $G_{cov} = (V_{cov}, E_{cov})$ be an undirected graph and $k \in \mathbb{N}$. In the vertex cover problem one has to determine whether a subset of V_{cov} of cardinality at most k exists that is incident to all edges E_{cov}. Let $n := |V_{cov}|$ and $m := |E_{cov}|$. Assume that the vertices and edges of G_{cov} are given as $\{v_1, v_2, ..., v_n\}$ and $\{e_1, e_2, ..., e_m\}$, respectively. Construct a PCSTP instance $I' = (V', E', c', p')$ with $2n+m+1$ vertices and $2n+2m$ edges as follows. Denote the vertices of V' by u'_i and v'_i for $i = 1, ..., n$, and w'_i for $i = 0, 1, ..., m$. For each original edge $e_i = \{v_j, v_k\} \in E_{cov}$ create the two edges $\{w'_i, v'_j\}$ and $\{w'_i, v'_k\}$ with cost $c'(\{w'_i, v'_j\}) = c'(\{w'_i, v'_k\}) = 8$. For each original vertex $v_i \in V_{cov}$ create the two edges $\{v'_i, u'_i\}$ and $\{u'_i, w'_0\}$ with cost $c(\{v'_i, u'_i\}) = 1$ and $c(\{u'_i, w'_0\}) = 4$. Finally, define the prizes of I' as follows. First, $p'(w'_i) = 10$ for $i = 0, 1, ..., m$. Second, $p'(v'_i) = p'(u'_i) = 2$ for $i = 1, ..., n$. Observe that all vertices in V' are proper potential terminals. We claim that an independent set for G_{cov} of cardinality at most k exists if and only if there is a tree $S' \subseteq (V', E')$ that satisfies

$$C(S') - 8m - 4n \leqslant k. \tag{4.4}$$

First, assume that a vertex cover with vertex index set J_{cov} exists such that $|J_{cov}| \leqslant k$. Build a tree $S' \subseteq (V', E')$ as follows. Initially, set

$$V'(S') := \{v'_j, u'_j \mid j \in J_{cov}\} \cup \{w_j \mid j \in \{0, ..., m\}\}.$$

For $E'(S')$ take all edges $\{v'_j, u'_j\}, \{u'_j, w'_0\}$ with $j \in J_{cov}$. Furthermore, for $i = 1, ..., m$ add exactly one edge $\{w'_i, v'_j\}$ with $j \in J_{cov}$. For S' one observes that

$$\sum_{e \in E'(S')} c'(e) = 8m + 5k$$

and

$$\sum_{v \in V' \setminus V'(S')} p'(v) = 4(n - k).$$

Thus, $C(S') - 8m - 4n = k$.

Conversely, assume that a tree S' exists that satisfies (4.4). Assume that S' is an optimal solution to I'. One verifies that S' contains all vertices w'_i (e.g. by using Theorem 4.22). Note that also $\delta_{S'}(w'_i) = 1$ for $i = 1, ..., m$. Let $J_{cov} \subseteq \{1, ..., n\}$ such that $v'_j \in S' \iff j \in J_{cov}$ and set $k' := |J_{cov}|$. From the optimality of S' one obtains

$$C(S') = 8m + 5k' + 4(n - k'). \tag{4.5}$$

From (4.4) it follows that $k' \leqslant k$. □

4.2.2 From PCSTP to MWCSP and NWSTP

The distinction of proper potential terminals also arises in relation with the maximum-weight connect subgraph problem (MWCSP), which was introduced in the previous chapter. We re-define the MWCSP with a slightly different notation here. Given an undirected graph $G = (V, E)$ with node weights $w : V \to \mathbb{Q}$, the MWCSP asks for a connected subgraph $S \subseteq G$ that maximizes

$$\sum_{v \in V(S)} w(v). \tag{4.6}$$

Let $I = (V, E, w)$ be an MWCSP instance and assume that $w_0 := \min_{v \in V(S)} w(v)$ is negative (otherwise I is trivial to solve). I can be transformed to an equivalent PCSTP $I' = (V, E, c, p)$ by setting $c(e) := -w_0$ for all $e \in E$, and $p(v) := w(v) - w_0$ for all $v \in V$, as described in Dittrich et al. (2008). It should be noted, though, that due to the special form of I', the state-of-the-art MWCSP algorithms introduced in the previous chapter perform significantly better in practice on I than PCSTP algorithms on I'. As to proper potential terminals, one observes the following: For any $v \in V$ it holds that $w(v) > 0$ (in I) if and only if v is a proper potential terminal in I'. Thus, one also obtains the following corollary (which improves on a result from Buchanan et al. (2018)).

Corollary 4.5. *MWCSP can be solved in time* $O(3^q n + 2^q n^2 + n^2 \log n + mn)$*, where* q *denotes the number of positive weight vertices.*

Another natural distinction between proper and non-proper potential terminals can be observed for the *node-weighted Steiner tree problem* (NWSTP), see e.g. Klein and Ravi (1995). Given an undirected, connected graph $G = (V, E)$ with vertex weights $w : V \to \mathbb{Q}_{\geqslant 0}$ and edge weights $c : E \to \mathbb{Q}_{\geqslant 0}$, and given a set of *terminals* $T \subseteq V$, the NWSTP asks for tree $S \subseteq G$ with $T \subseteq V(S)$ that minimizes

$$\sum_{e \in E(S)} c(e) + \sum_{v \in V(S)} w(v). \tag{4.7}$$

Let $I = (V, E, T, c, w)$ be an NWSTP instance and assume w.l.o.g. that $w(t) = 0$ for all $t \in T$. I can be reduced to an equivalent PCSTP $I' = (V, E, c', p')$ by the following, new, transformation. Let $z := \max_{v \in V} w(v)$. Define $c'(e) := c(e) + z$ for all $e \in E$. Define $p'(t) := k$ for all $t \in T$, with a sufficiently large $k \in \mathbb{Q}_{\geqslant 0}$, e.g. $k = \sum_{e \in E} c'(e)$. Finally, define $p'(v) = z - w(v)$ for all $v \in V \setminus T$. A tree S is an optimal solution to I if and only if it is an optimal solution to I'. Furthermore, in this case S satisfies

$$\sum_{e \in E(S)} c(e) + \sum_{v \in V(S)} w(v) = C(S) - (|T| - 1)z - \sum_{v \in V} p'(v). \tag{4.8}$$

Note that $T_p^- \supseteq \{v \in V \setminus T \mid w(v) < z\}$. Likewise, the set of terminals T for I corresponds to the set of proper potential terminals for I'.

One can alternatively transform NWSTP to RPCSTP to avoid the use of the large constant k. Also, one immediately obtains the following corollary (which was already shown algorithmically in Buchanan et al. (2018)) from Proposition 4.2.

Corollary 4.6. *NWSTP can be solved in time* $O(3^k n + 2^k n^2 + n^2 \log n + mn)$*, where* k *denotes the number of terminals.*

Finally, one notes that PCSTP can be seen as a generalization of both MWCSP and NWSTP, as these problems can be transformed to a PCSTP with the same number of edges and vertices.

4.3 Reductions within the problem class

The methods described in the following aim to reduce a given instance to a smaller one of the same problem class. Several articles have addressed such techniques for the PCSTP, e.g. Leitner et al. (2018a); Ljubic et al. (2006); Uchoa (2006), but most are dominated by the methods described in the following. In particular, several simple reduction techniques described in the literature are just special cases of the algorithms described in the following. See Rehfeldt et al. (2019) for more details. The new methods will not only be employed for classic preprocessing, but also throughout the entire solving process, e.g. for domain propagation, or within heuristics.

4.3.1 Taking short walks

The following approach uses a new, walk-based, distance function. It generalizes the bottleneck Steiner distance concept that was the central theme of Uchoa (2006) and is defined as follows. Let $v, w \in V$ be two distinct vertices. Denote by $\mathcal{P}(v, w)$ the set of all paths between v and w. Recall that we assume all paths to be simple. For a path P and vertices $x, y \in V(P)$ let $P(x, y)$ be the subpath of P between x and y. Define the *Steiner distance* of path P as

$$SD(P) := \max_{x,y \in V(P)} \sum_{e \in E(P(x,y))} c(e) - \sum_{v \in V(P(x,y)) \setminus \{x,y\}} p(v). \tag{4.9}$$

With this definition at hand, define the *bottleneck Steiner distance* between v and w as

$$B(v, w) := \min\{SD(P) \mid P \in \mathcal{P}(v, w)\}. \tag{4.10}$$

Now, we describe a stronger distance concept, which is closely related to the implied bottleneck Steiner distance that we introduced for the SPG in Section 2.3.1. Let $v, w \in V$. A finite walk $W = (v_1, e_1, v_2, e_2, ..., e_{r-1}, v_r)$ with $v_1 = v$ and $v_r = w$ will be called *prize-constrained* (v, w)*-walk* if no $v \in T_p^+ \cup \{v, w\}$ is contained more than once in W. For any $k, l \in \mathbb{N}$ with $1 \leqslant k \leqslant l \leqslant r$ define the subwalk $W(v_k, v_l) := (v_k, e_k, v_{k+1}, e_{k+1}, ..., e_{l-1}, v_l)$; note that $W(v_k, v_l)$ is again a prize-constrained walk. In the following, let W be a prize-constrained (v, w)-walk. Define the *prize-collecting cost* of W as

$$c_{pc}(W) := \sum_{e \in E(W)} c(e) - \sum_{u \in V(W) \setminus \{v,w\}} p(u). \tag{4.11}$$

Thereupon, define the *prize-constrained length* of W as

$$l_{pc}(W) := \max\{c_{pc}(W(v_k, v_l)) \mid 1 \leqslant k \leqslant l \leqslant r,\ v_k, v_l \in T_p^+ \cup \{v, w\}\}. \tag{4.12}$$

Intuitively, $l_{pc}(W)$ provides the cost of the least profitable subwalk of W. This measure will in the following be useful to bound the cost of connecting any two disjoint trees that contain the first and the last vertex of W, respectively. Finally, we denote the set of all prize-constrained (v, w)-walks by $\mathcal{W}_{pc}(v, w)$ and define the *prize-constrained distance* between v and w as

$$d_{pc}(v, w) := \min\{l_{pc}(W) \mid W \in \mathcal{W}_{pc}(v, w)\}. \tag{4.13}$$

Note that $d_{pc}(v, w) = d_{pc}(w, v)$ for any $v, w \in V$. Also, it is important to note that for each subwalk the cost of an edge and the prize of an inner vertex are counted exactly once, even if an edge or vertex is contained multiple times in the subwalk. Using the same measuring concept, one could in fact also allow arbitrary finite walks instead of prize-constrained ones—and count for each subwalk the costs of its edges and the prizes of its inner vertices exactly once. However, the prize-constrained distance is already arbitrarily stronger than the bottleneck Steiner distance, and additionally more closely related to the algorithms described below.

Furthermore, one could also apply the concept of implied profit, introduced for the SPG in Section 2.3.1, to further improve the prize-constrained distance. Still, adaptations are necessary to also take the prize of the potential terminal that induces the profit into account. We do not consider this improvement in the following, as it would considerably complicate the theoretical analyses. Also, this improved version has not been implemented yet.

By using the prize-constrained distance one can formulate a reduction criterion that dominates the *special distance* test from Uchoa (2006). This criterion is expressed in the following theorem:

Theorem 4.7. *Let $\{v,w\} \in E$. If*

$$c(\{v,w\}) > d_{pc}(v,w) \tag{4.14}$$

is satisfied, then $\{v,w\}$ cannot be contained in any optimal solution to I_{PC}.

Proof. Let S be a tree with $\{v,w\} \in E(S)$. Further, let $W = (v_1, e_1, ..., e_{r-1}, v_r)$ be a prize-constrained (v,w)-walk with $l_{pc}(W) = d_{pc}(v,w)$. Remove $\{v,w\}$ from S to obtain two new trees. Of these two trees denote the one that contains v by S_v, and the other (containing w) by S_w. Define $b := \min\{k \in \{1,...,r\} \mid v_k \in V(S_w)\}$ and $a := \max\{k \in \{1,...,b\} \mid v_k \in V(S_v)\}$. Further, define $x := \max\{k \in \{1,...,a\} \mid v_k \in T_p^+ \cup \{v\}\}$ and $y := \min\{k \in \{b,...,r\} \mid v_k \in T_p^+ \cup \{w\}\}$. By definition, $x \leqslant a < b \leqslant y$ and furthermore:

$$c_{pc}(W(v_a, v_b)) \leqslant c_{pc}(W(v_x, v_y)). \tag{4.15}$$

Reconnect S_v and S_w by $W(v_a, v_b)$, which yields a new tree S'. For this tree it holds that:

$$\begin{aligned}
C(S') &\leqslant C(S) + c_{pc}(W(v_a, v_b)) - c(\{v,w\}) \\
&\overset{(4.15)}{\leqslant} C(S) + c_{pc}(W(v_x, v_y)) - c(\{v,w\}) \\
&\leqslant C(S) + l_{pc}(W) - c(\{v,w\}) \\
&= C(S) + d_{pc}(v,w) - c(\{v,w\}) \\
&\overset{(4.14)}{<} C(S).
\end{aligned}$$

Because of $C(S') < C(S)$ no optimal solution can contain $\{v,w\}$. □

To obtain a criterion for the case of equality in (4.14), define $d^-_{pc}(v,w)$ as the prize-constrained distance with respect to the PCSTP $(V, E \setminus \{e\}, c, p)$, with $e := \{v,w\}$. If v and w are disconnected in $(V, E \setminus \{e\}, c, p)$, define $d^-_{pc}(v,w) := \infty$. With this definition at hand, one obtains the following corollary to Theorem 4.7.

Corollary 4.8. *Let $e = \{v,w\} \in E$. If*

$$c(e) \geqslant d^-_{pc}(v,w) \tag{4.16}$$

is satisfied, then e is not contained in at least one optimal solution to I_{PC}.

Figure 4.1 shows a PCSTP instance on which Theorem 4.7 allows one to eliminate an edge. Only vertex v_4 has a non-zero prize. Consider the (dashed) edge $\{v_1, v_2\}$. For the prize-constrained (v_1, v_2)-walk

$$W = (v_1, \{v_1, v_3\}, v_3, \{v_3, v_4\}, v_4, \{v_3, v_4\}, v_3, \{v_2, v_3\}, v_2)$$

it holds that $d_{pc}(v_1, v_2) = l_{pc}(W) = 6$.

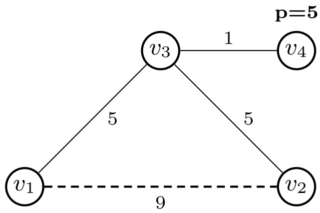


Figure 4.1: PCSTP instance. Edge $\{v_1, v_2\}$ (dashed) can be eliminated due to Theorem 4.7.

Algorithms for the prize-constrained distance

Since computing the bottleneck Steiner distance is already $\mathcal{NP}$-hard (Uchoa, 2006), it does not come as a surprise that the same holds for d_{pc} (which can be shown in the same way). However, the definition of d_{pc} allows us to design a simple algorithm for finding upper bounds that yields empirically strong results—significantly better than those reported by Uchoa (2006).

In particular, while the bottleneck Steiner distance heuristics in Uchoa (2006) consider only paths with at most two intermediary potential terminals, the following algorithm can find walks where the number of intermediary potential terminals is only bounded by $|T_p|$. Besides dual-ascent (Rehfeldt et al., 2019), the bottleneck Steiner distance has been the most important reduction concept for PCSTP. Due to this importance, we will take a deeper look at the prize-constrained distance, as well as at associated algorithms for computing upper bounds.

To check whether an edge $\{v, w\}$ can be deleted by means of criterion (4.16), we suggest the procedure detailed in Algorithm 4.1, which is based on Dijkstra's algorithm (Dijkstra, 1959). The algorithm is given the edge $e = \{v, w\}$ as well as one of its endpoints, say v, from which it computes prize-constrained walks of length not higher than $c(e)$. The algorithm starts with a priority queue that contains v. In contrast to Dijkstra's algorithm, all vertices except for potential terminals and v can be reinserted into the priority queue after they have been removed. We associate with each vertex u the distance $dist_{pc}[u]$—initially set to 0 for v and to ∞ otherwise. As with Dijkstra's algorithm, in each iteration one vertex u with minimum distance value is removed from the priority queue and neighboring vertices of u are updated. However, a different distance value than in Dijkstra's algorithm is used and a neighboring vertex q is only updated if $dist_{pc}[u] + c(\{u, q\}) \leqslant c(e)$.

Throughout the computation the following invariant is satisfied for any $u \in V \setminus \{v\}$: either $dist_{pc}[u] = \infty$ or $dist_{pc}[u] + p(u) \leqslant c(e)$, and in the latter case there exists a prize-constrained (v, u)-walk W_u such that $l_{pc}(W_u) \leqslant c(e)$.

Algorithm 4.1: PcdCheck-Edge

Data: PCSTP (V, E, c, p), edge $\{v_{start}, v_{end}\} \in E$
Result: *deletable* if edge has shown to be redundant, *unknown* otherwise

$Q := \{v_{start}\}$
$E_0 := E \setminus \{\{v_{start}, v_{end}\}\}$
$c_0 := c(\{v_{start}, v_{end}\})$
foreach $v \in V \setminus \{v_{start}\}$ **do**
    $dist_{pc}[v] := \infty$
    $forbidden[v] := false$
$dist_{pc}[v_{start}] := 0$
$forbidden[v_{start}] := true$
while $Q \neq \emptyset$ **do**
    $v := \arg\min_{u \in Q} dist_{pc}[u]$
    $Q := Q \setminus \{v\}$
    if $v \in T_p$ **then**
        $forbidden[v] := true$
    foreach $w \in V$ *with* $\{v, w\} \in E_0$ **do**
        if $forbidden[w] = false$ **and** $dist_{pc}[v] + c(\{v, w\}) \leqslant c_0$ **and** $dist_{pc}[v] + c(\{v, w\}) - p(w) < dist_{pc}[w]$ **then**
            if $w = v_{end}$ **then**
                return *deleteable*
            $dist_{pc}[w] := \max\{0, c(\{v, w\}) + dist_{pc}[v] - p(w)\}$
            if $w \notin Q$ **then**
                $Q := Q \cup \{w\}$
return *unknown*

One obtains the following results concerning the strength of the prize-constrained distance, and the upper bound provided by Algorithm 4.1.

Proposition 4.9. *For any two vertices $v, w \in V$ it holds that*

$$d_{pc}(v, w) \leqslant B(v, w). \tag{4.17}$$

Furthermore, let $\mathcal{I}_{pc}$ be the set of all PCSTP instances. It holds that

$$\sup_{(V,E,p,c) \in \mathcal{I}_{pc}} \max_{v,w \in V} \frac{B(v, w)}{d_{pc}(v, w)} = \infty. \tag{4.18}$$

Proof. The relation (4.17) can be verified by the definitions of B and d_{pc}. For the second part consider the PCSTP depicted in Figure 4.2. Let $n \in \mathbb{N}, n \geqslant 3$. The prizes

p and costs c are defined as follows: $p(v) = 0$ for $i = 0, ..., n$, and $p(w_i) = 3$ for $i = 1, ..., n-1$; further, $c \equiv 1$. Observing that

$$\lim_{n\to\infty} \frac{B(v_0, v_n)}{d_{pc}(v_0, v_n)} = \lim_{n\to\infty} \frac{n}{3} = \infty,$$

one can validate that (4.18) holds. □

Next, we show that (the heuristic) Algorithm 4.1 can yield better results than the (exact) bottleneck Steiner distance. To this end, first define B^- analogously to d^-_{pc}. The next corollary shows that the results from Algorithm 4.1 can be (in a relative sense) arbitrarily better than the bottleneck Steiner distance.

Corollary 4.10. *For any $K \in \mathbb{N}_0$ there is a PCSTP instance I_K with an edge $e = \{v, w\}$ such that*

$$\frac{B^-(v, w)}{c(e)} \geqslant K,$$

while Algorithm 4.1 returns deletable for $(I_K, \{v, w\})$.

Proof. If $K = 0$, condition (4.10) is trivially satisfied. Assume $K \geqslant 1$, and consider the PCSTP instance in Figure 4.2 for $n = 3K$. Add an arc $\{v_0, v_n\}$ of cost 3 to the instance. For this PCSTP together with the edge $\{v_0, v_n\}$ the PCD algorithm returns *deletable*. On the other hand, $B^-(v_0, v_n) = 3K$. □

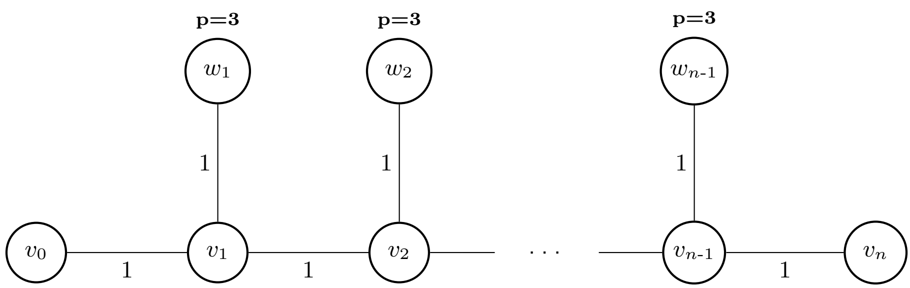


Figure 4.2: PCSTP instance such that the ratio of the bottleneck Steiner distance and the prize-constrained distance between v_0 and v_n becomes arbitrarily large.

Since Algorithm 4.1 runs in polynomial time and the decision variant of the prize-constrained distance is $\mathcal{NP}$-complete, Algorithm 4.1 cannot be in general exact—in the sense that it might return *unknown* even though $c(\{v_{start}, v_{end}\}) \geqslant d^-_{pc}(v_{start}, v_{end})$—unless $\mathcal{P} = \mathcal{NP}$. However, under certain conditions Algorithm 4.1 is exact, as detailed in the following proposition (see Appendix A.3.2 for a proof).

Proposition 4.11. *Let* $\{v, w\} \in E$ *and let* W *be a* (v, w)*-walk with* $l_{pc}(W) = d^-_{pc}(v, w)$*. If for all* $t \in (V(W) \setminus \{v, w\}) \cap T^+_p$

$$p(t) \leqslant \min_{e \in \delta(t) \cap E(W)} c(e) \tag{4.19}$$

holds, then Algorithm 4.1 returns deletable if and only if $c(\{v, w\}) \geqslant d^-_{pc}(v, w)$*.*

Corollary 4.12. *Let* $v, w \in V$*. If* $T^+_p \subseteq \{v, w\}$ *holds (which includes* $T^+_p = \emptyset$*), then both* $d^-_{pc}(v, w)$ *and* $d_{pc}(v, w)$ *can be computed in polynomial time (with respect to the encoding size of* I_{PC}*).*

To check whether an edge e can be eliminated, we run a restricted version of Algorithm 4.1 (which only checks at most a fixed number of edges during its execution) from both endpoints of e. If none of the two tests are successful, we check for each vertex that has been visited in both runs whether the corresponding walks can be combined to obtain a walk that allows to delete e. This procedure will be referred to as *prize-constrained distance* (*PCD*) *test*.

We have also implemented an extension of PCD, referred to as *extended prize-constrained distance* (*EPCD*) *test*, which will be sketched in the following. One downside of PCD, even in its unrestricted form, is that once a potential terminal has been removed from the priority queue, it cannot be used in any other walk. Thus, EPCD keeps for each vertex v a (bounded) list of potential terminals that are part of the current (v_{start}, v)-walk. Whenever a vertex w could be updated from a vertex v, but $forbidden[w] = false$, it is checked whether w is in the potential terminal list of v, and if not, w is still updated. Another problem of PCD is that the cost of an edge on a subwalk might be counted several times. Consider for example the instance described in the proof of Corollary 4.10 and change the prizes for all w_i to $p(w_i) := 2$. It still holds that $d^-_{pc}(v_0, v_n) = 3$, but PCD for edge $\{v_0, v_n\}$ will only return *deletable* if $c(\{v_0, v_n\}) \geqslant n$. Therefore, EPCD saves for each vertex (a limited number of) edges on the current subwalk. This list is cleared as soon as the distance value of a vertex is set to 0. EPCD also allows that non-proper potential terminals can be used several times on one walk, by keeping a similar list of non-proper potential terminals on the current subwalk.

Further applications of the prize-constrained distance

The prize-constrained distance can be combined with a related walk-based concept introduced in Section 4.4 to obtain a reduction method that allows for the contraction of edges. Consider a graph (V, E) with non-negative edge costs c. Let P be a (simple) path. The *bottleneck length* (Duin and Volgenant, 1989a) of P is

$$bl(P) := \max_{e \in E(P)} c(e). \tag{4.20}$$

For vertices $v, w \in V$, the *bottleneck distance* (Duin and Volgenant, 1989a) is defined as

$$b(v, w) := \inf\{bl(P) \mid P \in \mathcal{P}(v, w)\}. \tag{4.21}$$

Note that if all vertices have a sufficiently large prize, then $b(v, w) = d_{pc}(v, w)$ for all $v, w \in V$, but in general $b(v, w) \leqslant d_{pc}(v, w)$ holds. For a fixed edge $e \in E$ we define the *restricted bottleneck distance* $b_e(v, w)$ as the bottleneck distance on $(V, E \setminus \{e\})$. This definition gives rise to the following proposition.

Proposition 4.13. *Let $\{v, w\} \in E$ and $t_i, t_j \in T_p, t_i \neq t_j$ such that:*

1. *If an optimal solution S with $t_i \in V(S)$ exists, then there is an optimal solution $S' \supseteq S$ with $t_j \in V(S')$.*
2. *If an optimal solution S with $v \in V(S)$ or $w \in V(S)$ exists, then there is an optimal solution $S' \supseteq S$ with $t_i \in V(S')$.*

If furthermore it holds that

$$d_{pc}(v, t_i) + c(\{v, w\}) + d_{pc}(w, t_j) \leqslant b_{\{v,w\}}(t_i, t_j), \tag{4.22}$$

then for any optimal solution S with $v \in V(S)$, $w \in V(S)$, or $t_i \in V(S)$ there is an optimal solution S' with $\{v, w\} \in E(S')$.

Proof. Assume there is an optimal solution S such that $v \in V(S)$ or $w \in V(S)$. Because of the first two conditions of the proposition we can assume that $v \in V(S)$, $w \in V(S)$, $t_i \in V(S)$, and $t_j \in V(S)$. Suppose $\{v, w\} \notin E(S)$. Remove from $E(S)$ an edge on the (unique) path between t_i and t_j in S of maximum cost. This operation results in two disjoint trees: S_i with $t_i \in S_i$ and S_j with $t_j \in S_j$. By definition of $b_{\{v,w\}}(t_i, t_j)$ it holds that

$$C(S_i) + C(S_j) + b_{\{v,w\}}(t_i, t_j) \leqslant C(S). \tag{4.23}$$

Similarly to the proof of Theorem 4.7, condition (4.22) allows us to connect S_i to v such that the resulting tree $\tilde{S}_i$ satisfies

$$C(\tilde{S}_i) \leqslant C(S_i) + d_{pc}(v, t_i). \tag{4.24}$$

Equivalently, we can connect S_j to w with the result satisfying

$$C(\tilde{S}_j) \leqslant C(S_j) + d_{pc}(w, t_j). \tag{4.25}$$

Finally, we define $\tilde{S}$ as the union of $\tilde{S}_i$, $\tilde{S}_j$ and $\{v, w\}$. This connected subgraph is not necessarily a tree, but can be made one without increasing $C(\tilde{S})$ by deleting an edge from each cycle. It holds that

$$\begin{aligned}
C(S) &\overset{(4.23)}{\geqslant} C(S_i) + C(S_j) + b_{\{v,w\}}(t_i, t_j) && (4.26)\\
&\overset{(4.22)}{\geqslant} C(S_i) + C(S_j) + d_{pc}(v, t_i) + d_{pc}(w, t_j) + c(\{v, w\}) && (4.27)\\
&\overset{(4.24)}{\geqslant} C(\tilde{S}_i) + C(S_j) + d_{pc}(w, t_j) + c(\{v, w\}) && (4.28)\\
&\overset{(4.25)}{\geqslant} C(\tilde{S}_i) + C(\tilde{S}_j) + c(\{v, w\}) && (4.29)\\
&\geqslant C(\tilde{S}). && (4.30)
\end{aligned}$$

The case that there is an optimal solution S with $t_i \in V(S)$ can be handled in the same way. $\square$

Note that the contraction of edges for PCSTP is not as straightforward as for SPG, since determining whether the contracted edge is part of any optimal solution is $\mathcal{NP}$-complete. Thus, one needs to adapt the prize of vertex t_i from Proposition 4.13. We discuss this technical issue in Rehfeldt et al. (2019). We just note here that if the conditions of the proposition are satisfied, and $p(t_i) \geqslant c(\{v,w\})$, one can contract $\{v,w\}$ and subtract $c(\{v,w\})$ from $p(t_i)$.

We also remark that Proposition 4.13 is similar to Proposition 2.14, which we established in Section 2.3.1 for the SPG. However, for the SPG the conditions (1) and (2) are not required. To check whether (1) and (2) hold, the *left-rooted prize-constrained distance* introduced in Section 4.4 will be used. Finally, the following lemma allows one to efficiently check the test condition of Proposition 4.13 for all edges, similarly to the corresponding SPG result.

Lemma 4.14. *Let $\{v,w\} \in E$ and $t_i, t_j \in T_p, t_i \neq t_j$. If (4.22) holds, then there is a minimum spanning tree S_{MST} on (V,E,c) such that $\{v,w\} \in E(S_{MST})$.*

A proof of the lemma can be found in Appendix A.3.3.

Finally, a great advantage of the prize-constrained distance over the implied bottleneck Steiner distance that we introduced for SPG is that the former can be directly utilized for a generalization of the classic node replacement test from Duin and Volgenant (1989b).

Proposition 4.15. *Let $v \in V \setminus T_p$. There is an optimal solution S with $|\delta_S(v)| \leqslant 2$ if for each $\Delta \subseteq N(v)$ with $|\Delta| \geqslant 3$ the following holds: $c(\delta(v) \cap \delta(\Delta))$ is not less than the weight of a minimum spanning tree for the distance network $D_G(\Delta, d_{pc})$.*

The proof of the proposition follows the familiar reconnection pattern and is omitted here.

4.3.2 Using bounds

Bound-based reduction techniques identify edges and vertices for elimination by examining whether they induce a lower bound that exceeds a given upper bound (Hwang et al., 1992). In this section, we will introduce several new bound-based reduction methods for PCSTP, based on the following decomposition concept.

For any $U \subseteq V$ such that $T_p^+ \subseteq U$, and for any $v_i, v_j \in V$ let $\mathcal{Q}_U(v_i,v_j)$ be the set of all (v_i,v_j)-paths in the graph induced by $V \setminus (U \setminus \{v_i,v_j\})$. Define $\underline{d}_U : V \times V \mapsto \mathbb{Q}_{\geqslant 0} \cup \{\infty\}$ as

$$\underline{d}_U(v_i,v_j) := \inf_{Q \in \mathcal{Q}_U(v_i,v_j)} \sum_{e \in E(Q)} c(e) - \sum_{v \in V(Q) \setminus \{U \cup \{v_i\}\}} p(v), \tag{4.31}$$

with the common convention $\inf \emptyset := \infty$. The distance function defined in (4.31) will later be used to bound the cost of connecting a vertex $v_i \in V \setminus U$ to a vertex $v_j \in U$. Let $v_i \in V$. Define $\underline{v}^U_{i,0} := v_i$, and, recursively, for $k \in \mathbb{N}$

$$\underline{v}^U_{i,k} := \arg\min\{\underline{d}_U(v_i, v) \mid v \in U \setminus \cup_{j=0}^{k-1}\{\underline{v}^U_{i,j}\}\}, \tag{4.32}$$

assuming that such a vertex exists. So $\underline{v}^U_{i,k}$ is the k-th closest vertex from v_i in U with respect to the distance function $\underline{d}_U$.

With these definitions at hand, we introduce the following concept: a *terminal-regions decomposition* of I_{PC} is a partition $(H_0, H_1, H_2, ..., H_{s^+})$ of V such that for $i = 1, ..., s^+$ it holds that $T_p^+ \cap H_i = \{t_i^+\}$ and that the subgraph induced by H_i is connected—recall that $T_p^+ = \{t_1^+, t_2^+, ..., t_{s^+}^+\}$. Note that H_0 does not need to be connected. We say that H_i is the *terminal region* of t_i. Define $H^p := T_p^+ \cup (H_0 \cap T_p^-)$. Further, define $r_H^{pc} : T_p^+ \mapsto \mathbb{Q}_{\geqslant 0}$ by

$$r_H^{pc}(t_i^+) := \min\left\{p(t_i^+), \min\{\underline{d}_{H^p}(t_i^+, v) \mid v \notin H_i\}\right\} \tag{4.33}$$

for $t_i^+ \in T_p^+$. We will refer to this value as the *prize-collecting radius* of H_i. The idea of this radius concept is to bound the cost of connecting a proper potential terminal t_i with a vertex $v \notin H_i$ within an optimal solution, or the prize that is lost if t_i is not included in the solution. The decomposition can easily be reduced to SPG by using $\min\{d(t_i, v) \mid v \notin H_i\}$ instead of $r_H^{pc}(t_i)$, which corresponds to setting sufficiently high prizes for each terminal of the SPG. Indeed, in this way we obtain the terminal-regions decomposition concept that we introduced in Section 2.3.2. For ease of presentation assume $r_H^{pc}(t_i^+) \leqslant r_H^{pc}(t_j^+)$ for $1 \leqslant i < j \leqslant s^+$. Also, we assume that in the following a fixed terminal-regions decomposition H is given.

Proposition 4.16. *Let $v_i \in V \setminus T_p^+$. If $v_i \in V(S)$ for all optimal solutions S, then a lower bound on $C(S)$ is defined by*

$$\underline{d}_{H^p}(v_i, \underline{v}^{H^p}_{i,1}) + \underline{d}_{H^p}(v_i, \underline{v}^{H^p}_{i,2}) + \sum_{k=1}^{s^+-2} r_H^{pc}(t_k^+) + \sum_{t \in T_p^- \setminus \{v_i\}} p(t). \tag{4.34}$$

Before providing a formal proof, we remark that the bound (4.34) can be motivated by the following two observations. First, there is always an optimal solution S such that v_i is connected to two distinct proper potential terminals by edge disjoint paths in S. The cost of these paths is bounded by $\underline{d}_{H^p}(v_i, \underline{v}^{H^p}_{i,1}) + \underline{d}_{H^p}(v_i, \underline{v}^{H^p}_{i,2})$. Second, for all other proper potential terminals t_i contained in S, there needs to be a path in S from t_i to a vertex in $V(S) \setminus H_i$. To bound the cost of these paths, the prize-collecting radius values are used.

Proof of Proposition 4.16. Initially, define $b : V \rightarrow \{0, 1, 2, ..., s^+\}$ such that $v \in H_{b(v)}$ for all $v \in V$. Assume that v_i is contained in all optimal solutions. This assumption implies that $|T_p^+| \geqslant 2$. Let S be any optimal solution. Denote the (unique) path in S between v_i and any $t_j \in V(S) \cap H^p$ by Q_j and the set of all such paths by $\mathcal{Q}$. First, we can assume that $|\mathcal{Q}| \geqslant 2$, because if $\mathcal{Q}$ just contained one path, say Q_k, then we could simply remove v_i from Q_k to obtain another optimal solution. Second, if a vertex v_k is contained in two distinct paths in $\mathcal{Q}$, the subpaths of these two paths between v_i and v_k coincide. Otherwise there would need to be a cycle in S. Additionally, there are at least two paths in $\mathcal{Q}$ having only the vertex v_i in common. Otherwise, due to the precedent observation, all paths would have one edge $\{v_i, v_i'\}$ in common. This edge could be discarded to yield a tree of smaller cost than $C(S)$.

Let $Q_k \in \mathcal{Q}$ and $Q_l \in \mathcal{Q}$ be two distinct paths with $V(Q_k) \cap V(Q_l) = \{v_i\}$ such that

$$\big|\{\{v,w\} \in E(Q_k) \cup E(Q_l) \mid b(v) \neq b(w)\}\big| \tag{4.35}$$

is minimized. Define $\mathcal{Q}^- := \mathcal{Q} \setminus \{Q_k, Q_l\}$. Consider a (t, v_i)-path $Q_r \in \mathcal{Q}^-$. If $t \in T_p^+$, let Q_r' be the subpath of Q_r between t and the first vertex not in the region of t. Suppose that Q_k has an edge $e \in E(S)$ in common with a Q_r': Consequently, Q_l cannot have any edge in common with Q_r, because this would require a cycle in S. Furthermore, Q_k and Q_r have to contain a joint subpath including v_i and e. But this would imply that Q_k contained at least one additional edge $\{v_x, v_y\}$ with $b(v_x) \neq b(v_y)$. Thus, Q_r would have initially been selected instead of Q_k.

Following the same line of argumentation, one validates that Q_l has no edge in common with any Q_r'. Conclusively, the paths Q_k, Q_l, and all Q_r' are edge-disjoint. Next, we use these paths to derive the lower bound (4.34) on $C(S)$. To this end we introduce additional notation. First, denote the union of Q_k, Q_l, and all Q_r' by Q. Define $S_Q := S \cap Q$. Because for each non-proper potential terminal in $V(S) \setminus V(S_Q)$ there is one incident edge in $E(S) \setminus E(S_Q)$, and because this mapping can be chosen to be bijective, it holds that

$$c(E(S) \setminus E(S_Q)) - p((V(S) \setminus V(S_Q)) \cap T_p^-) \geqslant 0. \tag{4.36}$$

From the definitions of $\underline{d}_{H^p}$ and r_H^{pc} one infers

$$\begin{aligned} & c(E(S_Q)) + p(T_p^+ \setminus V(S)) - p(V(S_Q) \cap T_p^-) && (4.37)\\ \geqslant & \sum_{q=1}^{s^+-2} r_H^{pc}(t_q^+) + \underline{d}_{H^p}(v_i, \underline{v}_{i,1}^{H^p}) + \underline{d}_{H^p}(v_i, \underline{v}_{i,2}^{H^p}) - p(v_i). && (4.38)\end{aligned}$$

Finally, one obtains:

$$\begin{aligned}
C(S) &= c(E(S)) + p(V \setminus V(S)) \\
&= c(E(S)) + p(T_p^+ \setminus V(S)) + p(T_p^- \setminus V(S)) \\
&= c(E(S)) + p(T_p^+ \setminus V(S)) + p(T_p^-) - p(V(S) \cap T_p^-) \\
&= c(E(S)) + p(T_p^+ \setminus V(S)) + p(T_p^-) \\
&\quad - p(V(S_Q) \cap T_p^-) - p((V(S) \setminus V(S_Q)) \cap T_p^-) \\
&= c(E(S_Q)) + c(E(S) \setminus E(S_Q)) + p(T_p^+ \setminus V(S)) + p(T_p^-) \\
&\quad - p(V(S_Q) \cap T_p^-) - p((V(S) \setminus V(S_Q)) \cap T_p^-) \\
&\overset{(4.36)}{\geqslant} c(E(S_Q)) + p(T_p^+ \setminus V(S)) + p(T_p^-) - p(V(S_Q) \cap T_p^-) \\
&\overset{(4.37)}{\geqslant} \sum_{q=1}^{s^+-2} r_H^{pc}(t_q^+) + \underline{d}_{H^p}(v_i, \underline{v}_{i,1}^{H^p}) + \underline{d}_{H^p}(v_i, \underline{v}_{i,2}^{H^p}) - p(v_i) + p(T_p^-) \\
&= \sum_{q=1}^{s^+-2} r_H^{pc}(t_q^+) + \underline{d}_{H^p}(v_i, \underline{v}_{i,1}^{H^p}) + \underline{d}_{H^p}(v_i, \underline{v}_{i,2}^{H^p}) + p(T_p^- \setminus \{v_i\}).
\end{aligned}$$

The first equality is just the definition of $C(S)$. The second inequality follows from $T_p = T_p^+ \dot\cup T_p^-$. The next three equalities result from splitting up individual sums. The last equality follows from the fact that either $v_i \in T_p^-$ or $p(v_i) = 0$. □

Each vertex $v_i \in V \setminus T_p^+$ with the property that the affiliated lower bound (4.34) exceeds a known upper bound can be eliminated. Moreover, if a solution S corresponding to the upper bound is given and $v_i \notin V(S)$, one can also eliminate v_i if the lower bound (4.34) is equal to $C(S)$. A result similar to Proposition 4.16 can be formulated for edges of an optimal solution.

Another application of the terminal-regions decomposition can be found for PC-STP instances with articulation points. While only a few instances from the literature contain articulation points in their original form, this situation changes once the reduction techniques described so far have been applied. Even more so once branch-and-bound has been initiated, see Section 4.5.1. Recall that throughout this chapter I_{PC} is assumed to be connected.

First, one observes that if a biconnected component $B \subseteq G$ satisfies $T_p^+ \subseteq V(B)$ and no trivial solution exists, then there is at least one optimal solution S with $S \subseteq B$. The opposite case, namely that $T_p^+ \nsubseteq V(B)$, is not as straightforward and requires some groundwork. In the remainder of this section it will be assumed that I_{PC} contains at least one biconnected component that does not contain all proper potential terminals. Let $B \subseteq G$ be a biconnected component and let $R \subseteq G$ be a connected subgraph such that $E(R) \cap E(B) = \emptyset$. A vertex $v \in B$ such that there is a path Q from v to R with $V(Q) \cap V(B) = \{v\}$ will be called *gate vertex* from B to R. One readily acknowledges the following result.

Lemma 4.17. *Let $B \subseteq G$ be a biconnected component and $R \subseteq G$ a nonempty, connected subgraph such that $E(R) \cap E(B) = \emptyset$. There exists exactly one gate vertex from B to R.*

Based on Lemma 4.17 and the terminal-regions decomposition, the following proposition gives an additional criterion to eliminate biconnected components. The proposition can be motivated similarly to Proposition 4.16. This time, however, we compare the lower bound against the maximum profit that can be attained in a biconnected component.

Proposition 4.18. *Let $B \subseteq G$ be a biconnected component with $T_p^+ \nsubseteq V(B)$. Let $R \subseteq G$ be a connected subgraph with $T_p^+ \setminus V(B) \subseteq V(R)$ and assume that $E(R) \cap E(B) = \emptyset$. If $T_p^+ \cap V(B) = \emptyset$, then there is an optimal solution that does not contain any edge of B. Otherwise, let $v_i \in V(B)$ be the gate vertex from B to R, let H be a terminal-regions decomposition of B, and define $\underline{d}_{H^p}$ and $\underline{v}_{i,1}^{H^p}$ with respect to B. Let $X := V(B) \setminus \{v_i\}$ and define*

$$L := \underline{d}_{H^p}(v_i, \underline{v}_{i,1}^{H^p}) + \sum_{t \in X \cap T_p^+} r_H^{pc}(t) - \max_{t \in X \cap T_p^+} r_H^{pc}(t) + \sum_{t \in X \cap T_p^-} p(t). \tag{4.39}$$

If

$$L \geqslant \sum_{v \in X} p(v) \tag{4.40}$$

holds, then for at least one solution S to I_{PC} it holds that either $S \subseteq B$ or $E(S) \cap E(B) = \emptyset$.

Proof. Throughout this proof it will be assumed that no trivial (i.e. single-vertex) optimal solution exists—otherwise, the proof is already complete. Note that this assumption implies that $T_p^+ \neq \emptyset$.

First, assume that $T_p^+ \cap B = \emptyset$. Thus, $T_p^+ \subseteq R$. Recall that because no trivial solution exists, there is at least one optimal solution S whose leaves are a subset of T_p^+. Consequently, S cannot contain any edge of B. Otherwise, from the ends of this edge there would be two disjoint paths to T_p^+, and thus there would be at least two gate vertices from B to R.

Second, assume that $T_p^+ \cap B \neq \emptyset$. Let S be a feasible solution with $S \nsubseteq B$ and $E(S) \cap E(B) \neq \emptyset$. We will show that a feasible solution S' with $C(S') \leqslant C(S)$ exists, such that either $S' \subseteq B$ or $E(S') \cap E(B) = \emptyset$. In this way, the proof is concluded. Assume that all leaves of S are contained in T_p^+, otherwise one can always choose a S of no higher cost that satisfies this property. Because this assumption implies $S \cap R \neq \emptyset$, the gate vertex v_i from B to R is contained in S. Moreover, any path $Q \subseteq S$ starting from v_i satisfies either $E(Q) \cap E(B) = \emptyset$ or $E(Q) \subseteq E(B)$. Otherwise, there would be at least two gate vertices from B to R. Let S_B be the subgraph of S that consists of all paths in S from v_i to vertices in B. The above considerations imply that the subgraph S' obtained by removing S_B from S and adding v_i is connected. Similar to the proof of Proposition 4.16 one can now show that

$$c(E(S_B)) + p(X \setminus V(S_B)) \geqslant L. \tag{4.41}$$

Therefore, it holds that

$$\begin{aligned} C(S) &= c(E(S)) + p(V \setminus V(S)) \\ &= c(E(S')) + p((V \setminus V(S')) \setminus X) + c(E(S_B)) + p(X \setminus V(S_B)) \\ &\overset{(4.41)}{\geqslant} c(E(S')) + p((V \setminus V(S')) \setminus X) + L \\ &\overset{(4.40)}{\geqslant} c(E(S')) + p((V \setminus V(S')) \setminus X) + p(X) \\ &= c(E(S')) + p((V \setminus V(S')) \\ &= C(S'), \end{aligned}$$

which concludes the proof. □

The set R in Proposition 4.18 can for example be computed (or its non-existence can be shown) by a depth-first-search on the graph $(V, E \setminus E(B))$ starting from any $t \in T_p^+ \setminus V(B)$. If (4.40) holds, it might still be the case that $S \subseteq B$ for all optimal solutions S. This case can for example be ruled out if a feasible solution $S' \subsetneq B$ is known that satisfies $C(S') \leqslant \sum_{v \in V \setminus V(B)} p(v)$ (or by more sophisticated criteria involving the terminal-regions decomposition of B). If such a S' is known, one can eliminate all edges of B from I_{PC}.

To efficiently apply the previous two propositions, one would like to maximize the lower bounds (4.34), and (4.39) respectively. Note that if $T_p^- = \emptyset$, then there always exists a terminal-regions decomposition with $H_0 = \emptyset$ that maximizes the lower bound (4.34)—and equivalently, (4.39). Thus, one directly obtains the following result from Proposition 2.21.

Proposition 4.19. *Given a $v_i \in V \setminus T_p^+$, finding a terminal-regions decomposition that maximizes* (4.34) *is $\mathcal{NP}$-hard. The same holds for* (4.39).

Thus, to compute a terminal-regions decomposition a heuristic approach will be used. The heuristic is similar to Dijkstra's algorithm. Put all $t_i^+ \in T_p^+$ in the initial priority queue (with distance value 0). Similar to Algorithm 4.1, we subtract from the distance value of each vertex $v \in V \setminus T_p^+$ its prize $p(v)$ when it is updated. Moreover, the algorithm does not extend a region H_i from a vertex $v \in H_i$ if an upper bound $\overline{b}_i \in \mathbb{Q}_{\geqslant 0}$ on $r_H^{pc}(t_i^+)$ is already known and $\underline{d}_{H^p}(t_i^+, v) \geqslant \overline{b}_i$. Such upper bounds can be computed during the execution of the algorithm. Finally, we apply a simple local heuristic that checks edges between different regions and fully includes them in one of the regions if advantageous.

4.4 Changing the problem class

The previous section discussed several techniques to prove that certain edges or vertices are not contained in an optimal solution. This section uses similar techniques to prove the opposite: That certain vertices are contained in at least one optimal solution. We show that such a *rooting* of a PCSTP instance sets the stage for more efficient algorithms.

4.4.1 Identifying roots

A cornerstone of the approach described in this section is the Steiner arborescence problem (SAP) and the associated *DCut* IP formulation; see Section 1.1.3. As shown in Chapter 3, the dual-ascent algorithm by Wong (1984) for *DCut* can quickly compute empirically strong lower bounds. Moreover, we have seen that the information provided by dual-ascent can be used for the generation of initial cutting planes, for reduction methods, and for primal heuristics. Thus, it seems promising to devise a transformation from PCSTP to SAP, to be able to apply this algorithm. Such a transformation is given in the following. We apply the idea of *cost-shifting* (Duin, 1993; Ljubic et al., 2006), to get rid of non-proper potential terminals (in Step 2). We also note that Ljubic et al. (2006) describe a transformation from PCSTP to an SAP variant with additional constraints.

Transformation 4.20 (PCSTP to SAP)**.**
Input: *PCSTP* (V, E, c, p)
Output: *SAP* (V', A', T', c', r')

1. *Set* $V' := V$, $A' := \{(v,w) \in V' \times V' \mid \{v,w\} \in E\}$, *and* $M := \sum_{t \in T_p^+} p(t)$.
2. *Define* $c' : A' \to \mathbb{Q}_{\geqslant 0}$ *for all* $a = (v,w) \in A'$ *by*
$$c'(a) := \begin{cases} c(\{v,w\}) - p(w) & \text{if } w \in T_p^-, \\ c(\{v,w\}) & \text{otherwise.} \end{cases}$$
3. *Add vertices* r' *and* v'_0 *to* V'.
4. *For each* $i \in \{1, ..., s^+\}$*:*
   (a) *add arc* (r', t_i) *of weight* M *to* A'*;*
   (b) *add node* t'_i *to* V'*;*
   (c) *add arcs* (t_i, v'_0) *and* (t_i, t'_i) *to* A'*, both being of weight* 0*;*
   (d) *add arc* (v'_0, t'_i) *of weight* $p(t_i)$ *to* A'.
5. *Define set of terminals* $T' := \{t'_1, ..., t'_{s^+}\} \cup \{r'\}$.
6. ***Return*** (V', A', T', c', r').

The underlying idea of the transformation is to add a new terminal t'_i for each original potential terminal t_i and provide additional arcs that make it possible to connect t'_i from any original potential terminal t_j with cost $p(t_i)$. See Figure 4.4b for an example. Note that one needs to compare any solution obtained by the above transformation with the best single vertex tree in order to not miss a trivial optimal solution. In the following, we assume for ease of presentation that no such trivial optimal solution exists.

Each optimal solution S' to the SAP obtained from Transformation 4.20 can be transformed to an optimal solution S to the original PCSTP. This mapping can be done similarly to the transformation from MWCSP to SAP in the previous chapter; we give more details in Rehfeldt and Koch (2018a). The following relation holds:

$$c'(A'(S')) - M + p(T_p^-) = C(S).$$

For $I_{PC} = (V, E, c, p)$ one can define the following formulation, which uses the SAP (V', A', T', c', r') obtained from applying Transformation 4.20 on I_{PC}:

Formulation 4.21. *Transformed prize-collecting cut (PrizeCut)*

$$\begin{aligned}
&min \quad {c'}^T x - M + p(T_p^-) && && (4.42)\\
&\qquad x \text{ satisfies } (1.3), (1.4) && && (4.43)\\
&y(\{v,w\}) = x((v,w)) + x((w,v)) && \textit{for all } \{v,w\} \in E && (4.44)\\
&\qquad y(e) \in \{0,1\} && \textit{for all } e \in E. && (4.45)
\end{aligned}$$

The y variables correspond to the solution to I_{PC}; note that removing them does not change the optimal solution value, neither that of the LP relaxation.

Implied potential terminals

To avoid adding an artificial root (which entails *big M* constants and symmetry) in the transformation to SAP, one can attempt to identify vertices that are part of all optimal solutions to the original PCSTP. To this end, consider a PCSTP and let $v, w \in V$. Further, let $W = (v_1, e_1, v_2, e_2, ..., e_{r-1}, v_r)$ with $v_1 = v$ and $v_r = w$ be a prize-constrained (v, w)-walk (as defined in Section 4.3). Define the *left-rooted prize-constrained length* of W as:

$$\vec{l}_{pc}(W) := \max\{c_{pc}(W(v, v_i)) \mid v_i \in V(W) \cap (T_p^+ \cup \{w\})\}. \tag{4.46}$$

Recall that we defined c_{pc} in (4.11). As compared to the prize-constrained length, (4.46) considers only subwalks starting from the first vertex of walk W. Furthermore, define the *left-rooted prize-constrained* (v, w)*-distance* as:

$$\vec{d}_{pc}(v, w) := \min\{\vec{l}_{pc}(W) \mid W \in \mathcal{W}_{pc}(v, w)\}. \tag{4.47}$$

Note that in general $\vec{d}_{pc}$ is not symmetric. Definition (4.47) gives rise to

Theorem 4.22. *Let $v, w \in V$. If*

$$p(v) > \vec{d}_{pc}(v, w) \tag{4.48}$$

is satisfied, then every optimal solution that contains w also contains v.

Proof. Let S be a tree with $w \in V(S)$ and $v \notin V(S)$. Further, let $W = (v_1, e_1, ..., e_{r-1}, v_r)$ be a prize-constrained (v, w)-walk with $\vec{l}_{pc}(W) = \vec{d}_{pc}(v, w)$ and define $a := \min\{k \in \{1, ..., r\} \mid v_k \in V(S)\}$ and $b := \min\{k \in \{a, a+1, ..., r\} \mid v_k \in T_p^+ \cup \{w\}\}$. Note that

$$c_{pc}(W(v, v_a)) \leqslant c_{pc}(W(v, v_b)). \tag{4.49}$$

Add the subgraph corresponding to $W(v, v_a)$ to S, which yields a new connected subgraph S'. If S' is not a tree, make it one by removing redundant edges, without removing any node (which can only decrease $C(S')$). It holds that:

$$\begin{aligned} C(S') &\leqslant C(S) + c_{pc}(W(v, v_a)) - p(v) \\ &\overset{(4.49)}{\leqslant} C(S) + c_{pc}(W(v, v_b)) - p(v) \\ &\leqslant C(S) + \vec{l}_{pc}(W) - p(v) \\ &= C(S) + \vec{d}_{pc}(v, w) - p(v) \\ &\overset{(4.48)}{<} C(S). \end{aligned}$$

The relation $C(S') < C(S)$ implicates that any optimal solution that contains w also contains v. $\square$

Corollary 4.23. *Let $v, w \in V$. If*

$$p(v) \geqslant \vec{d}_{pc}(v, w) \tag{4.50}$$

is satisfied and w is contained in an optimal solution, then v is also part of an optimal solution.

The left-rooted prize-constrained distance can be exemplified by means of Figure 4.3. It holds that $\vec{d}_{pc}(v_0, v_5) = 2$, but $\vec{d}_{pc}(v_5, v_0) = 3$. A walk corresponding to $\vec{d}_{pc}(v_0, v_5)$ is $(v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, v_2, \{v_2, v_1\}, v_1, \{v_1, v_3\}, v_3, \{v_3, v_4\}, v_4, \{v_4, v_5\}, v_5)$. Corollary 4.23 implies that if v_5 is part of an optimal solution, then there is an optimal solution that contains v_0. The converse does not necessarily hold. Indeed, v_5 is not part of any optimal solution even though v_0 is (together with v_2 and v_3).

As for d_{pc}, computing $\vec{d}_{pc}$ is $\mathcal{NP}$-hard (which can be shown analogously). However, one can devise a simple algorithm for finding upper bounds, which is very similar to Algorithm 4.1. Let $t_0 \in T_p^+$. The subsequently sketched algorithm provides a set of vertices $\bar{T}_{t_0}$ such that $\vec{d}_{pc}(t_0, v) < p(t_0)$ for all $v \in \bar{T}_{t_0}$. Initialize $dist_{pcr}[v] := \infty$ for all $v \in V \setminus \{t_0\}$, and set $dist_{pcr}[t_0] := 0$. Start Dijkstra's algorithm with only t_0 in the priority queue, but apply the following modifications: First, update vertex w from vertex u if and only if both

$$dist_{pcr}[u] + c(\{u, w\}) < p(t_0) \tag{4.51}$$

and

$$dist_{pcr}[w] > dist_{pcr}[u] + c(\{u, w\}) - p(w). \tag{4.52}$$

In this case set $dist_{pcr}[w] := dist_{pcr}[u] + c(\{u, w\}) - p(w)$. No $t \in T_p$ is allowed to be reinserted into the priority queue after it has been removed. Finally, define $\bar{T}_{t_0} := \{u \in V \mid dist_{pcr}[u] < p(t_0)\}$. Note that $t_0 \in \bar{T}_{t_0}$. This algorithm is basically the same as Algorithm 4.1, except for using $p(v)$ instead of c_0 and using a slightly different update scheme.

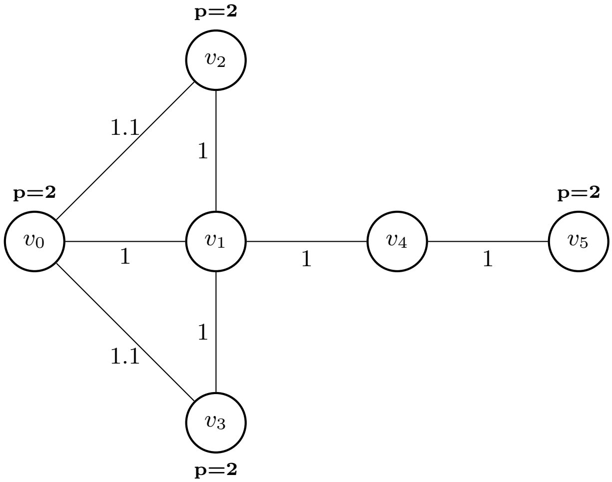


Figure 4.3: PCSTP instance. The prizes of the individual vertices are specified by p; only non-zero prizes are shown.

Combining implications and reduced-costs

By using LP information, the above algorithm can be combined with Transformation 4.20 to obtain a criterion for potential terminals to be part of all optimal solutions. First, note that if a separation algorithm or dual-ascent is applied, one obtains reduced-costs for an LP relaxation of *DCut* that contains only a subset of constraints (1.3). Second, observe that given an SAP I' obtained from I_{PC} with corresponding optimal solutions S' and S, for $t_i \in T_p$ it holds that $t_i \in V(S)$ if and only if $(v'_0, t'_i) \notin A'(S')$. As a consequence one obtains

Proposition 4.24. *Consider (V', A', T', c', r') obtained by applying Transformation 4.20 on I_{PC}. Let $\tilde{\mathcal{U}} \subseteq \{U \subset V' \mid r' \notin U, U \cap T' \neq \emptyset\}$ and let $\tilde{L}$ be the objective value and $\tilde{c}$ the reduced-costs of an optimal solution to the LP:*

$$
\begin{aligned}
min \quad & {c'}^T x - M + p(T_p^-) && && (4.53)\\
& x(\delta^-(U)) \geqslant 1 && \text{for all } U \in \tilde{\mathcal{U}}, && (4.54)\\
& x(a) \in [0,1] && \text{for all } a \in A'. && (4.55)
\end{aligned}
$$

Moreover, let K be an upper bound on the cost of an optimal solution to I_{PC}. Finally, let $t_i \in T_p^+$ and let $\bar{T}_i \subseteq T_p^+$ such that $V(S) \cap \bar{T}_i \neq \emptyset \Rightarrow t_i \in V(S)$ for each optimal solution S to I_{PC}. If

$$\sum_{j \mid t_j \in \bar{T}_i} \tilde{c}((v'_0, t'_j)) + \tilde{L} > K \tag{4.56}$$

holds, then t_i is part of all optimal solutions to I_{PC}.

If a $t_i \in T_p^+$ has been shown to be part of all optimal solutions, by building $\bar{T}_i$ with Theorem 4.22 and using (4.56), Theorem 4.22 can again be applied—to directly identify further $t_j \in T_p$ that are part of all optimal solutions by using the condition $p(t_j) > \vec{d}_{pc}(t_j, t_i)$. Identifying such *fixed* terminals can considerably improve the strength of the techniques described in Section 4.3, which usually leads to further graph reductions and the fixing of additional terminals.

4.4.2 Rooting the problem: RPCSTP and SPG

Once at least one vertex has been shown to be part of at least one optimal solution, the PCSTP can be reduced to a RPCSTP. Recall that we assume $p(t) = 0$ for all $t \in T_f$. We introduce the following simple transformation.

Transformation 4.25 (RPCSTP to SAP)**.**
Input: *RPCSTP* (V, E, T_f, c, p) *and* $t_p, t_q \in T_f$
Output: *SAP* (V', A', T', c', r')

1. *Set* $V' := V$, $A' := \{(v, w) \in V' \times V' \mid \{v, w\} \in E\}$, $r' := t_q$.
2. *Define* $c' : A' \to \mathbb{Q}_{\geqslant 0}$ *for all* $a = (v, w) \in A'$ *by*

$$c'(a) = \begin{cases} c(\{v, w\}) - p(w) & \textit{if } w \in T_p^-, \\ c(\{v, w\}) & \textit{otherwise.} \end{cases}$$

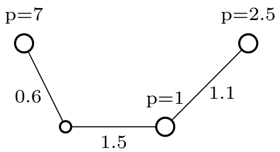
3. *For each* $i \in \{1, ..., s^+\}$*:*
   (a) *add node* t'_i *to* V'*,*
   (b) *add arc* (t_i, t'_i) *of weight* 0 *to* A'*,*
   (c) *add arc* (t_p, t'_i) *of weight* $p(t_i)$ *to* A'*.*
4. *Define set of terminals* $T' := \{t'_1, ..., t'_{s^+}\} \cup T_f$*.*
5. ***Return*** (V', A', T', c', r')*.*

A comparison of Transformation 4.20 and Transformation 4.25 is illustrated in Figure 4.4.

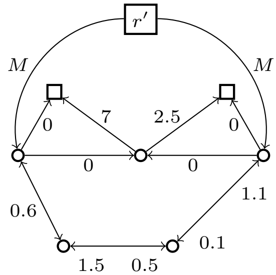
For an RPCSTP (V, E, T_f, c, p) we define the *transformed rooted prize-collecting cut* (*PrizeRCut*) formulation, similar to *PrizeCut*, based on the SAP instance (V', A', T', c', r') obtained from Transformation 4.25:

Formulation 4.26. *Transformed rooted prize-collecting cut (PrizeRCut)*

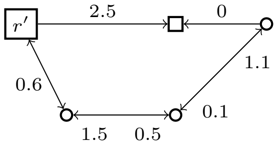
$$\min\{{c'}^T x + p(T_p^-) \mid x \textit{ satisfies } (1.3), (1.4),\ (x, y) \textit{ satisfies } (4.44),\ y \textit{ satisfies } (4.45)\}. \tag{4.57}$$



(a) PCSTP instance



(b) SAP instance obtained from Transformation 4.20



(c) SAP instance obtained from Transformation 4.25

Figure 4.4: Illustration of a PCSTP instance (left) and the equivalent SAP obtained by Transformation 4.20 (middle). Given the information that the potential terminal with weight $p = 7$ is part of at least one optimal solution, Transformation 4.25 yields the SAP depicted on the right. The terminals of the SAPs are drawn as squares and the (two) potential terminals for the PCSTP are enlarged.

By $PrizeRCut(I_{RPC}, t_p, t_q)$ we denote the $PrizeRCut$ formulation for an RPC-STP I_{RPC} when using (fixed) terminals t_p, t_q in Transformation 4.25. One might wonder whether the choice of t_p and t_q in Transformation 4.25 can affect the value of the LP relaxation $v_{LP}(PrizeRCut(I_{RPC}, t_p, t_q))$. However, this value does not change, and even more:

Theorem 4.27. *Let I_{RPC} be an RPCSTP and let $t_p, t_q, t_{\tilde{p}}, t_{\tilde{q}}$ be any of its fixed terminals. Define $R(t_i, t_j) := \mathcal{P}_{LP}(PrizeRCut(I_{RPC}, t_i, t_j))$. It holds that:*

$$proj_y(R(t_p, t_q)) = proj_y(R(t_{\tilde{p}}, t_{\tilde{q}})). \tag{4.58}$$

Proof. Let (V, E, T_f, c, p) be the RPCSTP I_{RPC} and denote the SAP resulting from applying Transformation 4.25 on (I_{RPC}, t_p, t_q) by (V', A', T', c', t_q). Set $D = (V', A')$. Furthermore, let (x, y) be a feasible solution to the LP relaxation of $PrizeRCut(I_{RPC}, t_p, t_q)$—so $(x, y) \in R(t_p, t_q)$. For ease of presentation, we will use the notation x_{ij} instead of $x((v_i, v_j))$ for an arc (v_i, v_j). The theorem will be proved in two steps: first by fixing t_q and changing t_p, and second by fixing t_p and changing t_q. Note that due to symmetry reasons in both cases it is sufficient to show that one projection is contained in the other.

1) $proj_y(R(t_p, t_q)) = proj_y(R(t_{\tilde{p}}, t_q))$ Let $\tilde{I}_{\tilde{p}} = (\tilde{V}, \tilde{A}, \tilde{T}, \tilde{c}, t_q)$ be the SAP resulting from applying Transformation 4.25 on $(I_{RPC}, t_{\tilde{p}}, t_q)$, and set $\tilde{D} := (\tilde{V}, \tilde{A})$; note that $\tilde{V} = V'$ and $\tilde{T} = T'$. Define $\tilde{x} \in [0, 1]^{\tilde{A}}$ by $\tilde{x}((t_{\tilde{p}}, t'_i)) := x((t_p, t'_i))$ for $i = 1, ..., z$ (with the notation from Transformation 4.25) and by $\tilde{x}_{ij} := x_{ij}$ for all remaining arcs. Suppose that there is a $U \subset \tilde{V}$ with $t_q \notin U$ and $U \cap \tilde{T} \neq \emptyset$ such that $\tilde{x}(\delta^-_{\tilde{D}}(U)) < 1$. From $x(\delta^-_D(U)) \geqslant 1$ and the construction of $\tilde{x}$ it follows that $t_{\tilde{p}} \in U$—otherwise

$\tilde{x}(\delta^-_{\tilde{D}}(U)) \geqslant x(\delta^-_{D}(U))$. For $U^z := U \setminus \{t'_1, ..., t'_z\}$ one obtains

$$x(\delta^-_{D}(U^z)) = \tilde{x}(\delta^-_{\tilde{D}}(U^z)) \leqslant \tilde{x}(\delta^-_{\tilde{D}}(U)) < 1. \tag{4.59}$$

Because of $t_q \notin U^z$ and $U^z \cap \tilde{T} \supseteq \{t_{\tilde{p}}\} \neq \emptyset$, one obtains a contradiction from (4.59). Therefore, $\tilde{x}$ satisfies (1.3) for the SAP $\tilde{I}_{\tilde{p}}$. Furthermore, $\tilde{y}$ defined by $\tilde{y}(\{v_i, v_j\}) := \tilde{x}_{ij} + \tilde{x}_{ji}$ for all $\{v_i, v_j\} \in E$ satisfies $\tilde{y} = y$.

2) $proj_y(R(t_p, t_q)) = proj_y(R(t_p, t_{\tilde{q}}))$ Define the SAP $\tilde{I}_{\tilde{q}} := (V', A', T', c', t_{\tilde{q}})$ (the result of transforming $(I_{RPC}, t_p, t_{\tilde{q}})$). As there is only one underlying directed graph (namely D), in the following we write δ^- instead of δ^-_D. Let f be a 1-unit flow from t_q to $t_{\tilde{q}}$ such that $f_{ij} \leqslant x_{ij}$ for all $(v_i, v_j) \in A'$. Define $\tilde{x}$ by $\tilde{x}_{ij} := x_{ij} + f_{ji} - f_{ij}$ for all $(v_i, v_j) \in A'$. Let $U \subset V'$ such that $t_{\tilde{q}} \notin U$ and $U \cap T' \neq \emptyset$. If $t_q \notin U$, then $f(\delta^-(U)) = f(\delta^+(U))$ and so $\tilde{x}(\delta^-(U)) = x(\delta^-(U)) \geqslant 1$. On the other hand, if $t_q \in U$, then $f(\delta^+(U)) = f(\delta^-(U)) + 1$, so

$$\tilde{x}(\delta^-(U)) \geqslant x(\delta^-(U)) + 1 \geqslant 1. \tag{4.60}$$

Consequently, $\tilde{x}$ satisfies (1.3) for the SAP $\tilde{I}_{\tilde{q}}$. From $x_{ij} + x_{ji} \leqslant 1$ for all $(v_i, v_j) \in A'$, it follows that $\tilde{x} \in [0,1]^{A'}$, and for the corresponding $\tilde{y}$ one verifies $\tilde{y} = y$. □

Consequently, if only the y variables are of interest, we write $PrizeRCut(I_{RPC})$ instead of $PrizeRCut(I_{RPC}, t_p, t_q)$. For the (heuristic) dual-ascent algorithm the choice of t_p and t_q still matters, as it can change both lower bound and reduced-costs, see Section 4.5.1.

Theorem 4.27 has also consequences for two well-known IP formulations for SPG and NWSTP. Recall that for SPG a widely used formulation is the bidirected cut formulation (Wong, 1984): Replace each edge by two anti-parallel arcs of the same weight as the original edge, and consider the resulting problem as an SAP with an arbitrary terminal as the root. Solve this SAP by Formulation 1.1. As a corollary of Theorem 4.27 one easily obtains a result that was proved by Goemans and Myung (1993) in a more involved way.

Corollary 4.28. *The optimal LP value of the bidirected cut formulation for SPG is independent of the choice of the root.*

For NWSTP a similar formulation has proven effective in practice (Gamrath et al., 2017; Leitner et al., 2018a): Transform the problem to SAP in the same way as for SPG. Additionally, add the weight of each vertex to all its incoming arcs. Note that when transforming an NWSTP to RPCSTP as described in Section 4.2.2, and applying Transformation 4.25 to this RPCSTP, one obtains the same SAP as with the procedure described above (given the same choice of the root). Thus, Theorem 4.27 directly yields the following, new, result.

Corollary 4.29. *The optimal LP value of the bidirected cut formulation for NWSTP is independent of the choice of the root.*

From the definitions of Transformation 4.20 and 4.25 one can acknowledge that switching from $PrizeCut$ to $PrizeRCut$ (if possible) does not deteriorate (and can improve) the tightness of the LP relaxation; due to its importance we formally state this observation in the following proposition; a proof is given in Appendix A.3.4.

Proposition 4.30. *For $I_{PC} = (V, E, c, p)$ let $T_0 \subseteq T_p$ such that $T_0 \subseteq V(S)$ for at least one optimal solution S to I_{PC}. Let $I_{T_0} := (V, E, T_0, c, p)$ be an $RPCSTP$. With $R_{T_0} := \mathcal{P}_{LP}(PrizeRCut(I_{T_0}))$, $R := \mathcal{P}_{LP}(PrizeCut(I_{PC}))$ it holds that*

$$proj_y(R_{T_0}) \subseteq proj_y(R). \tag{4.61}$$

Moreover, the inequality

$$v_{LP}(PrizeCut(I_{PC})) \leqslant v_{LP}(PrizeRCut(I_{T_0})) \tag{4.62}$$

holds and can be strict.

Finally, by combining the reductions to RPCSTP and SAP with the reductions techniques described in Section 4.3, it is sometimes possible to either eliminate or fix each potential terminal. Hence the instance becomes an SPG, which allows us to apply the more advanced SPG solution techniques introduced in Chapter 2.

4.5 Solving to optimality

In the following, the integration of the individual PCSTP techniques within an exact solving approach will be described. Furthermore, the performance of the resulting solver will be discussed. As before, the implementations are realized within the branch-and-cut solver SCIP-Jack.

4.5.1 Interleaving the components within branch-and-cut

This section demonstrates the broad applicability of the PCSTP algorithms and techniques introduced so far in a branch-and-cut framework. It also aims to highlight the strong interrelation between the individual techniques.

As to IP formulations, for RPCSTP instances (including instances transformed to RPCSTP) we use $PrizeRCut$. However, if the given instance could not be transformed to RPCSTP during presolving, we use an IP formulation similar to the ESA^+ formulation from Chapter 3.2.3—which requires fewer variables than $PrizeCut$. Still, $PrizeCut$ is being used for dual-ascent, and as such is a vital component within the branch-and-cut framework.

Presolving We perform presolving in several rounds—as long as a predefined percentage of edges has been eliminated during the previous round. All PCSTP reduction techniques described in this chapter are applied. We also employ a (significantly restricted) adaptation of the extend reduction framework from Chapter 2.4. Because we only use standard distances, these extended reduction techniques have a far smaller impact than their SPG counterparts. During presolving we also try to transform any PCSTP instance to RPCSTP or even SPG.

Domain propagation During the separation phase, SCIP-Jack uses the reduced-costs provided by the LP solver and the best known upper bound to perform variable fixings, see Chapter 2.6. These variable fixings can often be translated into the deletion or contraction of edges, and thus can allow for further PCSTP reductions. Therefore, we also re-employ the reduction techniques for domain propagation (once a predefined number of edges has been deleted by the reduced-cost criterion). Subsequently, we translate the deletion of edges and the fixing of potential terminals into variable fixings in the IP.

Dual heuristics Recall that the dual-ascent heuristic by Wong (1984) provides a dual bound as well as reduced-costs. Based on our new graph transformations, we use this dual heuristic in presolving (for reduced-cost based reduction tests), for primal heuristics (to find a subgraph that contains a good feasible solution), and for computing initial cuts. Whenever a problem has been transformed to RPCSTP, we perform the dual-ascent heuristic on several SAPs resulting from different choices of t_p and t_q, which usually changes the lower bound (and the reduced-costs) provided by the heuristic.

Primal heuristics As to primal heuristics, we use adaptations of the MWCSP primal heuristics described in Chapter 3.5.1. Several of these heuristics compute solutions on newly built subgraphs (e.g. by merging feasible solutions). For such heuristics we also employ the PCSTP reductions from this chapter. Additionally, we use an adapted version of the SPG local-search heuristics from Uchoa and Werneck (2010), shortly described in Section 2.5. For the PCSTP we consider all proper potential terminals as key-vertices, and furthermore use cost-shifting (see Section 4.4.1) to take the prizes of non-proper potential terminals into account.

LP and cutting-planes Also the LP kernel interacts with the remaining components: By means of the prize-constrained distances and upper bounds provided by the heuristics it is usually possible to switch to the $PrizeRCut$ formulation. In turn, the reduced-costs and lower bound provided by an improved LP solution can be used to reduce the problem size—which can even enable further prize-constrained walk based reductions. As in the previous chapters, we use a specialized maximum-flow algorithm for the separation of the directed cut constraints (1.3). Additionally, we separate constraints for $TransRCut$ of the form

$$x(\delta^-(v)) + x((t_p, t_i')) \leqslant 1 \qquad t_i \in T_p^+ \setminus T_f, v \in \{u \in V \mid \vec{d}_{pc}(t_i, u) < p(t_i)\},$$

with t_p and t_i' as defined in Transformation 4.25. The constraints represent the implication that $v \in V(S) \Rightarrow t \in V(S)$ for any optimal solution S if $\vec{d}_{pc}(t, v) < p(t)$. Corresponding constraints are separated for $TransCut$.

Restart In the course of the solution process one can regularly either delete or fix each potential terminal—through the combination of presolving, primal heuristics, the left-rooted prize-constrained distance, and graph transformation and LP methods. In such a case one might restart the solution process and use the SPG solver described in Chapter 2. If a PCSTP instance is transformed to SPG at the root node of the branch-and-bound tree, we run aggressive SPG presolving and translate it into variable fixings in the IP formulation. In the remainder of the solution process SPG specific primal heuristics and reduction techniques are used, but the remaining algorithmic components are left unchanged. If all potential terminals are fixed already during presolving, a full restart is initiated, including full SPG-specific presolving, and the instance is handled entirely as an SPG.

Branching Just as for SPG, branching is performed on vertices. In the case of PCSTP or RPCSTP, we make the vertex to branch on a fixed terminal in one branch-and-bound child node, and remove it in the sibling node. The implications from the left-rooted prize-constrained distance often set the state for further graph changes, resulting in (local) variable fixings.

4.5.2 Computational results

For details on the hardware used in this thesis see Chapter 1.2.1. To the best of our knowledge, the two other fastest PCSTP solvers are *mozartballs* from Fischetti et al. (2017) (the winner of the exact PCSTP categories at the 11th DIMACS Challenge), and *dapcstp* (Leitner et al., 2018a). While no solver dominates on all benchmarks, the branch-and-bound based *dapcstp* is very competitive, and on several test-sets orders of magnitude faster than mozartballs—it is even faster than state-of-the-art heuristic methods, e.g. from Fu and Hao (2017a). Thus, *dapcstp*, which is publicly available[19], will in the following be used for comparison. Only single-thread mode is used (also because *dapcstp* does not support multiple threads).

For the following experiments 12 well-known benchmark test-sets are used, as detailed in Table 4.1. ACTMOD and HIV contain originally MWCSP and NWSTP instances, which have been transformed to PCSTP by using the methods described in Chapter 4.2.2.

Presolving results

We restrict the analysis of the impact of the individual algorithmic components to presolving. The reason for this decision is the strong interaction of the individual components, which makes the individual impact difficult to measure. For example, deactivating the primal heuristics also has a large effect on the reduction methods, since heuristics are heavily used for the bound-based reductions. Vice-versa, reduction techniques are a central ingredient of several primal heuristics. The reader nevertheless interested in such results is referred to Rehfeldt and Koch (2020).

Presolving can, arguably, be considered the most independent component within our branch-and-cut algorithm, not least because it already solves most instances to optimality. Recall that during presolving we not only make use reduction methods, but also of primal and dual heuristics, as well as of the new graph transformations

[19] `https://github.com/mluipersbeck/dapcstp`

Name	#	$\|V\|$	$\|E\|$	Status	Description
JMP	34	100 - 400	315 - 1576	solved	Sparse instances of varying structure, introduced in Johnson et al. (2000).
Cologne	29	741 - 1810	6332 - 16794	solved	Instances from fiber optic network design for German cities (Ljubic, 2004).
CRR	80	500 - 1000	625 - 25000	solved	Mostly sparse instances, based on test-sets from SteinLib (Ljubic, 2004).
ACTMOD	8	2034 - 5226	3335 - 93394	solved	Instances from integrative biological network analysis (Dittrich et al., 2008).
RANDOM	68	200 - 14000	1575 - 112369	solved	Randomly generated instances published in Biazzo et al. (2012).
E	40	2500	3125 - 62500	solved	Mostly sparse instances originally for SPG, introduced in Ljubic (2004).
HANDS	20	39600 - 42500	78704 - 84475	solved	Images of hand-written text from signal processing (DIMACS, 2015).
HANDB	28	158400 - 169800	315808 - 338551	unsolved	
PUCNU	18	64 - 4096	192 - 28512	unsolved	Instances derived from PUC test-set, introduced at 11th DIMACS Challenge.
H2	14	64 - 4096	192 - 24576	unsolved	Hard instances based on hypercubes, introduced at 11th DIMACS Challenge.
HIV	2	386 - 205717	1477 - 2466001	unsolved	HIV mutation networks, introduced at 11th DIMACS Challenge.
MA	20	1000000	10000000	unsolved	Random instances with $T_p = V$, introduced by Sun et al. (2019).

Table 4.1: Details on PCSTP tests sets.

(for applying dual-ascent). Table 4.2 shows the arithmetic mean of the percentage of vertices and edges in the presolved problems. Further, we report the shifted geometric mean of the run-time needed per test-set, with shift $s = 1$. It can be seen that the size of most instances is drastically reduced. Apart from PUCNU and H2, the average size of both the number of vertices and edges is reduced by more than 95 percent on all test-sets. PUCNU and H2 were indeed designed to defy reduction techniques. Against this backdrop, especially on PUCNU the performance of the reduction algorithms is still notable.

While we do not provide detailed results, we note that the new PCSTP presolving techniques are significantly stronger than previous results such as for example Ljubic et al. (2006); Uchoa (2006). The otherwise empirically strongest reduction techniques have been recently introduced in Leitner et al. (2018a) as part of the *dapcstp* solver which we discuss below. Especially the new prize-constrained distance based techniques developed in this article have in several cases a drastic impact. One example is the hardest instance from the E set, *E18-B*. When exchanging just the prize-constrained edge elimination method by that described in Uchoa (2006), the number of remaining edges roughly doubles (to around 11 thousand). Similarly, the size of several of the hardest MA instances is almost halved when using the prize-constrained distance methods instead of their predecessors.

Branch-and-cut results, and comparisons

This subsection provides computational results of the entire branch-and-cut framework developed for this chapter. Table 4.3 provides aggregated results of the experiments with a time limit of two hours for test-sets that contain only instances

Test-set	average reduced problem size vertices[%]	edges[%]	mean reduction time [s]
JMP	0.6	0.0	0.0
Cologne	0.0	0.0	0.0
ACTMOD	0.4	0.1	0.2
CRR	1.5	0.1	0.0
HANDS	0.0	0.0	0.4
RANDOM	1.2	0.3	0.4
E	4.5	0.5	0.3
HANDB	3.0	3.0	2.9
PUCNU	72.2	62.3	1.7
H2	91.2	89.9	2.4
HIV	0.0	0.0	29.6
MA	3.3	3.3	1456.3

Table 4.2: Average problem sizes after application of reduction algorithms.

with less than a million edges, and a time limit of 24 hours for the remaining (two) sets. We have excluded the well-known benchmark sets *Cologne* (Ljubic, 2004) and *JMP* (Johnson et al., 2000), because all contained instances can be solved in less than a second by both *dapcstp* and SCIP-Jack. We note, however, that SCIP-Jack is faster on both benchmarks sets—with respect to the mean as well as to the maximum time.

The second column of Table 4.3 shows the number of instances in the test-set. Columns three and four show the number of solved instance by *dapcstp*, and SCIP-Jack, respectively. The next two columns show the shifted geometric mean (see Section 1.2.2) with shift $s = 1$ of the run-time taken by the respective solvers: First, *dapcstp*, second, SCIP-Jack. The next column shows the speed-up obtained by SCIP-Jack. The next two columns provide the maximum run-time, the last column the speed-up of SCIP-Jack with respect to the maximum time.

		# solved		mean time (sh. geo. mean)			maximum time		
Test-set	#	dapcstp	S.-J.	dapcstp [s]	S.-J. [s]	speedup	dapcstp [s]	S.-J. [s]	speedup
CRR	80	80	80	0.2	0.0	–	4.9	0.7	**7.0**
ACTMOD	8	8	8	0.8	0.2	**4.0**	3.3	0.8	**4.1**
RANDOM	68	68	68	0.8	0.4	**2.0**	78.6	16.4	**4.8**
HANDS	20	20	20	2.5	0.4	**6.2**	54.1	1.5	**36.1**
E	40	37	**40**	2.0	0.4	**5.0**	>7200	22.5	>**320.0**
HANDB	28	25	**26**	32.9	5.0	**6.6**	>7200	>7200	1.0
PUCNU	18	7	**13**	441.7	67.6	**6.5**	>7200	>7200	1.0
H2	14	5	5	525.5	897.6	0.6	>7200	>7200	1.0
HIV	2	1	**2**	293.0	29.8	**9.8**	>86400	950.7	>**90.9**
MA	20	0	**16**	86400	9296.0	**9.3**	>86400	>86400	1.0

Table 4.3: Computational comparison of the solvers *dapcstp* (Leitner et al., 2018a) and SCIP-Jack.

SCIP-Jack is on all but one test-sets faster than *dapcstp*. Furthermore, it solves 28 more instances than *dapcstp* to optimality. The first three test-sets can be solved within seconds by both solvers, with *dapcstp* being significantly slower than SCIP-Jack both for the mean and maximum time. On the next two sets, HANDS and RANDOM, SCIP-Jack is again faster, especially with respect to the maximum time—being up to 33 times faster. For the next four test-sets, the new solver again consistently dominates, with the exception of test-set H2, where *dapcstp* is faster with respect to the shifted geometric mean. A striking example is the PUCNU test-set, where the new solver can solve almost twice as many instances.

On HIV, the new solver is able so solve the *hiv-1* instance, which contains more than two million edges, to optimality in roughly 15 minutes. The best previously known result from the literature was achieved in a 72 hours run on a large-memory machine, see Gamrath et al. (2017). For the MA instances, with 10 million edges each, *dapcstp* fails to solve any instance. In contrast, the new solver can solve all but four of them, some even in less than one hour. To the best of the authors' knowledge, these are by far the largest PCSTP instances that have been solved to optimality in the literature to date.

Finally, we remark that the heavy machinery used in this chapter can sometimes be a disadvantage. Much effort is spent at the root node, and also within each branch-and-bound node many cutting rounds and aggressive propagation are applied. In contrast, the more light-weight, *dapcstp* searches the branch-and-bound tree far more aggressively. Still, dual-ascent based bounds (heavily utilized by *dapcstp*) are often remarkably tight. On the highly symmetric H2 instances, which are unfavorable for LP based algorithms, *dapcstp* is thus competitive with our solver.

Newly solved instances

Finally, we report results on previously unsolved instances from the 11th DIMACS Challenge. The results were obtained with a time limit of 24 hours. We used two different random seeds, which gave slightly better results for three instances; only the best bounds are reported here. All improved instances are listed in Table 4.4, with the first column giving the name of the instance, the second its primal-dual gap, the third the improved found bound, and the fourth the previously best known one. The previously best known solutions are from DIMACS (2015); Braunstein and Muntoni (2016); Fischetti et al. (2017); Fu and Hao (2017a); Gamrath et al. (2017); Leitner et al. (2018a), respectively. We note that the *cc* and *handb* instances, which have unit edge weights, can be transformed to MWCSP. The new results achieved for the MWCSP versions of these instances in the previous chapter are not considered here.

Five DIMACS instances can be solved for the first time to optimality, three of them, *hiv-1*, *cc10-2nu*, and *hc9u2*, within the standard time limit of two hours. Furthermore, the new solver improves the best known upper bounds for another 11 instances, which comprises almost half of the still unsolved PCSTP instances from the 11th DIMACS Challenge.

4.6 Conclusion

This chapter has introduced a number of techniques and algorithms that aim at faster optimal solution of PCSTP. Based on the newly shown fixed-parameter tractability of

Name	gap [%]	new UB	previous UB
hiv-1	**opt**	**656955.33150**	656970.94
handbd04	**opt**	**3202.18574**	3202.710021
cc7-3nu	**opt**	**270**	271
cc10-2nu	**opt**	**167**	168
hc9u2	**opt**	**190**	190
handbd13	0.0	13.18549	13.19699
handbi13	0.1	4.24964	4.251
cc11-2nu	0.8	303	304
cc12-2nu	0.7	563	565
hc8p	1.2	15204	15206
hc9p	1.1	3015	3043
hc9p2	1.4	30228	30242
hc10p	1.4	59778	59866
hc10p2	1.4	59752	59930
hc11p	1.6	118729	119191
hc11p2	1.7	118869	119236

Table 4.4: Improvements on unsolved DIMACS instances.

PCSTP with respect to the number of proper potential terminals, a key element has been the distinction of these vertices within most new algorithms. As an interesting byproduct, we have also demonstrated that any PCSTP can be transformed to an SPG by adding $|T_p|+1$ terminals. Besides the theoretical analyses of the new methods, a central result of this chapter is the integration of the various methods into an exact branch-and-cut framework. The resulting solver significantly pushes the boundaries of computational tractability for the PCSTP, being able to solve instances with up to 10 million edges—over 30 times larger than any PCSTP instance solved in the literature so far.

A computationally promising route for further research would be to design and implement a PCSTP version of the SPG extended reduction paradigm described in Chapter 2. Also, implementing the new FPT dynamic programming algorithm together with pruning rules (similarly to the FPT algorithm in Chapter 2) could significantly accelerate the solution of PCSTP instances with few proper potential terminals. Additionally, further improving the LP relaxation seems to hold a high potential, both from a computational and theoretical point of view.

Chapter 5

Further related problems

The algorithmic parts of the previous three chapters were dominated by highly intricate techniques tailored to individual, albeit certainly related, problems. A central aim was to demonstrate how far the boundaries of computational tractability for each of these problems can be pushed. While there are many commonalities between the three solution approaches, achieving this aim required many problem-specific techniques.

This chapter moves into a different direction. It shows how the algorithmic framework established so far can be used to efficiently solve a considerable number of further related problem with little algorithmic and implementation effort. In this way, this chapter demonstrates the versatility of the algorithmic framework described so far, and its applicability beyond individual problems. This chapter also moves away from the more theoretical aspects prominently featured in the previous chapters. Instead, the focus will be on the extension of previously introduced algorithms, and on computational results.

All problem classes covered in this chapter are solved within the exact branch-and-cut framework developed so far. For all problem classes the following algorithmic components are used. First, by means of transformations, we use some variant of Formulation 1.1, usually strengthened by the flow-balance constraints (2.9) and by the constraints (2.81). We use dual-ascent for computing the initial cuts, and use the maximum-flow algorithm described in Section 6.2.4 for further separation. Additionally, we always apply reduced-costs based domain propagation. As to primal heuristics, we use a simplified version of the recombine-and-reduce heuristic introduced in Section 2.5.2, and some (problem-specific) variant of the shortest-path heuristic introduced in Section 2.5.1. For averaging the run-times and the numbers of branch-and-bound nodes in this chapter, we use the shifted geometric mean with shift $s = 1$. Furthermore, we set a time limit of two hours for all runs.

5.1 The partial and full terminal Steiner tree problems

The *partial terminal Steiner tree problem* (PTSTP) is a generalization of the SPG. Let $G = (V, E)$ be an undirected, connected graph with costs $c : E \to \mathbb{Q}_{\geqslant 0}$ and a set $T \subseteq V$ of *terminals*. Further, let $T_L \subseteq T$ be the set of *partial terminals*. A *partial terminal Steiner tree* is a tree $S \subseteq G$ with $T \subseteq V(S)$ such that all vertices in T_L are leaves of S. The PTSTP asks for a partial Steiner tree S such that $c(E(S))$ is minimized. Note that unlike normal Steiner trees, no partial Steiner tree might exist for a given instance. The PTSTP is for example discussed in Chen (2016) or Hsieh and Gao (2007), both of which include complexity and approximation results. The special case $T_L = T$ is known as the *full Steiner tree problem* (FSTP), see e.g. Lu et al. (2003). FSTP is an important subproblem for both theoretical and practical results on SPG and geometric Steiner tree problems, see also Section 5.4.

A generalization of PTSTP is the *node-weighted partial terminal Steiner tree problem* (NWPTSTP), which additionally provides vertex weights $p : V \to \mathbb{Q}_{\geqslant 0}$. The NWPTSTP asks for a partial Steiner tree S such that $c(E(S)) + p(V(S))$ is minimized. Applications of the NWPTSTP can for example be found in network design, see Sun et al. (2020).

Note that any SPG can be (linearly) reduced to both PTSTP (by setting $T_L := \emptyset$) and FSTP (by adding $|T|$ additional edges and nodes), which shows the latter two problems are $\mathcal{NP}$-hard as well. Also, many results for exact PTSTP solution are the same as for SPG, as can be seen by the simple new transformation described below.

Algorithms

PTSTP (and thus also FSTP) can be readily reduced to SPG as follows. We assume $|T| > 2$, otherwise the problem can be solved easily. First, remove all edges $\{v, w\}$ with $v, w \in T_L$. Second, define $M := c(E)$, and add M to the weight of all edges incident to a partial terminal.

NWPTSTP can be transformed to SAP by the following simple (new) procedure. Let $I = (V, E, T, T_L, c, p)$ be a feasible NWPTSTP—note that the feasibility of any NWPTSTP instance can be checked efficiently (Sun et al., 2020). As before, we assume $|T| > 2$. Let (V, A) be the bidirected graph corresponding to (V, E). First, we assume that $T \neq T_L$. Choose any $r \in T \setminus T_L$. Let $A' := A \setminus \{(t, v) \in A \mid t \in T_L \setminus \{r\}\}$. Further, remove all arcs in $\delta^-(r)$ from A'. Next, define $c' : A' \to \mathbb{Q}_{\geqslant 0}$ by $c'((v, w)) := c((v, w)) + p(w)$ for all $(v, w) \in A'$. The SAP $I' := (V, A', T, c', r)$ is feasible, and any optimal solution to I' can be readily transformed to an optimal solution to I—by taking the undirected equivalent of each arc contained in the optimal solution to I'. If $T = T_L$, choose any $r \in T_L$ and proceed as before. Finally, increase the weight $c'(a)$ for all $a \in \delta^+(r)$ by a sufficiently large constant.

Within SCIP-Jack, we simply apply the above transformations to any PTSTP or NWPTSTP instance, and treat the resulting problem as a customary SPG or SAP.

Computational results

In Table 5.1 we provide aggregated results on 6000 NWPTSTP instances from Sun et al. (2020). The instances have between 127 and 810 vertices, and between 916 and 6076 edges. Because of the large number of instances we do not provide instance-wise computational results in the appendix.

Table 5.1: Computational results for NWPTSTP instances.

Test-set	# instances	# solved	mean time [s]	maximum time [s]
IND	6000	6000	0.0	0.1

All instances from Table 5.1 can be solved in less than 0.1 seconds. We also note that for these instances SCIP-Jack is (orders of magnitude) faster than specialized NWPTSTP heuristics—and, being exact, also the solution quality is consistently better. See Sun et al. (2020) for more details.

5.2 The Steiner arborescence problem

As the Steiner arborescence problem (SAP) was already introduced in Section 1.1.3, we do not provide a definition here. Since any SPG can be solved as a (bidirected) SAP, the SAP is $\mathcal{NP}$-hard as well. Further theoretical results, concerning complexity and approximability, can be found in Charikar et al. (1998); Halperin and Krauthgamer (2003). An overview of SAP heuristics and partly also exact algorithms is given in Siebert et al. (2020b). In contrast to the SPG, we allow arcs of cost 0 for the SAP, since such arcs cannot be contracted without possibly losing all optimal solutions.

Algorithms

For primal heuristics, we use an adaptation of the shortest-path SPG heuristic introduced in Section 2.5.1. Similarly, the arc weights used by the heuristic are modified according to the current LP solution during branch-and-bound. Furthermore, we run the shortest-path heuristic on the subgraph corresponding to the arcs of reduced-cost 0 obtained from dual-ascent.

Concerning reduction techniques, we note that most of the SPG methods cannot easily be extended to SAP. Most of the SAP reductions that we apply were already present in the SCIP-Jack version developed prior to this thesis. However, these are rather simple. Importantly, for this thesis we additionally apply the dual-ascent reduction method, see Section 2.3.2.

Computational results

Name	# Instances	$\|V\|$	$\|E\|$	Status	Description
Gene	10	335-602	456-858	solved	Sparse, non-bidirectional, instances with $c \equiv 1$. From a genetics application (Johnston et al., 2000).
Gene2002	9	297-484	396-706	solved	
NET	25	77120-225739	116357-321169	unsolved	Instances from telecommunication network design.

Table 5.2: Details on SAP benchmark instances.

The SAP benchmarks instances described in the literature are unfortunately rather small. An overview on these instances is provided in the first two rows of Table 5.2. Additionally, we use 25 real-world, but non-public, network design SAP instances, obtained from one of the largest German telecommunication companies (which applies SCIP-Jack to solve such problems). Statistics of these instances are given in the last row of Table 5.2.

SCIP-Jack is able to solve all instances of the two smaller test-sets within less than 0.1 seconds. All these instances are solved already during presolving. For the much larger instances from the *NET* test-set SCIP-Jack takes considerably longer, but is still able to solve all instances within half an hour. We note that several of the largest *NET* instances cannot be solved by SCIP-Jack within the two hours time limit without the SAP enhancements implemented as part of this thesis.

Table 5.3: Computational results for the SAP instances.

Test-set	# instances	# solved	mean time [s]	maximum time [s]
Gene	10	10	0.0	0.0
Gene2002	9	9	0.0	0.0
NET	25	25	9.4	1486.8

As to other results from the literature, Siebert et al. (2020b) show that their heuristic, dynamic programming based algorithm outperforms several other algorithms from Watel and Weisser (2016). Unfortunately, the solver from Siebert et al. (2020b) is not publicly available, and no explicit run times are given in Siebert et al. (2020b). However, the solver requires the computation of all-to-all shortest paths, which usually is already drastically slower than the entire run-time of our solver. Due to this prohibitively large run-time, the authors in Siebert et al. (2020b) confine their computational experiments to instances with at most 3500 nodes, whereas we can solve instances with more than 200 000 nodes within minutes to proven optimality.

5.3 The node weighted Steiner tree problem

The *node-weighted Steiner tree problem* (NWSTP) is a generalization of the SPG that also includes (non-negative) vertex weights. Given an undirected graph $G = (V, E)$, vertex weights $p : V \to \mathbb{Q}_{\geqslant 0}$, edge costs $c : E \to \mathbb{Q}_{\geqslant 0}$, and a set $T \subseteq V$ of terminals, the objective is to find a tree S with $T \subseteq V(S)$ that minimizes:

$$\sum_{e \in E(S)} c(e) + \sum_{v \in V(S)} p(v).$$

The NWSTP has been the subject of several publications, see e.g. Buchanan et al. (2018); Guha and Khuller (1999); Moss and Rabani (2001), although most focus on theoretical aspects. Besides SCIP-Jack, the best alternative NWSTP solver is described in Leitner et al. (2018a). As we have shown in Section 4.2.2, any NWSTP can be transformed to an PCSTP by only changing its vertex weights. Computationally, we simply use this transformation for any NWSTP instance and treat the resulting problem as a normal PCSTP. Computational results for NWSTP are given in Section 4.5, as part of the PCSTP evaluation. It is also shown that SCIP-Jack considerably outperforms the solver from Leitner et al. (2018a).

5.4 The Euclidean and the rectilinear Steiner minimum tree problems

The *rectilinear Steiner minimum tree problem* (RMSTP) is defined as follows: Given $k \in \mathbb{N}$ points in the Euclidean plane, find a shortest tree consisting just of vertical and horizontal line segments and containing all k points. The RMSTP can be seen as a variant of the Euclidean Steiner minimum tree problem (EMSTP), see Section 2.1.1: Instead of using the L_2 norm, the RMSTP uses the L_1 norm for computing distances.

The RMSTP is $\mathcal{NP}$-hard, as for example proven in Garey and Johnson (1977)[20]. The RMSTP is one of the best known Steiner tree relatives, and has been the subject of various research articles and books, see e.g. Brazil and Zachariasen (2015); Emanet (2010); Hwang et al. (1992). For recent complexity results see Cambazard and Catusse (2018); Fomin et al. (2020). For practical exact algorithms see Juhl et al. (2018). A typical RMSTP application is VLSI design, see Brazil and Zachariasen (2015).

A generalization of the RMSTP to the d-dimensional case, with natural $d \geqslant 2$, has also been described in the literature. Real-world applications in up to eight dimensions can for example be found in cancer research, see Chowdhury et al. (2013). Another variant of the RMSTP is the *obstacle-avoiding rectilinear Steiner minimum tree problem* (OARMSTP), see e.g. Brazil and Zachariasen (2015). This problems includes the additional condition that the minimum-length rectilinear tree does not pass through the interior of specified *obstacles*, which are axis-aligned rectangles. Such obstacles occur for example in VLSI design.

Algorithms

Hanan (1966) proves that the RMSTP can be reduced to the *Hanan grid*, which is obtained by constructing vertical and horizontal line segments through each given point of the RMSTP. In this way, the RMSTP can be reduced to an SPG. In SCIP-Jack, both this construction and its multi-dimensional generalization, see Snyder (1992), is used by default to transform any RMSTP to SPG.

For two-dimensional RMSTP an empirically much stronger solution approach can be obtained by using full Steiner trees (FSTs)—which were described in Section 5.1. We delineate the approach in the following. For a detailed description see Warme et al. (2000). In the first phase, called *generation*, one creates a set of FSTs that is guaranteed to contain a minimum Steiner tree. In the second phase, called *concatenation*, one selects a subset of the generated FSTs that induces a minimum Steiner tree. Warme (1998) introduced the seminal idea to reduce the FST concatenation to finding a minimum spanning tree in a hypergraph whose vertices are the terminals and whose (hyper-)edges correspond to the generated FSTs. This idea forms the basis of many theoretical results for RMSTP and SPG, see e.g. Byrka et al. (2013). Also, this hypergraph approach is used in the well-known RMSTP solver GeoSteiner, see Juhl et al. (2018). A similar approach can be used for solving EMSTP (Juhl et al., 2018).

A simpler method (for both RMSTP and EMSTP) is suggested by Polzin and Daneshmand (2003): By taking the union of the edge sets of the FSTs generated in

[20] Note that this reference is an article, and not *the* Garey and Johnson.

the first phase, the FST concatenation can be reduced to an SPG. This approach was shown to be faster than (a previous version of) GeoSteiner when the SPG solver developed by the authors of Polzin and Daneshmand (2003) was used. Recall that we showed in Chapter 2 that the solver by Polzin and Daneshmand (2003) is outperformed by SCIP-Jack. Polzin and Daneshmand (2003) use the FST generation provided by GeoSteiner. Note, however, that the concatenation phase usually takes much longer than the FST generation for large instances. In the following computational results we will compare SCIP-Jack with the latest version of GeoSteiner by using the FST union method described above.

Computational results

The current version 5.1. of GeoSteiner has seen many improvements compared to its predecessor, see Juhl et al. (2018) for details, Furthermore, unlike its predecessor, GeoSteiner 5.1. is freely available. Just as SCIP-Jack, GeoSteiner provides an interface to CPLEX for solving LPs during branch-and-cut.

As we have already given various results for RMSTP, and OARMSTP instances in Section 2.7.3, we concentrate on EMSTP in the following. We note however, that SCIP-Jack significantly outperforms GeoSteiner on the just mentioned RMSTP and OARMSTP test-sets. In Table 5.4 we give results for Euclidean instances from Juhl et al. (2018) with 25 thousand (*EST-25k*), 50 thousand (*EST-50k*), and 100 thousand (*EST-100k*) points in the plane. For all these problems, the FSTs have been generated by GeoSteiner. For *EST-25k* the mean and maximum times of SCIP-Jack are between one and two orders of magnitude faster those of GeoSteiner. Moreover, 7 of the 15 instances from *EST-50k* are solved for the first time to optimality—in at most 196 seconds. On the other hand, GeoSteiner cannot solve these instances even after seven days of computation (Juhl et al., 2018). For *EST-100k*, GeoSteiner even leaves 12 of the 15 instances unsolved after one week of computation. In contrast, we solve all these instances (for the first time) to optimality in less than 15 minutes.

Unfortunately, Polzin and Vahdati-Daneshmand (2014) do not report results for any of these instances. However, the solver by Pajor et al. (2017), which won the SPG heuristics category at the 11th DIMACS Challenge, does not reach the upper bounds from GeoSteiner on any of the *EST-25k*, *EST-50k*, and *EST-100k* instances.

Test-set	# instances	# solved	mean time [s]	maximum time [s]
ESMT-R25	15	15	43.2	54.6
ESMT-R50	15	15	128.2	196.5
ESMT-R100	15	15	477.9	729.7

Table 5.4: Results for Euclidean Steiner tree instances.

5.5 The degree constrained Steiner tree problem

The *degree-constrained Steiner tree problem* (*DCSTP*) is a generalization of SPG with additional degree constraints: One is given an undirected graph $G = (V, E)$, a set of terminals $T \subseteq V$, edge costs $c : E \to \mathbb{Q}_{>0}$, and a function $b : V \to \mathbb{N}$. The objective of the DCSTP is to find a Steiner tree $S \subseteq G$ that satisfies for all $v \in V(S)$

$$\delta_S(v) \leqslant b(v), \tag{5.1}$$

and minimizes $c(E(S))$.

A comprehensive discussion of the DCSTP, including its applications in biology, can be found in Liers et al. (2016).

Algorithms

We use a variation of the shortest-path heuristic that also takes the degree constraints into account. For example, the heuristic checks before each extension of the current solution whether any degree constraint would be violated. While the previously discussed reduction techniques cannot be applied for the DCSTP, it is still possible to use the dual-ascent method (while ignoring the additional constraints), as it provides a feasible lower bound and valid reduced costs. We note, however, that small degree constraints can considerably impede the (empirical) strength of both primal and dual algorithms used for DCSTP in SCIP-Jack.

Computational results

Results on the (real-world) DCSTP instances from the 11th DIMACS Challenge can be found in Table 5.5. These instances have up to 832 vertices and 345 696 edges—so they are quite dense. The second column of Table 5.5 lists the number of instances in the test-set, the third column states the number of instances solved to optimality within the time limit. The next two columns consider only instances that could be solved to optimality. We report the shifted geometric mean of the numbers of branch-and-bound nodes, and of the run time. The final two columns give results for all instances that could not be solved to optimality within the time limit. First, the mean number of branch-and-bound nodes is given, second the arithmetic mean of the optimality gaps.

Other results for those instances are given in Liers et al. (2016) and Fischetti et al. (2017). Compared to the specialized solver by Liers et al. (2016), SCIP-Jack solves several more instances to optimality. On the instances that can be solved by Liers et al. (2016), SCIP-Jack is an order of magnitude or more faster. The machine used by Liers et al. (2016) is described as an Opteron processor with 64 GB RAM and 12 cores. Unfortunately, no further details are given. We note that this processor type has a clock rate between 1.4 and 3.5 GHz. However, any 12-core Opteron processor generation has at least 2.4 GHz—compared to the 3.4 GHz of our machine. Compared to the solver described in Fischetti et al. (2017), SCIP-Jack is slightly faster with respect to the shifted geometric mean (we use the DIMACS benchmark score to compensate for the different machine in Fischetti et al. (2017)). Fischetti et al. (2017)

solves 12 instances within one hour, whereas SCIP-Jack solves 14 in the same time. Also, the average gap is much smaller for SCIP-Jack on the unsolved instances: it is less than 50% of that reported by Fischetti et al. (2017), even when we exclude the instances for which Fischetti et al. (2017) does not find a primal bound. Weak primal bounds are also responsible for the relatively large number of instances left unsolved by SCIP-Jack. Thus, further development for DCSTP should concentrate on better primal heuristics. Finally, we note that the instance *TF105897-t3* is solved for first time to optimality.

			optimal		timeout	
Test-set	#	# solved	mean time [s]	mean nodes	mean nodes	∅ gap [%]
DCST-TreeFam	20	14	27.5	266.0	1866.2	31.1

Table 5.5: Computational results for DCSTP instances

5.6 The maximum-weight connected subgraph problem with budget

A close relative of the MWCSP is the *maximum-weight connected subgraph problem with budgets* (MWCSPB), see e.g. Backes et al. (2011). Compared to the MWCSP, this problem additionally provides *vertex costs* $c : V \mapsto \mathbb{Q}_{\geqslant 0}$ and a *budget* $B \in \mathbb{Q}_{\geqslant 0}$. The MWCSPB requires a connected subgraph $S \subseteq G$ with $c(V(S)) \leqslant B$ that maximizes $p(V(S)$. Both the rooted and non-rooted MWCSPB are described in the literature. In the rooted case we are additionally given a non-empty set $T_f \subseteq V$, which needs to be contained in any feasible solution. We concentrate on this rooted variant in the following. This section describes joint work with Henriette Franz, who made major contributions to the underlying work.

SCIP-Jack includes only a specialized heuristic for the MWCSPB, and otherwise treats the problem as an SAP with an additional constraint. Since this heuristic has been implemented by Henriette Franz—as part of her master thesis—we do not provide much information here, but simply refer to Franz (2019). For computational results we also refer to Franz (2019). Here we merely provide some remarks on reduction techniques for the MWCSPB.

A simple, but often powerful reduction technique can be devised by using an RMWCSP I' as a subproblem. Let $I' := (G, T_f, p')$ where p' is defined as $p'(v) := -c(v)$ for all $v \in V \setminus T_f$ and $p'(v) := 0$ otherwise. Let $v_0 \in V \setminus T_f$ be an arbitrary node and add v_0 to T_f. Let S' be an optimal solution to I'. If $-w(S') > B$, then v_0 cannot be part of any solution to the original MWCSPB instance. A similar criterion can be formulated for edges. The approach can be sped-up by first checking whether the solution found by an MWCSP heuristic satisfies the budget constraint. In this case, we do not need to consider the exact solution. This method might appear computationally prohibitive in practice. However, this is not the case, by virtue of the powerful MWCSP solver developed in Chapter 3. Also, this reduction method is naturally parallelizable. Indeed, this reduction technique is shown to be highly effective in practice (Franz, 2019). However, is has not yet been included into SCIP-Jack.

Another possible approach for eliminating a vertex (or edge) is to show that for any connected subgraph that contains this vertex there is another connected subgraph that does not and moreover is of no smaller weight and no higher cost. Again, we stress that no such techniques have been implemented into SCIP-Jack. For example, using the concept of dominating connected sets introduced in Chapter 3, we obtain the following result. As before, we define $T_p := \{v \in V \mid p(v) > 0\} \setminus T_f$.

Proposition 5.1. *Let $U \subseteq V \setminus (T_f \cup T_p)$ and $X \subseteq V \setminus U$ such that $(X, E[X])$ is connected and*

$$\big\{v \in V \setminus U \mid \exists\{v, w\} \in E, w \in U\big\} \subseteq \big\{v \in V \mid \exists\{v, w\} \in E, v \in X\big\} \cup X.$$

If

$$\sum_{u \in U} p(u) \leqslant \sum_{u \in X : p(u) < 0} p(u),$$

and

$$\sum_{u \in U} c(u) \geqslant \sum_{u \in X} c(u),$$

then there is an optimal solution S such that $U \nsubseteq V(S)$.

As special cases of Proposition 5.1, namely $|U| = 1$ and $|U| = 2$, one obtains criteria to delete vertices or edges.

Name	# Instances	$\lvert V\rvert$	$\lvert E\rvert$	Status	Description
GSTP1	8	349-1253	731-2319	solved	Sparse instances derived from a problem in VLSI design.
GSTP2	10	838-3177	1468-5907	unsolved	

Table 5.6: Details on GSTP benchmark instances.

5.7 The group Steiner tree problem

The *group Steiner tree problem* (*GSTP*) is a generalization of the Steiner tree problem, motivated from VLSI design, see Reich and Widmayer (1990); Hwang et al. (1992). Instead of terminals, one considers terminal groups. Given an undirected graph $G = (V, E)$, edge costs $c : E \to \mathbb{Q}_{\geqslant 0}$ and a set of vertex subsets $T_1, ..., T_s \subset V$, $s \in \mathbb{N}$, the GSTP requires a tree $S \subseteq G$ with $T_i \cap V(S) \neq \emptyset$ for all $i \in \{1, ..., s\}$ such that $c(E(S))$ is minimized. By interpreting each terminal t as a set of cardinality 1, any SPG can be considered as a GSTP. Thus, GSTP is a generalization of SPG (and in particular $\mathcal{NP}$-hard).

Algorithms

Voss (1988) shows that any GSTP instance can be readily transformed to an SPG, by adding an additional terminal for each terminal group and connecting this terminal to each vertex of the terminal group. This approach is usually used in the literature when it comes to solving GSTP, see e.g. Duin et al. (2004). A notable exception are the GSTP reduction techniques described in Ferreira and de Oliveira Filho (2007). However, the empirical success of these techniques is limited. We have not implemented any GSTP-specific methods into SCIP-Jack, but simply transform any GSTP to an equivalent SPG.

Computational results

Several GSTP test-sets, transformed to SPG, are included in the SteinLib and have been covered in Chapter 2. It was shown that SCIP-Jack constitutes the state of the art for solving these instances. Here, we additionally consider two test sets of unpublished (and proprietary) group Steiner tree instances derived from industry wire routing problems, as detailed in Table 5.6. These instances come already in preprocessed form.

Computational results are presented in Table 5.7. We note that two of these instances cannot be solved without the new SPG reduction algorithms introduced in this thesis.

Test-set	# instances	# solved	mean time [s]	maximum time [s]
GSTP1	8	8	1.4	5.1
GSTP2	10	10	56.5	3451.2

Table 5.7: Computational results for GSTP instances.

5.8 The hop constrained directed Steiner tree problem

The *hop-constrained directed Steiner tree problem* (HCDSTP) is a generalization of the SAP, see Burdakov et al. (2014). Let (V, A, T, c, r) be an SAP instance, and let $H \in \mathbb{N}$ (called *hop limit*). A feasible solution S to the SAP is feasible for the HCDSTP if additionally:

1. $|A(S)| \leqslant H$,
2. $\delta^+_S(t) = 0 \; \forall t \in T \setminus \{r\}$.

The HCDSTP asks for feasible solution S of minimum cost $c(A(S))$. Heuristics (both primal and dual) for the HCDSTP can for example be found in Burdakov et al. (2014); Pugliese et al. (2018). Real-world applications of the HCDSTP include the three dimensional placement of drones for multi-target surveillance, see e.g. Olsson et al. (2010). Finally, we note that in the Steiner tree literature the term *hop constrained* is also used for problem classes where for any feasible solution the number of edges between the root and any node is bounded by a constant, see e.g. Voß (1999).

Algorithms

The flow-balance directed-cut formulation (Formulation 1.1) used by SCIP-Jack can be easily extended to handle the HCDSTP by first removing all outgoing arcs from each terminal, and second adding the following constraint to Formulation 1.1:

$$y(A) \leqslant H. \tag{5.2}$$

Most reduction techniques and heuristics introduced so far cannot easily be extended to HCDSTP. For example, many reduction techniques described in this thesis remove or include edges if a less costly alternative sub-graph can be found. However, these techniques disregard whether this alternative sub-graph includes a larger number of edges. Nevertheless, some previously introduced bound-based reduction techniques can be adapted to HCDSTP.

First, note that dual-ascent based reductions can still be applied, despite the additional constraint: the corresponding dual variable can simply be set to 0. Additionally, the terminal decomposition concept described in Section 2.3.2 can be adapted for HCDSTP. Importantly, all above reductions techniques require a primal bound. To this end, we use a simple modification of the shortest-path heuristic, which was already included in SCIP-Jack prior to this thesis. We perform the shortest path heuristic for modified arc costs c' with $c'(a) := 1 + \alpha c(a)$ for all $a \in A$, where $\alpha > 0$. Implementing more refined primal heuristics could considerably improve the performance. For example, the local-search heuristics introduced in Burdakov et al. (2014) would be promising candidates.

Computational results

Computational results on a number of benchmark instances (some with more than 600 000 arcs) from the 11th DIMACS Challenge are provided in Table 5.9. See Table 5.8 for more details on the instances.

Name	# Instances	$\|V\|$	$\|E\|$	Status	Description
gr12	19	809	7430-44696	solved	Instances derived from 3D placement of unmanned aerial vehicles (Burdakov et al., 2014).
gr14	20	3209	115502-643552	unsolved	

Table 5.8: Details on HCDSTP benchmark instances.

All instances can be solved to optimality. Notably, six of these instances are solved for the first time to optimality. Details are given in Appendix B.8. Computational results for these instances are also given in Burdakov et al. (2014); Pugliese et al. (2018). However, only (primal and dual) heuristics are used in these articles. Especially on the gr14 instances the run times of Burdakov et al. (2014); Pugliese et al. (2018) are shorter, but at the same time the primal bounds are worse for more than half of these instances. Moreover, experiments on the larger test-set gr16 (with more than 8 000 000 arcs) were performed, but SCIP-Jack ran out of memory for all but three instances. However, these three instances *wo11-gr16-cr100-tr100-se10*, *wo11-gr16-cr200-tr100-se3*, and *wo12-gr16-cr200-tr100-se9* could be solved for the first time to optimality—with optimal values of 121 234, 54 163 and 47 687, which also notably improves on the previously best known bounds (Burdakov et al., 2014; Gamrath et al., 2017; Pugliese et al., 2018).

Test-set	# instances	# solved	mean time [s]	maximum time [s]
gr12	18	18	0.4	6.8
gr14	21	21	85.5	2130.8

Table 5.9: Computational results for HCDSTP instances

Chapter 6

Implementation and parallelization

So far, this thesis has mostly concentrated on providing and proving mathematical results, as well as on formally describing and analyzing new algorithms. In this chapter, we enter the somewhat more mundane realms of implementation and algorithm engineering issues. While the theoretical algorithm design is usually the far more decisive issue, a state-of-the-art implementation cannot afford to ignore the realities of modern computer architecture—such as CPU cache. Additionally, the ubiquity of multiple CPU cores and even intra-core parallelism in modern computer systems strongly suggests the use of parallel algorithms and implementations.

This chapter offers an overview of the software implementations done for this thesis, and furthermore provides algorithm engineering details for key components. Finally, we describe the, shared- and distributed-memory, parallel extensions of the newly developed solver. We assume some familiarity with the basics of computer architecture and programming. For an excellent introduction to computer architecture and related concepts, the reader is referred to the book by Bryant and O'Hallaron (2015).

6.1 SCIP-Jack

This section provides some conceptual insight in, and background on the Steiner tree solver developed as part of this thesis: SCIP-Jack.

6.1.1 The origins

SCIP-Jack derives its name from two other solvers: SCIP and Jack-III.

SCIP is a non-commercial solver for MIP, mixed-integer nonlinear programming, and constraint integer programming. Additionally, SCIP can be used as a customized branch-and-cut framework. The plugin-based design of SCIP provides a convenient method of extension, and allows for a strong control of the solving process. For more information on SCIP see Achterberg (2009); Gamrath et al. (2020).

Jack-III is a solver for (classic) SPG introduced by Koch and Martin (1998). It uses a branch-and-cut approach based on the bidirected cut formulation ($BDCut$). Furthermore, it comprises several preprocessing techniques and heuristics. At the time of publishing, Jack-III was able to solve all problem instances that had hitherto been discussed in the literature to optimality. Furthermore, Jack-III outperformed several previously introduced SPG solvers, often by a wide margin (although no comparison with solver by Duin (1993) is given in the literature). However, the solver introduced by Polzin and Daneshmand (2002) four years later, drastically outperforms Jack-III, solving significantly more instances and being often more than three orders of magnitude faster on the remaining non-trivial ones, see Polzin (2003).

Prior to this thesis, an early version of SCIP-Jack was borne out the integration of Jack-III into SCIP. In this process, the hand-tailored branch-and-bound routine, and the cutting plane management of Jack-III were replaced by those of SCIP. The author of this thesis was the main developer of this early SCIP-Jack version, but significant contributions were made by Gerald Gamrath, Thorsten Koch, Stephan J. Maher, and Michael Winkler. Notably, Thorsten Koch was the main developer of Jack-III.

However, more than 95 percent of the source code in the current version of SCIP-Jack has been newly implemented as part of this thesis (by the author of this thesis).

Figure 6.1: Depiction of a skipjack tuna, see Wikipedia (2021).

6.1.2 The solver

SCIP-Jack encompasses more than 110 000 lines of code and is written entirely in C.[21] Besides the parameters of SCIP, the user is given more than a hundred Steiner tree specific parameters to control the solving behavior of SCIP-Jack. For reading Steiner tree instances, both the widely used *.stp* file format (Koch et al., 2001) and the *.gr* (Bonnet and Sikora, 2019) file format are supported. The final optimal solution, as well as any intermediary feasible solution, can be obtained by the user in the DIMACS format (DIMACS, 2015).

The use of a general MIP solver renders the model to be solved highly pliant, which is of central importance to the generic solving approach employed in this thesis. Furthermore, a general framework allows one to avoid the tedious implementation of generic components such as branch-and-bound, and cut management. In particular, MIP solvers usually provide a filtering of cuts to improve numerical stability and efficacy. SCIP, being freely available for academic research and providing the above described features, seems a natural choice. The plugin-based structure of SCIP also makes it possible to readily integrate our various algorithmic components within a branch-and-cut method.

[21] Somewhat to the regret of the author of this thesis, who would nowadays have preferred C++ or Rust as the programming language of choice.

On the downside, many native methods of SCIP are prohibitively slow or memory consuming for large or even medium-scale Steiner tree instances. This behavior can be attributed to the fact that MIP instances that are commonly used for benchmarking are much smaller than typical Steiner tree instances. As a consequence, we perform the Steiner tree specific presolving before initializing the problem in SCIP. Otherwise, many large-scale Steiner tree instances covered in this thesis would not fit into main memory on our default machines. Additionally, most native, general-purpose algorithms of SCIP such as non-trivial presolving, domain propagation, primal heuristics, conflict analysis, or generic cutting planes are deactivated in SCIP-Jack. Besides being too slow, most of these algorithms are (empirically) mostly not effective for Steiner tree instances. A notable exception are $\{0, 1/2\}$-cuts (Caprara and Fischetti, 1996), which are usually computed in short time, and which are reasonably effective for some Steiner tree instances.

6.2 Implementation details of key components

This section contains a selection of (auxiliary) algorithms, data structures, and implementation aspects that are of central importance to the practical performance of SCIP-Jack. We will mostly use the notion of (one-dimensional) arrays instead of vectors. Further, we will use zero-based indexing unless noted otherwise. Given an array A of size n, the i-th entry with $i \in \{0, 1, ..., n-1\}$ of A will be referred to as $A[i]$.

6.2.1 Graph data structures

In the following, we describe different data structures for sparse graphs used in SCIP-Jack. While SCIP-Jack also contains data structures for dense graphs, these are only used for (smaller) auxiliary graphs. The existence of multiple sparse graph data structures reflects the conflicting needs of fast (cache-efficient) access, and efficient adaptation to dynamic graph changes. In addition to the two data structures described in the following, SCIP-Jack also includes a fully dynamic graph data structure (i.e., with both edge insertion and deletion capability), which was already present in Jack-III, see Koch (1995). This data structure allows for the inclusion of an edge $\{v, w\}$ in $O(1)$, and its deletion in $O(|\delta(v)| + |\delta(w)|)$ time. However, the data structure is significantly less cache-efficient for static operations such as graph traversals. Thus, we use this data structure mostly in the context of node-replacement reduction methods, where we frequently have to both insert and delete edges. For more information on graph data structures and implementation aspects see Kepner and Gilbert (2011).

Compressed sparse row

Compressed sparse row (CSR) is a standard format for storing sparse matrices, see e.g. Davis (2006). It has also been widely used for sparse graphs, see e.g. Kepner and Gilbert (2011). Although it is fairly simple, we discuss the CSR graph format in more detail in the following, since we will use several adaptations of this format thereafter. Given an undirected graph G, the CSR format can be used by applying it

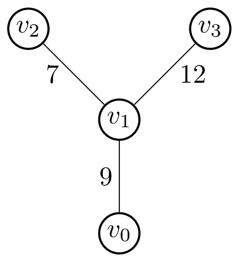


Figure 6.2: Weighted graph.

to the incidence matrix of the bidirected equivalent of G. In the following, we describe the graph CSR format for a (possibly bi-) directed graph (V, A) with arc weights c. Let $\{v_0, v_1, ..., v_{n-1}\} := V$ and $\{a_0, a_1, ..., a_{m-1}\} := A$. Let *Starts* be an array of size $n + 1$, and *Heads*, *Weights* be arrays of size m. Set $\mathit{Starts}[0] := 0$ and define $\mathit{Starts}[i] = \mathit{Starts}[i-1] + |\delta^+(v_i)|$ for $i = 1, ..., n$. For any vertex $v_i \in V$, the sub-array $\mathit{Heads}[\mathit{Starts}[i], \mathit{Starts}[i]+1, ..., \mathit{Starts}[i+1]-1]$ stores all vertex indices of $N^+(v_i)$. I.e., for all j with $v_j \in N^+(v_i)$, there is a $k \in \{\mathit{Starts}[i], \mathit{Starts}[i]+1, ..., \mathit{Starts}[i+1]-1\}$ with $\mathit{Heads}[k] = j$. Equivalently, the sub-array $\mathit{Weights}[\mathit{Starts}[i], \mathit{Starts}[i] + 1, ..., \mathit{Starts}[i + 1] - 1]$, stores the weights of the arcs $\delta^+(v_i)$. We do not assume any order of the entries in $\mathit{Heads}[\mathit{Starts}[i], \mathit{Starts}[i] + 1, ..., \mathit{Starts}[i + 1] - 1]$.

For the undirected graph shown in Figure 6.2, a CSR representation can be written as follows.

- *Starts*: $[0, 1, 4, 5, 6]$
- *Heads*: $[1, 0, 2, 3, 1, 1]$
- *Weights*: $[9, 9, 7, 12, 7, 12]$

Dynamic compressed sparse row

While allowing for cache-efficient access, the CSR format is ill suited for any graph changes, such as edge deletions. For this reason, we use a slightly modified data structure in settings where edge deletions are performed (so in particular during most of the reduction process). Instead of *Starts*, we keep two arrays *Starts* and *Ends* of length n. We set $\mathit{Starts}[0] := 0$ and define $\mathit{Starts}[i] := \mathit{Ends}[i-1] := \mathit{Starts}[i-1] + |\delta^+(v_i)|$ for $i = 1, ..., n-1$. Finally, we set $\mathit{Ends}[n-1] := \mathit{Starts} + |\delta^+(v_{n-1})|$. Equivalent formats have aleady been suggested for storing the constraints matrix during LP presolving, see Elble (2010). Note that in the actual implementation we use the *struct* construct of the *C* language to make sure that with each access to the row start of a vertex, also the row end pointer is being loaded into (L1) cache. We will refer to this extension of CSR as *dynamic compressed sparse row* (DCSR).

Deleting an arc $a = (v_i, v_j) \in \delta^+(v_i)$ can now be performed in $O(|\delta^+(v_i)|)$ as follows. Let $k \in \{\mathit{Starts}[i], \mathit{Starts}[i] + 1, ..., \mathit{Ends}[i] - 1\}$ be the index with $\mathit{Heads}[k] = j$. Set $\mathit{Heads}[k] := \mathit{Heads}[\mathit{Ends}[i] - 1]$, and decrement $\mathit{Ends}[i]$; adapt *Weights* equivalently.

6.2.2 Bottleneck Steiner distances

Duin and Volgenant (1989b) show that the (SPG) bottleneck Steiner distance s for all pairs of vertices can be computed in time $O(n(m + n \log n))$ and space $\Theta(n^2)$. However, even for (nowadays) medium-scale SPG instances with a few ten thousand vertices, both the run-time and the space are prohibitive. Thus, several authors have suggested to approximate $s(v, w)$. In the following, we suggest two new algorithms to approximate $s(v, w)$. The first algorithm approximates bottleneck Steiner distances such that the corresponding subgraph contains at least one terminal. The second algorithm covers the case of no intermediary terminals. The second algorithm can also easily be adapted for PCSTP and MWCSP.

Distances with terminals

Polzin and Daneshmand (2001b) suggest the following procedure. Initially, compute to each non-terminal v_r the (constant) k $\underline{d}$-nearest terminals $\underline{v}_{r,1}, ..., \underline{v}_{r,k}$ (see Section 2.3.2). For any $v_i, v_j \in V \setminus T$, use the upper bound

$$\hat{s}(v_i, v_j) := \min_{a,b \in \{1,\dots,k\}} \left\{ \max\{\underline{d}(v_i, \underline{v}_{i,a}), s(\underline{v}_{i,a}, \underline{v}_{j,b}), \underline{d}(v_j, \underline{v}_{j,b})\} \right\} \tag{6.1}$$

on $s(v_i, v_j)$. Empirically, that bound has shown to be a tight approximation of s. The $\hat{s}$ values are not pre-computed, in order to preserve an $O(m+n \log n)$ bound. Also, in most cases not all of the k^2 possible combinations have to be examined. For example, if $\max\{\underline{d}(v_i, \underline{v}_{i,1}), \underline{d}(v_j, \underline{v}_{j,1})\} > c(\{v_i, v_j\})$, the edge elimination test for $\{v_i, v_j\}$ can already be aborted; see Polzin and Daneshmand (2001b) for more such conditions.

For constant $k \in \mathbb{N}$, Duin and Voss (1997) show how to compute the k $\underline{d}$-nearest terminals to all vertices in $O(m + n \log n)$. Thus, in the following we concentrate on efficiently computing the exact bottleneck Steiner distances between pairs of terminals. We suggest a new approach that requires a preprocessing of $O(m+n \log n)$ time and space, and constant time for each query. By definition, for any $t, u \in T$, the bottleneck Steiner distance $s(t, u)$ corresponds to the (standard) bottleneck distance between t and u in the distance network $D_G(T)$. Let Y be a minimum spanning tree in $D_G(T)$. One notes that $s(t, u)$ is equal to the maximum cost of an edge on the path between t and u in Y. Furthermore, Mehlhorn (1988) shows that Y can be built in $O(m + n \log n)$—without constructing $D_G(T)$ beforehand. See also Floren (1991) for a simpler and practically more efficient realization.

Given a minimum spanning tree Y in $D_G(T)$, one can trivially compute $s(t, u)$ for any terminals t, u in $O(|T|)$. However, already for the (standard) edge elimination test based on (6.1), this approach is rather slow in practice. Polzin and Daneshmand (2001b) suggest to compute the bottleneck Steiner distance for pairs of adjacent terminals by creating an instance of the offline lowest common ancestor (LCA) problem, see Tarjan (1979). This approach runs in $O(q + |T| \log |T|)$ time for q queries. However, all queries need to be known beforehand, which essentially requires to run the bottleneck Steiner distance edge elimination algorithm that uses (6.1) twice. Furthermore, such an offline algorithm is prohibitive for the extended reduction techniques discussed in Section 2.4. Note that there are also online algorithms for LCA that require merely linear preprocessing time, and constant time for each query, but they are not competitive in practice, see Fischer and Heun (2006).

Similar to Polzin and Daneshmand (2001b), we essentially build an LCA instance first. However, this instance only has half as many nodes as that used by Polzin and Daneshmand (2001b). Let $\{e_1, ..., e_{|T|-1}\}$ be the edges of Y, and assume that $d(e_i) \leqslant d(e_j)$ for any i, j with $1 \leqslant i < j \leqslant |T| - 1$. Next, we build a binary tree B with vertices $V_B = \{b_1, ..., b_{|T|-1}\}$, and an initially empty edge set E_B. Define $f : T \mapsto V_B \cup \{null\}$ initially by $f(t) := null$ for all $t \in T$. For all $i = 1, ..., |T| - 1$ proceed as follows. For all (two) $t \in e_i$: If $f(t) = null$, set $f(t) := b_i$. Otherwise, add an edge between the root of the subtree that contains node $f(t)$ and b_i. Further, make b_i the root of this new subtree. Finally, define $p : B_V \mapsto \mathbb{Q}_{\geqslant 0}$ by $p(b_i) := d(e_i)$ for all $b_i \in V_B$.

The above algorithm to construct B can be realized in $O(|T| \log |T|)$ by initially sorting the edges of Y, and using a union-find data structure (of size $2|T| - 1$). To see the benefit of B, let $t, u \in T$ with $t \neq u$. Further, let b_i be the lowest common ancestor of $f(t)$ and $f(u)$ in B. One notes that $p(b_i) = s(t, u)$.

Observing that we are not interested in the lowest common ancestor itself, but in its value p, we proceed as follows. First, we use the *Euler tour technique* by Tarjan and Vishkin (1984) on B. We consider B as a bidirected tree and traverse its arcs in a DFS fashion starting from the root. Each time a node b_i is visited, append the value $p(b_i)$ to an initially empty array A. For simplicity, we assume A to be 1-indexed. Finally, A has size $2|T| - 3$.[22] Let $g : B_v \mapsto \{1, ..., 2|T| - 3\}$ such that $g(b_i)$ is the first index A at which b_i has been visited during the Euler tour. Thus, in particular $A[g(b_i)] = p(b_i)$. Define $h := g \circ f$. Next, we use the *sparse table* technique, see e.g. Bender et al. (2005), to efficiently find the maximum of any interval of A. Initially, we precompute the maximum of all intervals whose length is a power of 2. Let $x := |A|$. Let M be a (1-indexed, 0-indexed) two dimensional array defined recursively by $M[i][0] := A[i]$ for $i \in \{1, ..., x\}$, and $M[i][j] := \max\{M[i][j-1], M[i+2^{j-1}][j-1]\}$ for every $i \in \{1, ..., x\}$ and any $j \in \{1, ..., \lfloor \log n \rfloor\}$. The idea is to exactly cover any interval of A by two overlapping entries of M. Now, let $t, u \in T$ be distinct vertices such that $h(t) \leqslant h(u)$. If $h(t) = h(u)$, then $s(t, u) = d(t, u) = A[h(t)]$. Otherwise, define $i := h(t)$, $j := h(u)$, and $z := \lfloor \log(j - i) \rfloor$. It holds that

$$s(t, u) = \max\big\{M[i][z], M[j - 2^z + 1][z]\big\}. \tag{6.2}$$

Concerning implementation, one notes that z is equal to the most significant bit of $(j - i)_2$. Modern C compilers provide intrinsics for this operation (if $j - i > 0$). However, we have decided for a fully portable solution and use a table look-up instead.

Finally, we note that if the number of terminals is not more than 100, we do not use the above approach, but compute and store the Steiner bottleneck distance between all pairs of terminals. Computing these distances by an offline variant of the above approach takes $O(|T|^2 + m + n \log n)$ time and $\Theta(|T|^2)$ space.

[22] The Euler tour technique is also used to reduce LCA to the range minimum query problem, see e.g. Bender et al. (2005).

Distances without terminals

To also cover Steiner bottleneck distances that correspond to paths that do not include any terminals, several authors, e.g. Hwang et al. (1992); Polzin and Daneshmand (2001b), suggest to run a limited version of Dijkstra's algorithm from both endpoints of the edge to be eliminated. However, such a test can be considerably time consuming in practice. Here, we describe a more efficient alternative. Additionally, the proposed heuristic also takes the implied profit of vertices into account, and thus even serves to approximate the implied bottleneck Steiner distance.

Starting from a vertex v_0, the heuristic tries to delete several edges of $\delta(v_0)$ at once. Initially, define a distance array $\tilde{d}$ and a predecessor array *pred* as follows. For all $u \in V \setminus (\{v_0\} \cup N(v_0))$: $\tilde{d}[u] := \infty$ and $pred[u] := null$. For all $u \in N(v_0)$: $\tilde{d}[u] := c(\{v_0, u\})$ and $pred[u] := v_0$. Moreover, set $\tilde{d}[v_0] := 0$ and $pred[v_0] := v_0$. Finally, set $Q := N(v_0)$.

While $Q \neq \emptyset$ let $v := \arg\min_{u \in Q} \tilde{d}[u]$. For all $\{v, w\} \in \delta(v)$ proceed as follows. First, set $p_{vw} := \max\{p^+(v, \{e\}) \mid e \in \delta(v) : w, pred[v] \notin e\}$. If

$$\tilde{d}[v] + c(\{v, w\}) - \min\left\{c(\{v, w\}), p_{vw}, \tilde{d}[v]\right\} < \tilde{d}[w], \tag{6.3}$$

then set $\tilde{d}[w]$ to the left hand side of (6.3) and add w to Q. Further, set $pred[w] := v$. If (6.3) holds and $w \in N(v_0)$, then we can delete edge $\{v, w\}$.

Note that on the left hand side of (6.3) a possibly smaller value than p_{vw} is subtracted to prevent the algorithm from circling. Furthermore, note that a terminal might be used more than once for a profit calculation p_{vw} on one walk. However, since we subtract only a bounded part of the profit from the distance value in (6.3), the algorithm still works correctly. Note that one can extend the algorithm to cover the case of equality for edge deletion. In this case, one also needs to check whether (6.3) is satisfied with equality if $w \in N(v_0)$. In practice, one should bound the maximum number of visited edges. Additionally, one can abort the algorithm if $\min_{u \in Q} \tilde{d}[u] > \max_{e \in \delta(v_0)} c(e)$.

6.2.3 Extended reduction techniques

Initially, the reader is reminded that extended reduction techniques have only been implemented for SPG in this thesis. Thus, this section covers only the SPG. We note, however, that at least a partial extension of these methods to PCSTP and MWCSP is conceptually straightforward. We use a DFS strategy for the extension, see also Duin (2000). In this way, the re-use of intermediary results, such as MSTs, is simplified. Furthermore, we use the following criteria for the subroutines PROMISING and TRUNCATE. Depending on the size of the instance, we bound the maximum depth of the extension, and the maximum number of leaves allowed for an extension tree. Furthermore, we bound the maximum degree of any leaf along which we extend. No extensions along terminal leaves are performed.

In the following, we give details of several algorithms and data structures. Due to the complexity of the implementation, which encompasses more than 20 000 lines of C code, we need to be quite selective. Thus, we only focus on the most important components.

Storage and bookkeeping aspects

For simplicity, we restrict the following discourse to the extension of a single edge $\{v_0, v_1\}$ from the endpoint v_1. Recall that we perform extensions only in a DFS manner. I.e., we only extend the current extension tree from vertices that are at maximum distance from v_0 with respect to the number of edges. Figure 6.3 shows an exemplary extension tree in bold for the edge $\{v_0, v_1\}$. Edges that are not part of the given extension tree, but need to be considered in other extension trees are dashed. Further extensions of the bold extension tree are only possible from vertex v_7.

We use the following terms for describing the extension process. Let Y be an extension tree of $\{v_0, v_1\}$, and v be a leaf of Y. We define the *depth* of v in Y as the number of edges on the path from v to v_1 in Y. Each time we extend the current tree Y from a leaf v of depth $i-1$, we call all $\bar{L}(v, Y) := N(v) \setminus V(Y)$ the i-th *full extension level.* We call the subset of $\bar{L}(v, Y)$ that is used for the extending Y the i-th *partial extension level*, denoted by $L(v, Y)$. For the extension tree shown in Figure 6.3, the set $\{v_2, v_3, v_4\}$ is the $1st$ full extension level, and set $\{v_2, v_4\}$ the $1st$ partial extension level. We say that v is the *root* of both the partial extension level $L(v, Y)$ and the full extension level $\bar{L}(v, Y)$.

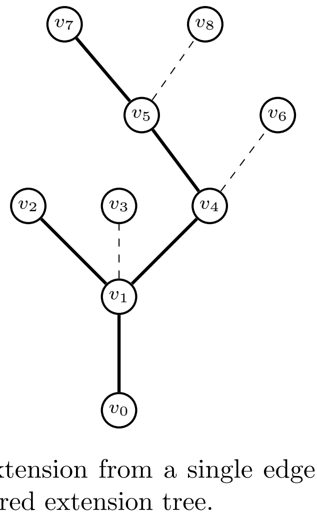


Figure 6.3: Illustration of extension from a single edge. The dashed edges are not part of the currently considered extension tree.

Throughout the extension of $\{v_0, v_1\}$, we store several auxiliary results, to avoid their continuous recomputation; most importantly, the bottleneck Steiner distances between leaves of the current extension tree, and the corresponding MSTs on the complete graphs induced by those leaves. In the following, we exemplarily describe the storage procedure for the bottleneck Steiner distances.

We call bottleneck Steiner distances between vertices that are in the same full extensions level *horizontal*, and bottleneck Steiner distances between vertices that are in the different full extensions levels *vertical.* For each vertex of a full extension level we store the (vertical) distances to all leaves of the current extension tree that are of smaller depth than the extension level. For example, for the 2nd full extension level, consisting of the vertices v_5 and v_6, we store the (vertical) bottleneck Steiner distances from both v_5 and v_6 to the vertices v_0 and v_2. For reasons of cache-efficiency, we keep all vertical and all horizontal bottleneck Steiner distances consecutively in one array, respectively. The distances can be efficiently queried by the use of a compressed system similar to a nested CSR format. For the horizontal distances we store for each vertex of a given full extension level the distances to all of its right siblings. E.g., in Figure 6.3 we store for vertex v_2 the distances to v_3 and v_4. For each vertex we keep the start index of the horizontal distances to its right siblings. These start indices are also kept continuously. For each full extension level we keep the index where the first of its start indices are stored.

Similarly, for each vertex of a full extension level, we keep the start index of its vertical distances. The latter are sorted according to the index of the corresponding ancestor leaf. E.g., for vertex v_5 we store the distances to v_0 and v_2. Note that the order of the ancestor leaves of any full extension level stays the same as long as the level is active. Again, we use the same nested start pointer storage already used for the horizontal distances.

As another speed-up method, we only reserve the spaces for the vertical and horizontal distances. The actual computation happens once a distance is queried for the first time. In this way, we avoid the computation of distances that are never used (because the search along the corresponding full extension level is truncated).

Computing and recomputing MSTs

Another important aspect is the storage of the MSTs on the complete graph induced by the current tree together with a pruning set. Note that while the use of minimum Steiner trees instead of MSTs is stronger, their computation is naturally more expensive. Therefore, we only use (not necessarily optimal) Steiner trees for extension trees with three leaves.

A classic result from Spira and Pan (1975) shows that it is possible to adapt an MST in $O(n)$ time after the insertion of a new vertex (and up to $O(n)$ edges). We use a different algorithm from Chin and Houck (1978), which also works in $O(n)$, but is practically more efficient than the approach by Spira and Pan (1975). Note that the deletion of a vertex is more expensive, taking $O(n^2)$ time. Thus, we only modify already computed MSTs by adding vertices, and never by deleting.

We store the following MSTs. First, before we add a new full extensions level, we compute and store the MST on the (complete) graph induced by all leaves of the extension tree with the new full level and without the root of the new full level. Second, whenever we add a new partial extension level, we compute and store the MST on the graph induced by all leaves of the extension tree including the new level. In this way, we can efficiently build new MSTs by extending already existing ones.

Furthermore, we can use this MST storage system together with our data structure for keeping the bottleneck Steiner distances to efficiently compute MSTs for partially contracted extensions trees: Consider the partial extension level $L(v, Y)$ such that v has the largest depth in Y. Store for each leaf w of Y that is not in $L(v, Y)$ the minimum among the bottleneck Steiner distances between w and a vertex in $L(v, Y)$. Note that these distances can simply be queried from the vertical distance storage for $\bar{L}(v, Y)$. Consider the vertices in $L(v, Y)$ as a single contracted node and compute a MST on the complete graph induced by this node together with the remaining leaves of Y—this MST can be readily obtained by extending an already computed one. If we cannot rule out Y with this MST, we proceed to the next lower partial extension level and implicitly contract both levels into a single node. Note that we just need to update the already computed distances from the remaining leaves of Y to this contracted node. We proceed in this way until all partial extension levels are contracted.

Finally, yet another advantage of the algorithm by Chin and Houck (1978) is the possibility to keep the MST to be extended in CSR format. This feature allows us to store several MSTs consecutively in a nested CSR format, similarly to the storage of the bottleneck Steiner distances described in the previous section.

Computing and recomputing bottleneck Steiner distances

As described in Section 6.2.2, approximate bottleneck Steiner distances along at least one terminal can be queried in constant time. However, as already observed by Polzin and Daneshmand (2002), these approximate values $\hat{s}$ lead to significantly worse results than the exact bottleneck Steiner distances for extended reduction tests. The authors suggest to use the value $\min\{\hat{s}(v, w), d(v, w)\}$ for vertices v, w instead. We follow this suggestion. Furthermore, just as Polzin and Daneshmand (2002), we do not compute all-to-all (standard) distances, but compute and store for each vertex v the distances to a constant number of nearest vertices. We sort these distances according to the indices of the corresponding end vertex of the shortest path. In this way, any (contained) distance can be readily queried by a binary search.

An equally important issue is the recomputation of the bottleneck Steiner distances after a graph modification. First, one can show that node replacements do not change the bottleneck Steiner distances. Thus, we only need to handle the deletion of edges. For the (standard) distances from each vertex to a constant number of nearest vertices, we proceed as follows. For each edge e we store all vertices v such that e is used in one of the (constant number of) shortest path distances stored for v. If edge e is deleted, we recompute the shortest path distances for all these v. In fact, we do so in a lazy fashion, i.e., we mark the vertices and only recompute the distances once they are required. Additionally, we keep state counters for all v to make sure that a vertex is not marked after the deletion of an edge e even though its shortest paths distances have already been updated and do not include e anymore.

The recomputation of bottleneck Steiner distances along terminals is more involved. Recall that we store for each vertex v_i the closest k terminals $\underline{v}_{i,1}, \underline{v}_{i,2}, ..., \underline{v}_{i,k}$. Furthermore, we store an MST for the distance graph $D_G(T)$. If an edge of the distance graph is deleted, we simply recompute the MST. Preliminary experiments have shown that such a deletion rarely happens and that the computing time is negligible. In contrast, the distances to the k closest terminals change, empirically, with almost every edge deletion. Thus, a complete recomputation is prohibitive, and a repairment algorithm is required. In the following, we describe the repairment of the distances to the closest terminal. This procedure is related to the reconstruction of Voronoi regions in the context of local-search heuristics described in Uchoa and Werneck (2010). The recomputation of the distances for the k'th nearest terminals with $k > 1$ is significantly more technical, and is therefore not presented here.

For each vertex $v_i \in V \setminus T$ we store the following information

- $base[v_i] := \underline{v}_{i,1}$;
- $pred[v_i]$: the last vertex before v_i on a shortest path from $base[v_i]$ to v_i;
- $dist[v_i] := d(v_i, \underline{v}_{i,1})$.

Let $\{v, w\}$ be the edge to be deleted. Assume $pred[w] = v$ (which implies $pred[v] \neq w$). Note that if both $pred[w] \neq v$ and $pred[v] \neq w$, we are already finished. First, we perform a graph traversal from w as follows: Initially, we set $U := \{w\}, Q := \{w\}$. While $U \neq \emptyset$, we remove any u from U and proceed as follows: For all $\{u, q\}$ such that $pred[q] = u$, set $base[q] := null$, $pred[q] := null$, $dist[q] := \infty$. Further, add q to U and Q.

Second, for each $q \in Q$ proceed as follows. For all $\{u, q\}$ with $u \notin Q$. If $dist[u] < dist[q] + c(\{u, q\})$, set $dist[u] := dist[q] + c(\{u, q\})$, $base[q] := base[u]$, and $pred[q] := u$. If after this processing still $pred[q] = null$ holds, remove q from Q. Finally, run a slightly modified version of Dijkstra's algorithm with all vertices of Q in the initial priority queue, and with distance values $dist$. During the computation of Dijkstra's algorithm we also update the $base$ values.

6.2.4 Separation algorithms

Since all problem classes described in this thesis are formulated (after suitable transformations) as some variant of the *DCut* formulation, a major algorithmic component of our solver is the separation algorithm for the constraints (1.3). We note that additional cuts, such as the flow-balance constraints, can be easily separated. Thus, we focus on the constraints (1.3) in the following.

It is well known that cut constraints such as (1.3) can be separated by using a maximum-flow algorithm—based on the classic max-flow/min-cut theorem. One merely needs to regard the values of the LP solution as capacities and compute a maximum-flow from the root r to each terminal $T \setminus \{r\}$. In Hao and Orlin (1992) an adaptation of a preflow-push algorithm is introduced that allows one to solve maximum-flows from a designated source to several other sink nodes with a run-time similar to that of a single maximum-flow computation. In Koch and Martin (1998) this algorithm is used for separating the constraints (1.3). Koch and Martin (1998) furthermore suggest to compute cuts of small cardinality by adding a small (additional) capacity $\varepsilon > 0$ to all arcs. While this approach deteriorates the run-time of computing minimum cuts, the time required for re-optimizing the linear program is often notably decreased, since the constraint matrix contains fewer non-zeroes.

For this thesis we have implemented a maximum-flow algorithm for separating (1.3) with the modifications described above. We note that this algorithm is usually several times faster than the implementation from Koch and Martin (1998). Just like Koch and Martin (1998), we use a push-relabel algorithm with additional heuristics—in particular we use the heuristics described in Cherkassky and Goldberg (1997). If we merely consider the run-time for computing the first maximum-flow, we observe a considerable speed-up (of up to an order of magnitude) as compared to the widely used push-relabel implementation from Cherkassky and Goldberg (1997)[23]. We note, however, that this speed-up holds in the context of Steiner tree separation problems, and a different behavior might be observed on other graphs. The speed-up can be attributed to a careful implementation, and the use of cache-efficient data structures. However, we also note that empirically only a smaller part of the overall separation time is spent in computing the first maximum-flow—even though the theoretical run-time for computing this first flow is the same as that for computing all flows.

6.3 Parallelization: Building Steiner trees on 43 000 cores

Parallel computing has become mainstream in the last decade. Also for Steiner tree problems there have been various publications considering parallel algorithms, see e.g. Bezenšek and Robič (2014); Ljubic (2020) for overviews. However, most of the reported computational results are obtained on test-sets that are considered too trivial to be included in this thesis (almost all of these instances can be solved in fractions of a second by SCIP-Jack). And even on these simple test-sets often enough no optimal solution is found, see Bezenšek and Robič (2014). Most of these publications concentrate on simple heuristics, which allow for good scalability, but are no match for state-of-the-art Steiner tree algorithms. On the other hand, the intricacy of state-of-the-art Steiner tree algorithms poses far more challenges for an efficient parallelization. Also, the usage of the Simplex algorithm, which is notoriously hard to efficiently parallelize, is problematic.

This section concentrates on the parallelization of several of the Steiner tree algorithms described in this thesis. We consider both shared- and distributed memory parallelizations.

6.3.1 Parallelizing heuristics and reduction methods

In the following, we describe the parallelization of several reductions methods and heuristics. The implementations are all shared-memory, and are realized with OpenMP (de Supinski et al., 2018). All parallelizations are still in an experimental stage, and not enough methods have been parallelized yet to obtain a significant parallel speed-up for most instances. Thus, we do not provide computational results in this section. Still, a short description of these methods is given because they open a promising route for further development.

One observes that most primal heuristics used in this thesis employ some version of the shortest-path heuristic described in Section 2.5.1. We always run the shortest-

[23] see `www.avglab.com/andrew/soft/` for the source code

path heuristic from several distinct start vertices—to increase the solution quality. Thus, a simple, so-called *embarrassingly parallel*, approach is to distribute these computations among the available threads. Using parallelization within a single run of the shortest-path heuristic does not seem promising, because of the short run-times, which cannot compensate the parallelization overhead. Since communication between the threads is restricted to the update of the best incumbent solution (if necessary), our embarrassingly parallel scheme scales quite well up to a handful of threads (say 8 or 16). Of course, the number of threads that can be efficiently employed is bounded by the number of distinct start vertices used by the shortest-path heuristic.

A corresponding, embarrassingly parallel, scheme could be employed for running dual-ascent in parallel from several root nodes—and store the lower bounds and reduced costs for each run. In this context, we note that Drummond et al. (2009) describe a sophisticated distributed parallelization of the dual-ascent heuristic. However, since our sequential implementation of dual-ascent is quite fast on most benchmark instances, any internal parallelization does not seem promising.

Concerning the parallelization of reduction techniques, one notes that most methods in this thesis loop over all edges or vertices, and check whether eliminations, contractions, or replacements are possible. If such an operation on a single vertex or edge does not change the validity of other reductions, we again obtain an embarrassingly parallel scheme: One simply needs to distribute all edges (or nodes) among the available threads. Unfortunately, that is not the case for most reduction methods, in particular not for the most time-consuming ones. On the other hand, communicating graph changes on the fly among the threads is prohibitive due to the short run-times of the individual checks. Thus, we do not apply any graph changes, but rather let each thread collect all possible reductions together with some *reduction proof*. For example, in the edge elimination test described at the end of Section 6.2.2, we store for each edge that could be eliminated, the corresponding path between its end points. Once all threads have become idle, we check sequentially for each of the stored edges whether the corresponding (alternative) path is still intact. If that is the case, we eliminate the edge. We sort the edges to be checked for elimination in non-descending order according to the number of alternative paths (needed for the elimination of other edges) in which they are contained. In particular, we first delete those edges that are not used in any alternative paths for other edges.

A similar scheme of reduction proofs can be used for extended reduction techniques. In this case, one can store for each reduction candidate the extension vertices that were used to rule it out. Once all threads have become idle, one only checks the extensions along these vertices—which are in general exponentially fewer extensions than in the original check.

6.3.2 Parallelizing branch-and-bound

Another seemingly promising candidate for parallelization within SCIP-Jack is the branch-and-bound search. However, most Steiner tree instances are solved at the root node of the branch-and-bound (B&B) tree, or even in preprocessing. Still, for several notoriously hard SPG instances many branching nodes are created—and those hard problems are also natural candidates for parallelization. In the following, we will concentrate on solving such hard instances by parallelizing the B&B search of SCIP-Jack, and using the computational power provided by supercomputers with thousands of CPU cores. This section is joint work with Yuji Shinano.

The framework

For parallelizing the branch-and-bound search we use the *Ubiquity Generator Framework* (UG) (Shinano et al., 2016), a software package to parallelize branch-and-bound based solvers—for both shared- and distributed-memory environments. More precisely, we use the software library included in UG for parallelizing extensions of SCIP. Usually, it is possible to employ this UG software library by adding only a small amount of glue-code (typically $100 - 200$ lines). However, several idiosyncrasies of SCIP-Jack (such as the preliminary use of reduction techniques) required to extend both SCIP-Jack and UG. In the following, we briefly describe UG, and go on to introduce the features newly added for parallelizing SCIP-Jack. For more details see Shinano et al. (2019b).

UG implements a *Supervisor-Worker load coordination scheme*, see e.g. Ralphs et al. (2018). Importantly, Supervisor functions make decisions about the load balancing without actually storing the data associated with the B&B search tree. In UG, the Supervisor is called `LoadCoordinator` (`LC`) and the Workers are called `ParaSolvers`. The B&B search tree data is managed by the `ParaSolvers`. The terminal nodes (subproblems) of the B&B search tree in the `ParaSolvers` are sent on demand to the `LC`; a set of subproblems in the `LC` works as a buffer to ensure subproblems are available to idle `ParaSolvers` as needed.

During the B&B process, SCIP-Jack selects a non-terminal vertex of the problem instance to be rendered a terminal in one B&B child node and to be excluded in the other child. These two operations are modeled in the underlying IP formulation by including one additional constraint. This procedure could not be used in previous versions of UG since branching on constraints was not supported. Therefore, a new feature for transferring branching constraints has been added to UG

A distinguishing feature of UG is the *layered presolving*, in which B&B tree nodes are transferred to the other `ParaSolvers` recursively and additional presolving is performed on the subproblems. Default MIP presolving realized in SCIP works without any additional code in this layered scheme. However, SCIP-Jack performs presolving before it formulates the subproblem as an IP. In order to realize this presolving, a callback to initialize the transferred subproblem has been added to UG. To retain previous graph based branching decisions, UG transfers the branching history together with each subproblem, enabling SCIP-Jack to change the underlying graph (by adding terminals and deleting vertices). Additionally, whenever a subproblem has been transferred, SCIP-Jack performs aggressive reduction routines to reduce the (modified) problem further, and translates the reductions into variable fixings by means of Proposition 2.33.

Computational results on supercomputers

Initially, we point out that all results reported in the following were obtained with previous versions of SCIP-Jack, due to resource constraints. We used two supercomputers. The first one (ISM) is a HPE SGI 8600 with 384 compute nodes, with each node consisting of two Intel Xeon Gold 6154 3.0GHz CPUs (18 cores$\times$2) sharing 384GB of memory, and an Infiniband (Enhanced Hypercube) interconnect. The other (HLRN III) is a Cray XC40 with 1872 compute nodes, each node consisting of two 12-core Intel Xeon IvyBridge/Haswell CPUs sharing 64 GiB of RAM, and with an Aries interconnect.

Due to resource constraints, we could only attempt to solve a few instances on the supercomputers. The best (and partly optimal) bounds obtained for these instances are shown in Table 6.3. We also report the best previously known primal bound for each instance. In the following, we give more insight into the solution process with UG/SCIP-Jack on a supercomputer. We provide details for one particular instance solved to optimality, and for one instance for which the best known primal bound could be improved.

We start with the PUC instance `hc9p`, which could be solved for the first time to optimality—by five restarted runs and by using up to 24 576 cores. Table 6.1 shows for each run: the supercomputer used, the computing time in seconds (racing time is shown in parentheses), the idle time ratio for all `ParaSolvers`, the number of B&B nodes transferred to `ParaSolvers`, primal and dual bounds, primal-dual gap, the number of B&B nodes generated, and the number of open B&B nodes. For each run the initial values are shown in the upper row, and the final values are shown in the lower row.

The initial primal solution to `hc9p` was found by a previous run of UG/SCIP-Jack. One notes that the final dual bound of a run is sometimes slightly different from the initial one in the following run. This means that the dual bound in the previous run was updated after the final checkpoint. One also observes that the number of open B&B nodes decreases strongly at restart, since the checkpointing mechanism only saves essential sub-tree roots. For example, run 1.1 ends up with 1 257 112 open B&B nodes, but run 1.2 starts with 15 open ones. This means that only 15 B&B sub-tree roots existed at the end of run 1.1 and the other sub-tree roots were descendants of one of the 15 B&B nodes. Notably, the idle time ratios for all runs are small, which indicates that the supercomputers are used efficiently.

Next, we describe the solution process for the PUC instance `hc11p`. We used two different strategies. First, a long, but small-scale run. Second, a short, but large-scale run. Statistics are given in Table 6.2. Run 1 on the ISM supercomputer generated 11 new incumbent solutions, with the best objective value being 119 297. In the followings runs 2.1 and 2.2 we started with the best of these solutions. Run 2.2 was conducted from the checkpoint file of run 2.1, since run 2.1 could not improve the

Run	Computer	Cores	Time (sec.)	Idle (%)	Trans.	Primal bound (Upper bound)	Dual bound (Lower bound)	Gap (%)	Nodes	Open nodes
1.1	ISM	72	604,796 (317)	< 0.3	738	30,242.0000	29,879.3721	1.21	0	0
						30,242.0000	30,058.9366	0.61	110,012,624	1,257,112
1.2	ISM	2,304	604,794	< 1.5	979,695	30,242.0000	30,058.7930	0.61	0	15
						30,242.0000	30,102.7556	0.46	3,758,532,600	723,167
1.3	HLRN III	24,576	86,336	< 1.7	8,811,512	30,242.0000	30,102.6645	0.46	0	35
						30,242.0000	30,116.3592	0.42	2,402,406,311	575,678
1.4	HLRN III	12,288	43,199	< 1.5	1,709,027	30,242.0000	30,115.3331	0.42	0	3,709
						30,242.0000	30,120.4801	0.40	664,909,985	602,323
1.5	HLRN III	12,288	118,259	1.5	9,158,920	30,242.0000	30,120.4801	0.40	0	285
						30,242.0000	30,242.0000	0.00	1,677,724,126	0

Table 6.1: Statistics for solving `hc9p` on supercomputers.

incumbent solution. Notably, run 2.2 used 43 000 thousand CPU cores, with an idle time of less than 5 %. Still, the integrality gaps seem to suggest that long small-scale runs are more efficient than short large-scale ones.

Run	Computer	Cores	Time (sec.)	Idle (%)	Trans.	Primal bound (Upper bound)	Dual bound (Lower bound)	Gap (%)	Nodes	Open nodes
1	ISM	72	604,799 (2,558)	< 0.3	71	119,492.0000	117,388.8528	1.79	0	0
						119,297.0000	117,496.5470	1.53	4,314,198	1,109,629
2.1	HLRN III	12,288	43,149 (7,164)	< 0.5	31,304	119,297.0000	117,388.7971	1.63	0	0
						119,297.0000	117,426.2226	1.59	28,491,470	5,433,482
2.2	HLRN III	43,000	86,354	< 4.9	86,152	119,297.0000	117,426.2226	1.59	0	103
						119,297.0000	117,468.8459	1.56	267,513,609	40,499,188

Table 6.2: Statistics for solving `hc11p` on supercomputers.

The numbers of transferred B&B nodes are very small compared to those for `hc9p`. This indicates that `hc11p` is much harder than `hc9p` for our solver. Here, the aggressive use of cutting planes and further algorithms by SCIP-Jack at the root node is also problematic.

Name	gap [%]	new UB	previous UB
bip52u	**opt**	**233**	234
hc9p	**opt**	**30 242**	30 242
hc10p	0.7	59 733	59 797
hc11p	1.6	119 297	119 492
i640-311	0.6	35 765	35 766

Table 6.3: Improvements on unsolved SPG benchmark instances.

Finally, we note that it seems likely that several more of the open PUC and I640 instances could be solved to optimality by using supercomputer resources comparable to those employed for the computational experiments above. In particular, the optimal solution of the five open I640 instances appears to be well within reach. However, some PUC instances, such as `hc11p`, seem to require further algorithmic improvements.

Chapter 7

Conclusion and outlook

This thesis has set about to advance the state of the art in solving SPGs to optimality. Furthermore, this thesis has aimed to combine SPG advancements with improvements in the exact solution of related problems. Two well-known SPG relatives have been given special attention: The prize-collecting Steiner tree problem, and the maximum-weight connected subgraph problem. Furthermore, this thesis has shown how to extend the new algorithms and techniques to solve 12 further related problem classes.

To significantly advance the state of the art, many new techniques and algorithms had to be devised. The underlying policy to move from theory to practice has resulted not only in theoretical analyses of the utilized techniques and algorithms, but has also led to independent results for example in polyhedral descriptions. The various new algorithms have been combined in an intricate implementation of more than 110 thousand lines of source code—and with parallelization extensions. The new algorithms span almost the entire spectrum of a general branch-and-cut framework: From preprocessing and probing, to (M)IP formulations and separation methods, to (primal and dual) heuristics, domain propagation, and branching.

In this way, this thesis succeeds in pushing the limits of computational tractability not only for the classic SPG, but also for the 14 additionally considered, related problems. The newly developed Steiner tree solver SCIP-Jack is able to solve 57 previously intractable benchmark instances from the literature to optimality—around half of the previously unsolved benchmark instances considered in this thesis. The newly solved instances are from seven different problem classes, including SPG, MWCSP, and PCSTP. Several of these instances contain millions of edges, and some had remained unsolved for more than 20 years. For all 15 problem classes SCIP-Jack significantly outperforms, to the best of the author's knowledge, all other solvers described in the literature. Perhaps most importantly, SCIP-Jack outperforms the long-reigning state-of-the-art SPG solver both in terms of run-time and number of solved instances. Even for the rectilinear and Euclidean Steiner tree problems SCIP-Jack outperforms the specialized, well-known GeoSteiner solver—by simply solving the SPG obtained from the union of the full Steiner trees from the generation phase of GeoSteiner. For example, several Euclidean Steiner tree instances that could previously not be solved even after one week of computation are solved by SCIP-Jack within three minutes.

The newly developed SCIP-Jack solver will be made freely available for academic use as part of the SCIP Optimization Suite. Previous versions of SCIP-Jack have already seen notable use for research purposes, and the latest version of SCIP-Jack is currently being applied in several industrial projects.

The future

> Though much is taken, much abides (...)
>
> Lord Alfred Tennyson

50 years after its inception, the SPG continues to attract researchers from mathematics, computer science, and operations research. Much the same can be said of the many SPG relatives described in the literature (and partly in this thesis). With new applications being regularly discovered, Steiner tree problems can also claim a strong interest from practitioners in many disciplines—for example in bioinformatics, or recently machine learning.

As to theoretical advancements, several important questions remain unanswered regarding the strength of different IP and MIP formulations for SPG and related problems. We provide some major points below.

- Although the widely used bidirected cut formulation ($BDCut$) shows a very strong practical performance, finding an upper bound better than 2 on its integrality gap remains a well-known open problem. Similarly, the question on the best lower bounds is quite intriguing, with the currently best result being given in Byrka et al. (2013).
- This thesis has provided improved theoretical results for the strength of the well-known flow-balance bidirected cut formulation $BDCut_{FB}$ (which improves $BDCut$), but there exists an even stronger hierarchy of formulations based on $BDCut_{FB}$ due to Polzin and Daneshmand (2001b). Further theoretical studies of this hierarchy are still missing, however. See also Filipecki and Van Vyve (2020) for another hierarchy of SPG formulations.
- Another interesting topic is the relation of general-purpose cutting planes, such as Gomory cuts, to classic MIP formulations for Steiner trees. For the undirected cut formulation, Gaul and Schmidt (2021) recently introduced such results. However, results for the practically and theoretically much stronger $BDCut$ formulation are still to be established.

As to more practical advancements, the author of this thesis also sees several promising ways forward. Certainly, with every further algorithmic improvement, and with the ever-increasing intricacy of state-of-the-art SPG solvers, achieving a substantial further improvement might appear a daunting task. Still, the author of this thesis believes that significant further performance improvements are well within reach. First of all, several of the newly introduced SPG techniques could be transferred to

PCSTP and MWCSP. An example are the powerful extended reduction techniques described in Section 2.4. For many of the other related problem classes more shortly covered in this thesis there is even more room for improvement. Additionally, major points for future development are as follows.

- Further improvements of state-of-the-art reduction techniques, which have proven an indispensible tool for fast exact solution of SPG and related problems, seem highly promising. An example are better approximations of the newly introduced (but $\mathcal{NP}$-hard) implied bottleneck Steiner distance.

- Further practical improvements might be possible by using IP formulations that are stronger than $BDCut_{FB}$. One candidate is the hierarchy from Polzin and Daneshmand (2001b) mentioned above. Indeed, a (very) restricted version of this hierarchy is already used in the solver from Polzin (2003); Vahdati Daneshmand (2004). Similarly, a better integration of general-purpose cuts, whose default generation by SCIP is currently prohibitively slow, might lead to further speed-ups.

- Another interesting venue are specialized algorithms for still unsolved benchmark instances, in particular from the PUC test-set. These instances typically show a special structure, such as being bipartite or highly symmetric (hypercube graphs). A notable approach in this direction is given in Fischetti et al. (2017), although they still solve fewer PUC instances than SCIP-Jack. Combining such algorithms with the distributed-memory parallelization framework described in this thesis might make it possible to solve significantly more of the remaining PUC instances.

- The solution of Euclidean and rectilinear Steiner tree problems could be further improved by incorporating information about the full Steiner trees that are used within the concatenation phase. At the moment, we treat the union of these full Steiner trees as a customary SPG instance. A natural idea would for example be to branch not on single vertices, but rather on the full Steiner trees. However, in this case one would need to retain sufficient information during preprocessing.

- Finally, there is considerable potential in (further) shared-memory parallelization of several of the key algorithms of this thesis. The implementations for this thesis have been mostly for proof of concept. In particular, for large problems with millions of edges, a strong speed-up with multiple threads seems possible even for instances that do not require any branching.

The author of this thesis hopes that the free availability of SCIP-Jack for academic purposes (which contrasts the fully proprietary nature of the previous leading SPG solver) will facilitate further algorithmic advancements. In particular so, the possibility to use the powerful reduction techniques included in SCIP-Jack for preprocessing. Indeed, previous versions of these reduction techniques have already been used as a basis for other algorithms, see Iwata and Shigemura (2019). Finally, the author hopes that the availability of SCIP-Jack will continue to foster the successful use of Steiner tree and related problems in real-world applications.

List of Abbreviations and Names

See Table 1 for a list of Steiner tree problem types and their abbreviations.

BFS . . . . . . Breadth-first-search

CPLEX . . . Optimization software for LPs, MIPs, and (MI)QPs

DFS . . . . . . Depth-first-search

FiberSCIP . Shared-memory parallelization extension of SCIP

IP . . . . Integer program/programming

LP . . . . Linear program/programming

MIP . . . . . . Mixed-integer (linear) program/programming

MST . . . . . Minimum spanning tree

ParaSCIP . . Distributed-memory parallelization extension of SCIP

SCIP . . . . . Optimization software for MIPs, and for more general problems

SCIP-Jack . Optimization software for Steiner tree and related problems (developed as part of this thesis)

SoPlex . . . Optimization software for LPs

UG . . . . . . Framework to parallelize branch-and-bound based optimization software

Bibliography

The page numbers in brackets at the end of each citation refer to the text.

T. Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4 (1):4–20, 2007a. doi: 10.1016/j.disopt.2006.10.006. [42]

T. Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universität Berlin, 2007b. [12, 57]

T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. doi: 10.1007/s12532-008-0001-1. [167]

M. Akhmedov, I. Kwee, and R. Montemanni. A divide and conquer matheuristic algorithm for the Prize-collecting Steiner Tree Problem. *Computers & Operations Research*, 70:18 – 25, 2016. doi: 10.1016/j.cor.2015.12.015. [116]

M. Akhmedov, L. Galbusera, R. Montemanni, F. Bertoni, and I. Kwee. A prize-collecting Steiner tree application for signature selection to stratify diffuse large b-cell lymphoma subtypes. *bioRxiv*, page 272294, 2018. [116]

N. Alcaraz, J. Pauling, R. Batra, E. Barbosa, A. Junge, A. G. Christensen, V. Azevedo, H. J. Ditzel, and J. Baumbach. Keypathwayminer 4.0: condition-specific pathway analysis by combining multiple omics studies and networks with cytoscape. *BMC Systems Biology*, 8 (1):99, Aug 2014. doi: 10.1186/s12918-014-0099-x. [69]

E. Althaus and M. Blumenstock. Algorithms for the Maximum Weight Connected Subgraph and Prize-collecting Steiner Tree Problems. Unpublished manuscript at http://dimacs11.cs.princeton.edu/workshop.html, 2014. [85, 111]

E. Álvarez-Miranda, I. Ljubić, and P. Mutzel. The maximum weight connected subgraph problem. In *Facets of Combinatorial Optimization*, pages 245–270. Springer Berlin Heidelberg, 2013a. doi: 10.1007/978-3-642-38189-8_11. [69, 70, 77, 83, 105, and 109]

E. Álvarez-Miranda, I. Ljubić, and P. Mutzel. *The Rooted Maximum Node-Weight Connected Subgraph Problem*, pages 300–315. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013b. ISBN 978-3-642-38171-3. doi: 10.1007/978-3-642-38171-3_20. [70, 72, 73, 208, and 209]

Y. P. Aneja. An integer linear programming approach to the Steiner problem in graphs. *Networks*, 10(2):167–178, 1980. doi: 10.1002/net.3230100207. [19]

D. L. Applegate, R. E. Bixby, V. Chvatál, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, 2006. ISBN 9780691129938. [19]

A. Archer, M. Bateni, M. Hajiaghayi, and H. Karloff. Improved approximation algorithms for prize-collecting Steiner tree and TSP. *SIAM Journal on Computing*, 40(2):309–332, 2011. doi: 10.1137/090771429. [116]

C. Backes, A. Rurainski, G. Klau, O. Müller, D. Stöckel, A. Gerasch, J. Küntzer, D. Maisel, N. Ludwig, M. Hein, A. Keller, H. Burtscher, M. Kaufmann, E. Meese, and H.-P. Lenhof. An integer linear programming approach for finding deregulated subgraphs in regulatory networks. *Nucleic Acids Res*, 40(6):e43, 2011. doi: 10.1093/nar/gkr1227. [70, 161]

M. Ball, W. Liu, and W. Pulleyblank. Two terminal Steiner tree polyhedra. In *Contributions to Operations Research and Economics – The Twentieth Anniversary of CORE*, pages 251–284, Cambridge, MA, 1989. MIT Press. [21]

J. Banfi. *Multirobot exploration of communication-restricted environments*. PhD thesis, Politecnico di Milano, 2018. [69]

M. Bateni, C. Chekuri, A. Ene, M. Hajiaghayi, N. Korula, and D. Marx. Prize-collecting Steiner problems on planar graphs. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011*, pages 1028–1049, 2011. doi: 10.1137/1.9781611973082.79. [116]

J. Beasley. An SST-based algorithm for the Steiner problem in graphs. *Networks*, 19:1–16, 1989. doi: 10.1002/net.3230190102. [16]

J. E. Beasley. An algorithm for the Steiner problem in graphs. *Networks*, 14(1):147–159, 1984. doi: 10.1002/net.3230140112. [16]

M. A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, and P. Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005. doi: 10.1016/j.jalgor.2005.08.001. [172]

P. Berman and V. Ramaiyer. Improved approximations for the Steiner tree problem. *Journal of Algorithms*, 17(3):381–408, 1994. doi: 10.1006/jagm.1994.1041. [16]

M. W. Bern and P. E. Plassmann. The Steiner Problem with Edge Lengths 1 and 2. *Information Processing Letters*, 32(4):171–176, 1989. doi: 10.1016/0020-0190(89)90039-2. [15]

S. Beyer and M. Chimani. The influence of preprocessing on steiner tree approximations. In Z. Lu, D. Kim, W. Wu, W. Li, and D.-Z. Du, editors, *Combinatorial Optimization and Applications*, pages 601–616, Cham, 2015. Springer International Publishing. doi: 10.1007/978-3-319-26626-8_44. [26]

S. Beyer and M. Chimani. Strong Steiner tree approximations in practice. *J. Exp. Algorithmics*, 24(1):1.7:1–1.7:33, Jan. 2019. doi: 10.1145/3299903. [16]

M. Bezenšek and B. Robič. A survey of parallel and distributed algorithms for the steiner tree problem. *International Journal of Parallel Programming*, 42(2):287–319, 2014. doi: 10.1007/s10766-013-0243-z. [178]

I. Biazzo, A. Braunstein, and R. Zecchina. Performance of a cavity-method-based algorithm for the prize-collecting Steiner tree problem on graphs. *Phys. Rev. E*, 86:026706, Aug 2012. doi: 10.1103/PhysRevE.86.026706. [146]

D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. P. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, 59:413–420, 1993. doi: 10.1007/BF01581256. [116]

M. D. Biha, H. L. M. Kerivin, and P. H. Ng. Polyhedral study of the connected subgraph problem. *Discrete Mathematics*, 338(1):80–92, 2015. doi: 10.1016/j.disc.2014.08.026. [70]

G. Bolukbasi and A. S. Kocaman. A prize collecting Steiner tree approach to least cost evaluation of grid and off-grid electrification systems. *Energy*, 160:536 – 543, 2018. doi: 10.1016/j.energy.2018.07.029. [116]

J. A. Bondy and U. S. R. Murty. *Graph Theory*. Springer London, 2008. doi: 10.1007/978-1-84628-970-5. [9]

É. Bonnet and F. Sikora. The PACE 2018 Parameterized Algorithms and Computational Experiments Challenge: The Third Iteration. In C. Paul and M. Pilipczuk, editors, *13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi: 10.4230/LIPIcs.IPEC.2018.26. [7, 18, 62, and 168]

G. Borradaile, P. Klein, and C. Mathieu. An O(n log n) approximation scheme for Steiner tree in planar graphs. *ACM Transactions on Algorithms (TALG)*, 5(3):1–31, 2009. doi: 10.1145/1541885.1541892. [16]

A. Braunstein and A. Muntoni. Practical optimization of Steiner trees via the cavity method. *Journal of Statistical Mechanics: Theory and Experiment*, 2016(7):073302, jul 2016. doi: 10.1088/1742-5468/2016/07/073302. [116, 148]

M. Brazil and M. Zachariasen. *Optimal interconnection trees in the plane: theory, algorithms and applications*. Algorithms and Combinatorics. Springer, 2015. ISBN 978-3-319-13914-2. [157]

M. Brazil, R. L. Graham, D. A. Thomas, and M. Zachariasen. On the History of the Euclidean Steiner Tree Problem. *Archive for History of Exact Sciences*, 68:327–354, 2014. doi: 10.1007/s00407-013-0127-z. [1, 14]

R. E. Bryant and D. R. O'Hallaron. *Computer Systems: A Programmer's Perspective*. Addison-Wesley Publishing Company, USA, 3rd edition, 2015. [167]

A. Buchanan, J. S. Sung, S. Butenko, and E. L. Pasiliao. An integer programming approach for fault-tolerant connected dominating sets. *INFORMS Journal on Computing*, 27(1): 178–188, 2015. doi: 10.1287/ijoc.2014.0619. [70]

A. Buchanan, Y. Wang, and S. Butenko. Algorithms for node-weighted Steiner tree and maximum-weight connected subgraph. *Networks*, 72(2):238–248, 2018. doi: 10.1002/net.21825. [15, 70, 120, 121, 122, 156, and 209]

O. Burdakov, P. Doherty, and J. Kvarnström. Local Search for Hop-constrained Directed Steiner Tree Problem with Application to UAV-based Multi-target Surveillance. In Butenko, S., Pasiliao, E.L., Shylo, and V., editors, *Examining Robustness and Vulnerability of Networked Systems*, volume 37 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 26–50. IOS Press, 2014. doi: 10.3233/978-1-61499-391-9-26. [164, 165]

J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. Steiner Tree Approximation via Iterative Randomized Rounding. *The Journal of the ACM*, 60(1):6, 2013. doi: 10.1145/2432622.2432628. [16, 19, 157, and 184]

H. Cambazard and N. Catusse. Fixed-parameter algorithms for rectilinear Steiner tree and rectilinear traveling salesman problem in the plane. *European Journal of Operational Research*, 270(2):419–429, 2018. doi: 10.1016/j.ejor.2018.03.042. [157]

S. A. Canuto, M. G. C. Resende, and C. C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 38(1):50–58, 2001. doi: 10.1002/net.1023. [116]

A. Caprara and M. Fischetti. {0, 1/2}-chvátal-gomory cuts. *Mathematical Programming*, 74(3):221–235, 1996. doi: 10.1007/BF02592196. [169]

R. Carvajal, M. Constantino, M. Goycoolea, J. P. Vielma, and A. Weintraub. Imposing connectivity constraints in forest planning models. *Operations Research*, 61(4):824–836, 2013. doi: 10.1287/opre.2013.1183. [70]

M. Charikar, C. Chekuri, T.-y. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation Algorithms for Directed Steiner Problems. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pages 192–200, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics. ISBN 0-89871-410-9. [154]

C. Chen and K. Grauman. Efficient activity detection with max-subgraph search. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 1274–1281, 2012. doi: 10.1109/CVPR.2012.6247811. [69]

Y. H. Chen. The bottleneck selected-internal and partial terminal Steiner tree problems. *Networks*, 68(4):331–339, 2016. doi: 10.1002/net.21713. [152]

X. Cheng and D.-Z. Du. *Steiner trees in industry*, volume 11. Springer Science & Business Media, 2004. doi: 10.1007/0-387-23830-1_4. [14]

B. V. Cherkassky and A. V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997. doi: 10.1007/PL00009180. [178]

M. Chimani, P. Mutzel, and B. Zey. Improved Steiner tree algorithms for bounded treewidth. *Journal of Discrete Algorithms*, 16:67 – 78, 2012. doi: 10.1016/j.jda.2012.04.016. Selected papers from the 22nd International Workshop on Combinatorial Algorithms (IWOCA 2011). [15, 54]

F. Chin and D. Houck. Algorithms for updating minimal spanning trees. *Journal of Computer and System Sciences*, 16(3):333–344, 1978. doi: 10.1016/0022-0000(78)90022-3. [175, 176]

M. Chlebík and J. Chlebíková. The Steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science*, 406(3):207–214, 2008. doi: 10.1016/j.tcs.2008.06.046. [15]

S. Chopra and M. Rao. The Steiner tree problem I: Formulations, compositions and extension of facets. *Mathematical Programming*, 64(2):209–229, 1994. doi: 10.1007/bf01582573. [20]

S. Chopra, E. R. Gorres, and M. Rao. Solving the Steiner tree problem on a graph using branch and cut. *ORSA Journal on Computing*, 4(3):320–335, 1992. doi: 10.1287/ijoc.4.3.320. [16]

S. A. Chowdhury, S. Shackney, K. Heselmeyer-Haddad, T. Ried, A. A. Schäffer, and R. Schwartz. Phylogenetic analysis of multiprobe fluorescence in situ hybridization data from tumor cell populations. *Bioinformatics*, 29(13):189–198, 2013. doi: 10.1093/bioinformatics/btt205. [157]

K. Ciebiera, P. Godlewski, P. Sankowski, and P. Wygocki. Approximation Algorithms for Steiner Tree Problems Based on Universal Solution Frameworks. *Computing Research Repository*, abs/1410.7534, 2014. URL `http://arxiv.org/abs/1410.7534`. [16]

D. Cieslik. *Steiner minimal trees*, volume 23. Springer, 1998. doi: 10.1007/978-1-4757-6585-4. [14]

J. Conrad, C. P. Gomes, W.-J. van Hoeve, A. Sabharwal, and J. Suter. Connections in networks: Hardness of feasibility versus optimality. In P. Van Hentenryck and L. Wolsey, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 16–28, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-72397-4. [73]

A. Costa, J.-F. Cordeau, and G. Laporte. Steiner Tree Problems With Profits. *INFOR: Information Systems and Operational Research*, 44(2):99–115, 2006. doi: 10.1080/03155986.2006.11732743. [116]

H. Courant, C. Courant, R. Courant, H. Robbins, I. Stewart, P. Stewart, and P. Robbins. *What is Mathematics?: An Elementary Approach to Ideas and Methods*. Oxford Paperbacks. Oxford University Press, 1941. [14]

A. S. da Cunha, A. Lucena, N. Maculan, and M. G. Resende. A relax-and-cut algorithm for the prize-collecting Steiner problem in graphs. *Discrete Applied Mathematics*, 157 (6):1198 – 1217, 2009. doi: 10.1016/j.dam.2008.02.014. Reformulation Techniques and Mathematical Programming. [116]

T. A. Davis. *Direct methods for sparse linear systems*. SIAM, 2006. doi: 10.1137/1.9780898718881. [169]

M. P. de Aragão and R. F. Werneck. On the Implementation of MST-based Heuristics for the Steiner Problem in Graphs. In *Proceedings of the 4th International Workshop on Algorithm Engineering and Experiments*, pages 1–15. Springer, 2002. doi: 10.1007/3-540-45643-0_1. [51, 106]

M. P. de Aragão, E. Uchoa, and R. F. Werneck. Dual heuristics on the exact solution of large Steiner problems. *Electronic Notes in Discrete Mathematics*, 7:150 – 153, 2001. doi: 10.1016/S1571-0653(04)00247-1. Brazilian Symposium on Graphs, Algorithms and Combinatorics. [17]

B. R. de Supinski, T. R. W. Scogland, A. Duran, M. Klemm, S. M. Bellido, S. L. Olivier, C. Terboven, and T. G. Mattson. The Ongoing Evolution of OpenMP. *Proceedings of the IEEE*, 106(11):2004–2019, Nov 2018. doi: 10.1109/JPROC.2018.2853600. [178]

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1): 269–271, Dec. 1959. doi: 10.1007/BF01386390. [125]

B. Dilkina and C. P. Gomes. Solving connected subgraph problems in wildlife conservation. In *Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, CPAIOR'10, pages 102–116, Berlin, Heidelberg, 2010. Springer-Verlag. doi: 10.1007/978-3-642-13520-0_14. [69, 73]

DIMACS. 11th DIMACS Challenge. `http://dimacs11.zib.de/`, 2015. Accessed: January 10. 2020. [25, 64, 109, 116, 146, 148, and 168]

M. Dittrich, G. Klau, A. Rosenwald, T. Dandekar, and T. Müller. Identifying functional modules in protein-protein interaction networks: An integrated exact approach. *Bioinformatics (Oxford, England)*, 24:i223–31, 08 2008. doi: 10.1093/bioinformatics/btn161. [69, 105, 109, 121, and 146]

M. Dom, D. Lokshtanov, and S. Saurabh. Kernelization Lower Bounds Through Colors and IDs. *ACM Transactions on Algorithms*, 11(2), Oct. 2014. doi: 10.1145/2650261. [15]

S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971. doi: 10.1002/net.3230010302. [15, 16, 17, 27, 54, 118, and 209]

L. M. A. Drummond, M. Santos, and E. Uchoa. A distributed dual ascent algorithm for Steiner problems in multicast routing. *Networks*, 53(2):170–183, 2009. doi: 10.1002/net.20276. [179]

D.-Z. Du, J. Smith, and J. H. Rubinstein. *Advances in Steiner trees*, volume 6. Springer, 2000. doi: 10.1007/978-1-4757-3171-2. [14]

C. Duin. *Steiner Problems in Graphs*. PhD thesis, University of Amsterdam, 1993. [16, 17, 20, 21, 25, 26, 33, 35, 38, 39, 71, 86, 91, 100, 136, and 168]

C. Duin. *Preprocessing the Steiner Problem in Graphs*, pages 175–233. Springer US, Boston, MA, 2000. doi: 10.1007/978-1-4757-3171-2_10. [43, 173]

C. Duin and A. Volgenant. An edge elimination test for the Steiner problem in graphs. *Operations Research Letters*, 8(2):79 – 83, 1989a. doi: 10.1016/0167-6377(89)90005-9. [16, 27, and 128]

C. Duin and S. Voss. Efficient path and vertex exchange in Steiner tree algorithms. *Networks*, 29(2):89–105, 1997. doi: 10.1002/(sici)1097-0037(199703)29:2⟨89::aid-net3⟩3.0.co;2-7. [16, 51, and 171]

C. W. Duin and A. Volgenant. Reduction tests for the Steiner problem in graphs. *Networks*, 19(5):549–567, 1989b. doi: 10.1002/net.3230190506. [16, 32, 34, 35, 55, 130, and 171]

C. W. Duin, A. Volgenant, and S. Voss. Solving group Steiner problems as Steiner problems. *European Journal of Operational Research*, 154(1):323–329, 2004. doi: 10.1016/s0377-2217(02)00707-5. [163]

C. El-Arbi. Une heuristique pour le problème de l'arbre de Steiner. *RAIRO-Operations Research*, 12(2):207–212, 1978. [16]

M. El-Kebir and G. W. Klau. Solving the Maximum-Weight Connected Subgraph Problem to Optimality. *Computing Research Repository*, abs/1409.5308, 2014. [85, 86, 93, 94, 105, and 111]

J. M. Elble. *Computational experience with linear optimization and related problems*. PhD thesis, University of Illinois at Urbana-Champaign, 2010. [170]

N. Emanet. *The Rectilinear Steiner Tree Problem*. Lambert Academic Publishing, 2010. [157]

R. E. Erickson, C. L. Monma, and A. F. Veinott. Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research*, 12(4):634–664, 1987. doi: 10.1287/moor.12.4.634. [15, 18, and 54]

J. Feigenbaum, C. H. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1):21 – 41, 2001. doi: 10.1006/jcss.2001.1754. [115]

A. E. Feldmann, S. Karthik C., E. Lee, and P. Manurangsi. A Survey on Approximation in Parameterized Complexity: Hardness and Algorithms. *arXiv e-prints*, art. arXiv:2006.04411, June 2020. [16]

P. Feofiloff, C. G. Fernandes, C. E. Ferreira, and J. C. de Pina. Primal-dual approximation algorithms for the prize-collecting Steiner tree problem. *Inf. Process. Lett.*, 103(5):195–202, Aug. 2007. doi: 10.1016/j.ipl.2007.03.012. [116]

C. E. Ferreira and F. M. de Oliveira Filho. New reduction techniques for the group steiner tree problem. *SIAM Journal on Optimization*, 17(4):1176–1188, 2007. doi: 10.1137/040610891. [163]

J. K. Fichte, M. Hecher, and A. Schidler. Solving the Steiner Tree Problem with few Terminals. *arXiv preprint arXiv:2011.04593*, 2020. [18]

B. Filipecki and M. Van Vyve. Stronger path-based extended formulation for the Steiner tree problem. *Networks*, 75(1):3–17, 2020. doi: 10.1002/net.21901. [25, 184]

J. Fischer and V. Heun. Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In *Annual Symposium on Combinatorial Pattern Matching*, pages 36–48. Springer, 2006. [172]

M. Fischetti, M. Leitner, I. Ljubić, M. Luipersbeck, M. Monaci, M. Resch, D. Salvagnin, and M. Sinnl. Thinning out Steiner trees: a node-based model for uniform edge costs. *Mathematical Programming Computation*, 9(2):203–229, Jun 2017. doi: 10.1007/s12532-016-0111-0. [17, 20, 23, 25, 26, 66, 67, 70, 78, 111, 116, 117, 145, 148, 159, 160, and 185]

R. Floren. A note on "a faster approximation algorithm for the steiner problem in graphs". *Information Processing Letters*, 38(4):177 – 178, 1991. doi: 10.1016/0020-0190(91)90096-Z. [171]

F. V. Fomin, F. Grandoni, D. Kratsch, D. Lokshtanov, and S. Saurabh. Computing optimal Steiner trees in polynomial space. *Algorithmica*, 65(3):584–604, 2013. doi: 10.1002/net.3230200606. [15]

F. V. Fomin, P. Kaski, D. Lokshtanov, F. Panolan, and S. Saurabh. Parameterized single-exponential time polynomial space algorithm for steiner tree. *SIAM Journal on Discrete Mathematics*, 33(1):327–345, 2019a. doi: 10.1137/17M1140030. [15]

F. V. Fomin, D. Lokshtanov, S. Saurabh, and M. Zehavi. *Kernelization: theory of parameterized preprocessing.* Cambridge University Press, 2019b. [2]

F. V. Fomin, D. Lokshtanov, S. Kolay, F. Panolan, and S. Saurabh. Subexponential algorithms for rectilinear Steiner tree and arborescence problems. *ACM Transactions on Algorithms (TALG)*, 16(2):1–37, 2020. [157]

H. Franz. Connected subgraphs with budget constraints: formulations and algorithms. Master's thesis, TU Berlin, 2019. [161]

Z. Fu and J. Hao. Knowledge-guided local search for the prize-collecting Steiner tree problem in graphs. *Knowl.-Based Syst.*, 128:78–92, 2017a. doi: 10.1016/j.knosys.2017.04.010. [116, 145, and 148]

Z. Fu and J.-K. Hao. Swap-vertex based neighborhood for Steiner tree problems. *Mathematical Programming Computation*, 9:297–320, 2017b. doi: 10.1007/s12532-016-0116-8. [105]

B. Fuchs, W. Kern, D. Molle, S. Richter, P. Rossmanith, and X. Wang. Dynamic programming for minimum Steiner trees. *Theory of Computing Systems*, 41(3):493–500, 2007a. doi: 10.1007/s00224-007-1324-4. [15]

B. Fuchs, W. Kern, and X. Wang. Speeding up the dreyfus–wagner algorithm for minimum Steiner trees. *Mathematical methods of operations research*, 66(1):117–125, 2007b. doi: 10.1007/s00186-007-0146-0. [15]

A. Fügenschuh and M. Fügenschuh. Integer linear programming models for topology optimization in sheet metal design. *Mathematical Methods of Operations Research*, 68(2):313 – 331, 2008. [72]

G. Gamrath, T. Koch, S. Maher, D. Rehfeldt, and Y. Shinano. SCIP-Jack - A solver for STP and variants with parallelization extensions. *Mathematical Programming Computation*, 9 (2):231 – 296, 2017. doi: 10.1007/s12532-016-0114-x. [17, 116, 142, 148, and 165]

G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, G. Hendel, C. Hojny, T. Koch, P. L. Bodic, S. J. Maher, F. Matter, M. Miltenberger, E. Mühmer, B. Müller, M. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, C. Tawfik, S. Vigerske, F. Wegscheider, D. Weninger, and J. Witzig. The scip optimization suite 7.0. Technical Report 20-10, ZIB, Takustr. 7, 14195 Berlin, 2020. [7, 11, and 167]

M. Garey and D. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977. doi: 10.1137/0132071. [157]

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, 1979. ISBN 0716710447. [37, 90, 97, and 120]

R. S. Garfinkel and G. L. Nemhauser. Optimal political districting by implicit enumeration techniques. *Management Science*, 16(8):B–495, 1970. doi: 10.1287/mnsc.16.8.b495. [70]

D. Gaul and D. R. Schmidt. Chvátal–gomory cuts for the steiner tree problem. *Discrete Applied Mathematics*, 291:188–200, 2021. doi: 10.1016/j.dam.2020.12.016. [184]

E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968. doi: 10.1137/0116001. [16]

A. Gionis, M. Mathioudakis, and A. Ukkonen. Bump hunting in the dark: Local discrepancy maximization on graphs. *IEEE Transactions on Knowledge and Data Engineering*, 29(3): 529–542, 2017. doi: 10.1109/icde.2015.7113364. [116]

A. Gleixner, M. Bastubbe, L. Eifler, T. Gally, G. Gamrath, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. E. Lübbecke, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, C. Schubert, F. Serrano, Y. Shinano, J. M. Viernickel, M. Walter, F. Wegscheider, J. T. Witt, and J. Witzig. The scip optimization suite 6.0. Technical Report 18-26, ZIB, Takustr. 7, 14195 Berlin, 2018. [11, 12]

M. X. Goemans and Y. Myung. A catalog of Steiner tree formulations. *Networks*, 23(1): 19–28, 1993. doi: 10.1002/net.3230230104. [19, 20, 72, 78, and 142]

M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, Apr. 1995. doi: 10.1137/S0097539793242618. [115, 116]

M. X. Goemans, N. Olver, T. Rothvoß, and R. Zenklusen. Matroids and Integrality Gaps for Hypergraphic Steiner Tree Relaxations. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, page 1161–1176, New York, NY, USA, 2012. Association for Computing Machinery. doi: 10.1145/2213977.2214081. [16]

M. Grötschel and C. L. Monma. Integer Polyhedra Arising from Certain Network Design Problems with Connectivity Constraints. *SIAM Journal on Discrete Mathematics*, 3(4): 502–523, 1990. doi: 10.1137/0403043. [20]

S. Guha and S. Khuller. Improved Methods for Approximating Node Weighted Steiner Trees and Connected Dominating Sets. *Information and Computation*, 150(1):57–74, 1999. doi: 10.1006/inco.1998.2754. [156]

S. L. Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1(2):113–133, 1971. doi: 10.1002/net.3230010203. [14, 15, and 120]

E. Halperin and R. Krauthgamer. Polylogarithmic inapproximability. In L. L. Larmore and M. X. Goemans, editors, *STOC*, pages 585–594. ACM, 2003. ISBN 1-58113-674-9. doi: 10.1145/780542.780628. [154]

M. Hanan. On Steiner's problem with rectilinear distance. *SIAM Journal of Applied Mathematics*, 14(2):255–265, 1966. doi: 10.1137/0114025. [157]

J. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a graph. In G. N. Frederickson, editor, *Proceedings of the Third Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 27-29 January 1992, Orlando, Florida, USA*, pages 165–174. ACM/SIAM, 1992. [177]

C. Hegde, P. Indyk, and L. Schmidt. A fast, adaptive variant of the goemans-williamson scheme for the prize-collecting Steiner tree problem. In *Workshop of the 11th DIMACS Implementation Challenge*. Workshop of the 11th DIMACS Implementation Challenge, 2014. [58, 109]

S. Held, B. Korte, D. Rautenbach, and J. Vygen. Combinatorial Optimization in VLSI Design. *Combinatorial Optimization-Methods and Applications*, 31:33–96, 2011. [14]

S. C. Hidayati, K. Hua, Y. Tsao, H. Shuai, J. Liu, and W. Cheng. Garment detectives: Discovering clothes and its genre in consumer photos. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 471–474, March 2019. doi: 10.1109/MIPR.2019.00095. [116]

S. Hougardy and H. J. Prömel. A 1.598 approximation algorithm for the Steiner problem in graphs. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 448–453, 1999. [16]

S. Hougardy, J. Silvanus, and J. Vygen. Dijkstra meets Steiner: a fast exact goal-oriented Steiner tree algorithm. *Mathematical Programming Computation*, 9(2):135–202, 2017. doi: 10.1007/s12532-016-0110-1. [17, 18, and 54]

S.-Y. Hsieh and H.-M. Gao. On the partial terminal Steiner tree problem. *The Journal of Supercomputing*, 41(1):41–52, Jul 2007. doi: 10.1007/s11227-007-0102-z. [152]

R. Hušek, D. Knop, and T. Masařík. Approximation algorithms for Steiner tree based on star contractions: A unified view. *arXiv preprint arXiv:2002.03583*, 2020. [18]

F. Hwang, D. Richards, and P. Winter. *The Steiner Tree Problem*. Annals of Discrete Mathematics. Elsevier Science, 1992. [1, 10, 14, 15, 39, 48, 51, 57, 86, 130, 157, 163, and 173]

F. K. Hwang and D. S. Richards. Steiner tree problems. *Networks*, 22(1):55–89, 1992. doi: 10.1002/net.3230220105. [16]

IBM. Cplex, 2020. URL `https://www.ibm.com/analytics/cplex-optimizer`. [11]

T. Ideker, O. Ozier, B. Schwikowski, and A. F. Siegel. Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics*, 18(1):S233–S240, 07 2002. doi: 10.1093/bioinformatics/18.suppl_1.s233. [116]

Y. Iwata and T. Shigemura. Separator-Based Pruned Dynamic Programming for Steiner Tree. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1520–1527, 2019. doi: 10.1609/aaai.v33i01.33011520. [7, 18, 54, and 185]

D. S. Johnson. The np-completeness column: An ongoing guide. *J. Algorithms*, 6(1):145–159, 1985. doi: 10.1016/0196-6774(85)90025-2. [69]

D. S. Johnson, M. Minkoff, and S. Phillips. The Prize Collecting Steiner Tree Problem: Theory and Practice. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 760–769, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics. ISBN 0-89871-453-2. [115, 116, 146, and 147]

J. Johnston, R. Kelley, T. Crawford, D. Morton, R. Agarwala, T. Koch, A. Schäffer, C. Francomano, and L. Biesecker. A novel nemaline myopathy in the Amish caused by a mutation in troponin T1. *American Journal of Human Genetics*, pages 814–821, October 2000. doi: 10.1002/mus.24528. [154]

D. Juhl, D. M. Warme, P. Winter, and M. Zachariasen. The GeoSteiner software package for computing Steiner trees in the plane: an updated computational study. *Mathematical Programming Computation*, 10(4):487–532, 2018. doi: 10.1007/s12532-018-0135-8. [58, 61, 66, 157, and 158]

R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. doi: 10.1007/978-1-4684-2001-2_9. [14]

M. Karpinski and A. Zelikovsky. New approximation algorithms for the Steiner tree problems. *Journal of Combinatorial Optimization*, 1(1):47–65, 1997. [16]

J. Kepner and J. Gilbert. *Graph algorithms in the language of linear algebra.* SIAM, 2011. doi: 10.1137/1.9780898719918. [169]

J. H. Kingston and N. P. Sheppard. On reductions for the Steiner Problem in Graphs. *Journal of Discrete Algorithms*, 1(1):77 – 88, 2003. doi: 10.1016/S1570-8667(03)00008-X. [12, 55]

S. Kisfaludi-Bak, J. Nederlof, and E. J. v. Leeuwen. Nearly eth-tight algorithms for planar steiner tree with terminals on few faces. *ACM Transactions on Algorithms*, 16(3), June 2020a. doi: 10.1145/3371389. [15]

S. Kisfaludi-Bak, J. Nederlof, and E. J. v. Leeuwen. Nearly ETH-tight algorithms for planar Steiner tree with terminals on few faces. *ACM Transactions on Algorithms (TALG)*, 16 (3):1–30, 2020b. doi: 10.1145/3371389. [15]

G. W. Klau, I. Ljubić, A. Moser, P. Mutzel, P. Neuner, U. Pferschy, G. Raidl, and R. Weiskircher. Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. In K. Deb, editor, *Genetic and Evolutionary Computation – GECCO 2004*, pages 1304–1315, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-24854-5. [116]

P. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted Steiner trees. *Journal of Algorithms*, 19(1):104 – 115, 1995. doi: 10.1006/jagm.1995.1029. [122]

F. Klimm, E. M. Toledo, T. Monfeuga, F. Zhang, C. M. Deane, and G. Reinert. Functional module detection through integration of single-cell rna sequencing data with protein–protein interaction networks. *BMC genomics*, 21(1):1–10, 2020. doi: 10.1186/s12864-020-07144-2. [69]

T. Koch. Jack-III: Ein Branch & Cut-Verfahren zur Lösung des gewichteten Steinerbaumproblems in Graphen. Master's thesis, Technische Universität Berlin, 1995. [169]

T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32: 207–232, 1998. doi: 10.1002/(SICI)1097-0037(199810)32:3\%3C207::AID-NET5\%3E3.0.CO;2-O. [16, 17, 20, 21, 26, 56, 58, 106, 167, 177, and 178]

T. Koch, A. Martin, and S. Voß. SteinLib: An updated library on Steiner tree problems in graphs. In D.-Z. Du and X. Cheng, editors, *Steiner Trees in Industries*, pages 285–325. Kluwer, 2001. ISBN 1-402-00099-5. [17, 22, 57, and 168]

T. Koch, A. Martin, and M. E. Pfetsch. Progress in Academic Computational Integer Programming. In M. Jünger and G. Reinelt, editors, *Facets of Combinatorial Optimization*, pages 483–506. Springer, 2013. doi: 10.1007/978-3-642-38189-8_19. [2]

B. Korte, L. Lovász, and R. Schrader. *Greedoids*, volume 4. Springer Science & Business Media, 2012. [76]

B. Korte, J. Vygen, B. Korte, and J. Vygen. *Combinatorial optimization*. Springer, 2018. [14]

L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta informatica*, 15(2):141–145, 1981. doi: 10.1007/bf00288961. [16]

L. T. Kou. On efficient implementation of an approximation algorithm for the Steiner tree problem. *Acta informatica*, 27(4):369–380, 1990. doi: 10.1007/bf00264613. [16]

M. Leitner, I. Ljubic, M. Luipersbeck, M. Prossegger, and M. Resch. New Real-world Instances for the Steiner Tree Problem in Graphs. Technical report, ISOR, Uni Wien, 2014. [14, 58]

M. Leitner, I. Ljubic, M. Luipersbeck, and M. Sinnl. A Dual Ascent-Based Branch-and-Bound Framework for the Prize-Collecting Steiner Tree and Related Problems. *INFORMS Journal on Computing*, 30(2):402–420, 2018a. doi: 10.1287/ijoc.2017.0788. [70, 85, 93, 100, 105, 109, 111, 116, 117, 122, 142, 145, 146, 147, 148, and 156]

M. Leitner, I. Ljubic, M. Luipersbeck, and M. Sinnl. Decomposition methods for the two-stage stochastic steiner tree problem. *Computational Optimization and Applications*, 69 (3):713–752, 2018b. doi: 10.1007/s10589-017-9966-x. [21]

A. Y. Levin. Algorithm for the shortest connection of a group of graph vertices. In *Doklady Akademii Nauk*, volume 200, pages 773–776. Russian Academy of Sciences, 1971. [15]

F. Liers, A. Martin, and S. Pape. Binary steiner trees: Structural results and an exact solution approach. *Discrete Optimization*, 21:85 – 117, 2016. doi: 10.1016/j.disopt.2016.05.006. [159]

W. Liu. A lower bound for the Steiner tree problem in directed graphs. *Networks*, 20(6): 765–778, 1990. doi: 10.1002/net.3230200606. [22]

I. Ljubic. *Exact and Memetic Algorithms for Two Network Design Problems*. PhD thesis, Vienna University of Technology, 2004. [116, 146, and 147]

I. Ljubic. Solving Steiner Trees — Recent Advances, Challenges and Perspectives. *Networks*, 2020. accepted for publication. [7, 14, 51, 116, and 178]

I. Ljubic, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti. An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem. *Mathematical Programming*, 105(2-3):427–449, 2006. doi: 10.1007/s10107-005-0660-x. [21, 116, 117, 122, 136, and 146]

A. A. Loboda, M. N. Artyomov, and A. A. Sergushichev. *Solving Generalized Maximum-Weight Connected Subgraph Problem for Network Enrichment Analysis*, pages 210–221. Springer International Publishing, Cham, 2016. doi: 10.1007/978-3-319-43681-4_17. [69, 86, 109, and 111]

A. Lodi and A. Tramontani. Performance variability in mixed-integer programming. In *Theory Driven by Influential Applications*, pages 1–12. INFORMS, 2013. doi: 10.1287/educ.2013.0112. [12]

C. L. Lu, C. Y. Tang, and R. C.-T. Lee. The full Steiner tree problem. *Theoretical Computer Science*, 306(1):55 – 67, 2003. doi: 10.1016/S0304-3975(03)00209-3. [152]

A. Lucena and J. E. Beasley. A branch and cut algorithm for the Steiner problem in graphs. *Networks*, 31(1):39–59, 1998. doi: 10.1002/(SICI)1097-0037(199801)31:1⟨39::AID-NET5⟩3.0.CO;2-L. [16]

T. L. Magnanti and L. A. Wolsey. *Handbooks in Operations Research and Management Science*, volume Volume 7, chapter Chapter 9 Optimal trees, pages 503–615. Elsevier, 1995. [19, 20, 73, and 106]

D. Marx, M. Pilipczuk, and M. Pilipczuk. On Subexponential Parameterized Algorithms for Steiner Tree and Directed Subset TSP on Planar Graphs. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 474–484, 2018. doi: 10.1109/FOCS.2018.00052. [15]

K. Mehlhorn. A Faster Approximation Algorithm for the Steiner Problem in Graphs. *Information Processing Letters*, 27(3):125–128, 1988. doi: 10.1016/0020-0190(88)90066-x. [16, 171]

Y.-F. Ming, S.-B. Chen, Y.-Q. Chen, and Z.-H. Fu. A fast vertex-swap operator for the prize-collecting Steiner tree problem. In Y. Shi, H. Fu, Y. Tian, V. V. Krzhizhanovskaya, M. H. Lees, J. Dongarra, and P. M. A. Sloot, editors, *Computational Science – ICCS 2018*, pages 553–560, Cham, 2018. Springer International Publishing. ISBN 978-3-319-93701-4. [116]

H. Mittelmann. Benchmarks for optimization software, 2020. `http://plato.asu.edu/bench.html`. [12]

J. Moody and D. R. White. Structural cohesion and embeddedness: A hierarchical concept of social groups. *American sociological review*, pages 103–127, 2003. [70]

A. Moss and Y. Rabani. Approximation algorithms for constrained for constrained node weighted Steiner tree problems. In J. S. Vitter, P. G. Spirakis, and M. Yannakakis, editors, *STOC*, pages 373–382. ACM, 2001. ISBN 1-58113-349-9. [156]

B. Müller, F. Serrano, and A. Gleixner. Using two-dimensional projections for stronger separation and propagation of bilinear terms. *SIAM Journal on Optimization*, 30(2):1339 – 1365, 2020. doi: 10.1137/19M1249825. [65]

J. Nederlof. Fast polynomial-space algorithms using Möbius inversion: Improving on Steiner tree and related problems. In *International Colloquium on Automata, Languages, and Programming*, pages 713–725. Springer, 2009. doi: 10.1007/978-3-642-02927-1_59. [15]

J. Nederlof. Fast polynomial-space algorithms using inclusion-exclusion. *Algorithmica*, 65 (4):868–884, 2013. doi: 10.1007/s00453-012-9630-x. [22]

M. Noormohammadpour, C. S. Raghavendra, S. Rao, and S. Kandula. Dccast: Efficient point to multipoint transfers across datacenters. In *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)*, Santa Clara, CA, July 2017. USENIX Association. URL `https://www.usenix.org/conference/hotcloud17/program/presentation/noormohammadpour`. [14]

P.-M. Olsson, J. Kvarnström, P. Doherty, O. Burdakov, and K. Holmberg. Generating UAV communication networks for monitoring and surveillance. In *In Proceedings of the International Conference on Control, Automation, Robotics and Vision (ICARCV*, 2010. doi: 10.1109/icarcv.2010.5707968. [164]

T. Pajor, E. Uchoa, and R. F. Werneck. A robust and scalable algorithm for the Steiner problem in graphs. *Mathematical Programming Computation*, Sep 2017. doi: 10.1007/s12532-017-0123-4. [17, 38, 51, 53, 66, 67, 102, and 158]

B. Peters. *Monomial patterns in polynomial optimization.* PhD thesis, Otto-von-Guericke-University Magdeburg, 2021. [7]

T. Polzin. *Algorithms for the Steiner problem in networks.* PhD thesis, Saarland University, 2003. [17, 19, 34, 39, 47, 48, 54, 60, 61, 63, 64, 65, 66, 67, 168, and 185]

T. Polzin and S. V. Daneshmand. A comparison of Steiner tree relaxations. *Discrete Applied Mathematics*, 112(1-3):241–261, 2001a. doi: 10.1016/s0166-218x(00)00318-8. [17, 19, 21, 22, 23, 25, 57, and 209]

T. Polzin and S. V. Daneshmand. Improved Algorithms for the Steiner Problem in Networks. *Discrete Applied Mathematics*, 112(1-3):263–300, Sept. 2001b. doi: 10.1016/S0166-218X(00)00319-X. [17, 20, 26, 28, 35, 37, 38, 39, 51, 52, 55, 106, 171, 172, 173, 184, and 185]

T. Polzin and S. V. Daneshmand. *Extending Reduction Techniques for the Steiner Tree Problem*, pages 795–807. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. ISBN 978-3-540-45749-7. doi: 10.1007/3-540-45749-6_69. [17, 21, 39, 41, 43, 45, 47, 168, and 176]

T. Polzin and S. V. Daneshmand. On Steiner trees and minimum spanning trees in hypergraphs. *Operations Research Letters*, 31(1):12 – 20, 2003. doi: 10.1016/S0167-6377(02)00185-2. [16, 17, 157, and 158]

T. Polzin and S. V. Daneshmand. Practical Partitioning-Based Methods for the Steiner Problem. In C. Àlvarez and M. Serna, editors, *Experimental Algorithms*, pages 241–252, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. doi: 10.1007/11764298_22. [17, 39, 40, and 54]

T. Polzin and S. Vahdati-Daneshmand. *Approaches to the Steiner Problem in Networks*, pages 81–103. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi: 10.1007/978-3-642-02094-0_5. [17]

T. Polzin and S. Vahdati-Daneshmand. The Steiner Tree Challenge: An updated Study. Unpublished manuscript at http://dimacs11.cs.princeton.edu/downloads.html, 2014. [17, 60, 63, 64, 65, 66, 158, 222, and 230]

H.-J. Prömel and A. Steger. *The Steiner Tree Problem: A Tour Through Graphs, Algorithms, and Complexity.* Advanced Lectures in Mathematics. Vieweg, 2002. [14]

L. D. P. Pugliese, M. Gaudioso, F. Guerriero, and G. Miglionico. A lagrangean-based decomposition approach for the link constrained Steiner tree problem. *Optimization Methods and Software*, 33(3):650–670, 2018. doi: 10.1080/10556788.2017.1392518. [164, 165]

T. Ralphs, Y. Shinano, T. Berthold, and T. Koch. *Parallel Solvers for Mixed Integer Linear Optimization*, pages 283–336. Springer International Publishing, Cham, 2018. doi: 10.1007/978-3-319-63516-3_8. [180]

D. Rehfeldt and T. Koch. Transformations for the Prize-Collecting Steiner Tree Problem and the Maximum-Weight Connected Subgraph Problem to SAP. *Journal of Computational Mathematics*, 36(3):459 – 468, 2018a. doi: 10.4208/jcm.1709-m2017-0002. [6, 102, and 137]

D. Rehfeldt and T. Koch. SCIP-Jack—a solver for STP and variants with parallelization extensions: An update. In *Operations Research Proceedings 2017*, pages 191 – 196, 2018b. doi: 10.1007/978-3-319-89920-6_27. [6]

D. Rehfeldt and T. Koch. Combining NP-Hard Reduction Techniques and Strong Heuristics in an Exact Algorithm for the Maximum-Weight Connected Subgraph Problem. *SIAM Journal on Optimization*, 29(1):369–398, 2019. doi: 10.1137/17M1145963. [6, 93, 105, and 110]

D. Rehfeldt and T. Koch. On the exact solution of prize-collecting Steiner tree problems. Technical Report 20-11, ZIB, Takustr. 7, 14195 Berlin, 2020. [6, 145]

D. Rehfeldt and T. Koch. Implications, conflicts, and reductions for steiner trees. In M. Singh and D. P. Williamson, editors, *Integer Programming and Combinatorial Optimization - 22nd International Conference, IPCO 2021, Atlanta, GA, USA, May 19-21, 2021, Proceedings*, volume 12707 of *Lecture Notes in Computer Science*, pages 473–487. Springer, 2021. doi: 10.1007/978-3-030-73879-2_33. [7]

D. Rehfeldt, T. Koch, and S. J. Maher. Reduction techniques for the prize collecting Steiner tree problem and the maximum-weight connected subgraph problem. *Networks*, 73(2): 206–233, 2019. doi: 10.1002/net.21857. [6, 74, 86, 91, 105, 122, 125, and 130]

D. Rehfeldt, Y. Shinano, and T. Koch. SCIP-Jack: An Exact High Performance Solver for Steiner Tree Problems in Graphs and Related Problems. In H. G. Bock, W. Jäger, E. Kostina, and H. X. Phu, editors, *Modeling, Simulation and Optimization of Complex Processes HPSC 2018*, pages 201–223, Cham, 2021. Springer International Publishing. ISBN 978-3-030-55240-4. [6]

G. Reich and P. Widmayer. Beyond Steiner's problem: A VLSI oriented generalization. In M. Nagl, editor, *Graph-Theoretic Concepts in Computer Science*, pages 196–210, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg. doi: 10.1007/3-540-52292-1_14. [163]

G. Reinelt. TSPLIB–a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991. doi: 10.1287/ijoc.3.4.376. [58]

C. C. Ribeiro, E. Uchoa, and R. F. Werneck. A Hybrid Grasp With Perturbations For The Steiner Problem In Graphs. *Informs Journal on Computing*, 14:200–2, 2001. doi: 10.1287/ijoc.14.3.228.116. [17, 51]

G. Robins and A. Zelikovsky. Tighter bounds for graph Steiner tree approximation. *SIAM Journal on Discrete Mathematics*, 19(1):122–134, 2005. doi: 10.1137/s0895480101393155. [16]

I. Rosseti, M. P. de Aragão, C. C. Ribeiro, E. Uchoa, and R. F. Werneck. *New Benchmark Instances for The Steiner Problem in Graphs*, pages 601–614. Springer US, Boston, MA, 2004. doi: 10.1007/978-1-4757-4137-7_28. [17, 58]

M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994. doi: 10.1287/ijoc.6.4.445. [42]

L. Schmidt, C. Hegde, P. Indyk, L. Lu, X. Chi, and D. Hohl. Seismic feature extraction using Steiner tree methods. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1647–1651, April 2015. doi: 10.1109/ICASSP.2015.7178250. [116]

A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998. [9]

A. Segev. The node-weighted Steiner tree problem. *Networks*, 17(1):1–17, 1987. doi: 10.1002/net.3230170102. [116]

Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, T. Koch, and M. Winkler. Solving open MIP instances with parascip on supercomputers using up to 80, 000 cores. In *2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, May 23-27, 2016*, pages 770–779. IEEE Computer Society, 2016. doi: 10.1109/IPDPS.2016.56. [180]

Y. Shinano, D. Rehfeldt, and T. Gally. An easy way to build parallel state-of-the-art combinatorial optimization problem solvers: A computational study on solving steiner tree problems and mixed integer semidefinite programs by using ug[scip-*,*]-libraries. In *Proceedings of the 9th IEEE Workshop Parallel / Distributed Combinatorics and Optimization*, pages 530 – 541, 2019a. doi: 10.1109/IPDPSW.2019.00095. [7]

Y. Shinano, D. Rehfeldt, and T. Koch. Building optimal steiner trees on supercomputers by using up to 43,000 cores. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research. CPAIOR 2019*, volume 11494, pages 529 – 539, 2019b. doi: 10.1007/978-3-030-19212-9_35. [7, 180]

M. L. Shore, L. R. Foulds, and P. B. Gibbons. An algorithm for the Steiner problem in graphs. *Networks*, 12(3):323–333, 1982. doi: 10.1002/net.3230120309. [16]

M. Siebert, S. Ahmed, and G. Nemhauser. A linear programming based approach to the Steiner tree problem with a fixed number of terminals. *Networks*, 75(2):124–136, 2020a. doi: 10.1002/net.21913. [15]

M. Siebert, S. Ahmed, and G. Nemhauser. A Simulated Annealing Algorithm for the Directed Steiner Tree Problem. *arXiv preprint arXiv:2002.03055*, 2020b. [154, 155]

D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, June 1983. doi: 10.1016/0022-0000(83)90006-5. [107]

T. L. Snyder. On the Exact Location of Steiner Points in General Dimension. *SIAM Journal of Applied Mathematics*, 21(1):163–180, 1992. doi: 10.1137/0221013. [157]

P. M. Spira and A. Pan. On finding and updating spanning trees and shortest paths. *SIAM Journal on Computing*, 4(3):375–380, 1975. doi: 10.1137/0204032. [175]

Y. Sun, M. Brazil, D. Thomas, and S. Halgamuge. The fast heuristic algorithms and post-processing techniques to design large and low-cost communication networks. *IEEE/ACM Transactions on Networking*, pages 1–14, 2019. doi: 10.1109/TNET.2018.2888864. [116, 146]

Y. Sun, D. Rehfeldt, M. Brazil, D. Thomas, and S. Halgamuge. A physarum-inspired algorithm for minimum-cost relay node placement in wireless sensor networks. *IEEE/ACM Transactions on Networking*, 28(2):681–694, 2020. doi: 10.1109/TNET.2020.2971770. [6, 152, and 153]

H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Mathematica Japonicae*, 24:573 – 577, 1980. [16, 51]

R. E. Tarjan. Applications of Path Compression on Balanced Trees. *The Journal of the ACM*, 26(4):690–715, Oct. 1979. doi: 10.1145/322154.322161. [171]

R. E. Tarjan and U. Vishkin. Finding biconnected componemts and computing tree functions in logarithmic parallel time. In *25th Annual Symposium on Foundations of Computer Science, 1984.*, pages 12–20. IEEE, 1984. doi: 10.1109/sfcs.1984.715896. [172]

N. Tuncbag, S. J. Gosline, A. Kedaigle, A. R. Soltis, A. Gitter, and E. Fraenkel. Network-based interpretation of diverse high-throughput datasets through the omics integrator software package. *PLoS computational biology*, 12(4):e1004879, 2016. [116]

E. Uchoa. Reduction tests for the prize-collecting Steiner problem. *Operations Research Letters*, 34(4):437 – 444, 2006. doi: 10.1016/j.orl.2005.02.007. [93, 94, 117, 122, 123, 124, 125, and 146]

E. Uchoa and R. F. F. Werneck. Fast Local Search for Steiner Trees in Graphs. In G. E. Blelloch and D. Halperin, editors, *ALENEX*, pages 1–10. SIAM Journal of Applied Mathematics, 2010. doi: 10.1137/1.9781611972900.1. [17, 51, 107, 144, and 177]

E. Uchoa, M. Poggi de Aragão, and C. C. Ribeiro. Preprocessing Steiner problems from VLSI layout. *Networks*, 40(1):38–50, 2002. doi: 10.1002/net.10035. [17, 26, and 43]

S. Vahdati Daneshmand. *Algorithmic approaches to the Steiner problem in networks*. PhD thesis, Universität Mannheim, 2004. [17, 19, 54, 60, 61, 63, 64, 65, 66, 67, and 185]

A. van den Boogaart. Efficient computation of fiber optic networks. Master's thesis, Eindhoven University of Technology, 2018. [7]

A. Verma, A. Buchanan, and S. Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS Journal on Computing*, 27(1):164–177, 2015. doi: 10.1287/ijoc.2014.0618. [85]

S. Voss. A survey on some generalizations of Steiner's problem. *1st Balkan Conference on Operational Research Proceedings*, 1:41–51, 1988. [163]

S. Voss. *Steiner-probleme in Graphen*. Hain, 1990. [14]

S. Voß. The Steiner tree problem with hop constraints. *Annals of Operations Research*, 86: 321–345, 1999. doi: 10.1023/A\%3A1018967121276. [164]

J. Vygen. Faster algorithm for optimum Steiner trees. *Information Processing Letters*, 111 (21):1075 – 1079, 2011. doi: 10.1016/j.ipl.2011.08.005. [15, 120]

Y. Wang, A. Buchanan, and S. Butenko. On imposing connectivity constraints in integer programs. *Mathematical Programming*, pages 1–31, 2017. doi: 10.1007/s10107-017-1117-8. [70, 74, 76, 84, and 86]

D. Warme, P. Winter, and M. Zachariasen. Exact algorithms for plane Steiner tree problems: A computational study. In D.-Z. Du, J. Smith, and J. Rubinstein, editors, *Advances in Steiner Trees*, pages 81–116. Kluwer, 2000. [157]

D. M. Warme. *Spanning Trees in Hypergraphs with Applications to Steiner Trees*. PhD thesis, University of Virginia, USA, 1998. [157]

S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal. Optil.io: Cloud based platform for solving optimization problems using crowdsourcing approach. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*, CSCW '16 Companion, page 433–436, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450339506. doi: 10.1145/2818052.2869098. [62]

D. Watel and M.-A. Weisser. A practical greedy approximation for the directed Steiner tree problem. *Journal of Combinatorial Optimization*, 32(4):1327–1370, 2016. doi: 10.1007/s10878-016-0074-0. [155]

Wikipedia. Skipjack tuna. `https://en.wikipedia.org/wiki/Skipjack_tuna`, 2021. Accessed: Feb. 18, 2021. [168]

P. Winter. Reductions for the rectilinear Steiner tree problem. *Networks*, 26(4):187–198, 1995. doi: 10.1002/net.3230260404. [43]

J. Witzig and A. Gleixner. Conflict-driven heuristics for mixed integer programming. *INFORMS Journal on Computing*, 2020. doi: 10.1287/ijoc.2020.0973. epub ahead of print. [65]

R. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28:271–287, 1984. doi: 10.1007/BF02612335. [11, 20, 38, 53, 100, 106, 136, 142, and 144]

A. Z. Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica*, 9(5):463–470, 1993. doi: 10.1007/bf01187035. [16]

E. Álvarez Miranda and M. Sinnl. A relax-and-cut framework for large-scale maximum weight connected subgraph problems. *Computers & Operations Research*, 87:63 – 82, 2017. doi: 10.1016/j.cor.2017.05.015. [86, 105, and 111]

Appendix A

Further proofs

A.1 Steiner tree problem in graphs

A.1.1 Proof of Proposition 2.23

Proof. Suppose that there is a minimum Steiner tree $S^{(k)}$ with $e_1, e_2 \in E^{(k)}(S^{(k)})$. Let $x \in \Lambda^{(k)}(e_1) \cap \Lambda^{(k)}(e_2)$. Let i be the first reduction iteration with $\lambda^{(i)} = x$. We may assume that $i = 1$. Otherwise, we can define additional ancestor information $\overline{\Pi}$ and $\overline{\Lambda}$ starting from $I^{(i-1)}$, and perform the reductions from iteration i to iteration k. Let v be the vertex that is replaced in iteration $i = 1$. Note that $x = \lambda^{(1)} = 1$. From Observation 2.22 we know that the tree S defined by $E(S) = \bigcup_{e \in E^{(k)}} \Pi^{(k)}(e) \cup \Pi^{(k)}_{FIX}$ is a minimum Steiner tree for I. However, because of $\lambda^{(1)} \in \Lambda^{(k)}(e_1) \cap \Lambda^{(k)}(e_2)$, we have that $\left| \left(\Pi^{(k)}(e_1) \cup \Pi^{(k)}(e_2) \right) \cap \delta_S(v) \right| \geqslant 3$. This implies however, that replacing v is not valid—a contradiction. □

A.2 Maximum-weight connected subgraph problem

A.2.1 Proof of Proposition 3.21

Proof. Let S be an optimal solution to I_{MW} (and, as before, a tree) such that $v_i \in V(S)$ and $|V(S) \cap T| \geqslant 2$. Define $\mathcal{Q}$ as in the proof of Proposition 3.20, and note that $\mathcal{Q} \neq \emptyset$ (because of $|V(S) \cap T| \geqslant 2$). Next, choose a path $Q_k \in \mathcal{Q}$ with a minimum number of H-boundary edges. Further, define $\mathcal{Q}^- := \mathcal{Q} \setminus \{Q_k\}$. As before, for all $Q_r \in \mathcal{Q}^-$, denote by Q'_r the subpath of Q_r from t_r up to the last vertex still in H_{t_r}. As in the proof of Proposition 3.20, one validates that the Q'_r are pairwise vertex disjoint and that Q_k has no vertex in common with any Q'_r. One goes on to obtain an upper bound on the weight of S:

$$
\begin{aligned}
P(S) &= \sum_{v \in V(S)} p(v) \\
&\leqslant \left(\sum_{Q_r \in \mathcal{Q}^-} P(Q'_r) \right) + P(Q_k) \\
&\leqslant \sum_{t \in T_p \setminus \{v_i\}} r^+_H(t) - \min\{ r^+_H(t) \mid t \in T_p \setminus \{v_i\}\} + P(Q_k) \\
&\leqslant \sum_{t \in T_p \setminus \{v_i\}} r^+_H(t) - \min\{ r^+_H(t) \mid t \in T_p \setminus \{v_i\}\} + \overline{d}(v_i, \underline{v}^{H^p}_{i,1}).
\end{aligned}
$$

These inqualities conclude the proof of the proposition. □

A.2.2 Node separators and rejoining of flows

The following is joint work with Henriette Franz. Consider the directed RMWCSP instance (G, T, p, r) with $G = (V, A)$ depicted in Figure A.1.

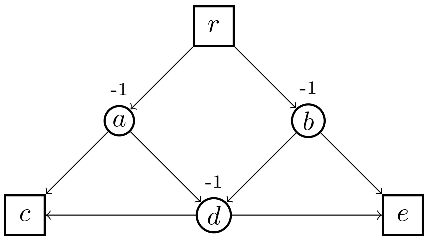


Figure A.1: Directed RMWCSP instance.

A proof from Álvarez-Miranda et al. (2013b) intends to show that $v_{LP}(RNCut) \leqslant v_{LP}(\text{RSA})$ holds. For this purpose, the authors consider an arbitrary solution $\bar{x} \in \mathcal{P}_{LP}(\text{RNCut})$ and construct an auxiliary graph G' by replacing each node $v \in V \setminus \{r\}$ with an arc (v_1, v_2). All ingoing arcs of v become ingoing arcs of v_1, and all outgoing arcs of v are now outgoing arcs of v_2. Moreover, (non-negative) capacities k' on G' are introduced for each arc (v', w') of G' by

$$k'(v', w') := \begin{cases} \bar{x}(v), & \text{if } (v', w') = (v_1, v_2) \text{ for a } v \in V \\ 1, & \text{otherwise.} \end{cases}$$

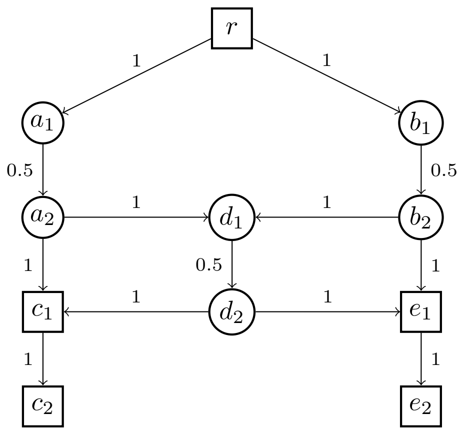


Figure A.2: Illustration of an auxiliary support graph G' corresponding to the instance in Figure A.1 regarding the optimal solution $\bar{x}(v) = 0.5$, $v \in V \setminus T$, and $\bar{x}(t) = 1$, $t \in T$, to the RNCut formulation.

Figure A.2 shows an auxiliary support graph of the instance illustrated by Figure A.1. It is possible to send a flow with flow value $\bar{x}(v)$ from root node r to each arc (v_1, v_2) with $v \in V \setminus \{r\}$ because of constraints (3.10). Let $f^v(j,l)$ be the amount of a flow with source node r, sink node $v \in V \setminus \{r\}$, and flow value $\bar{x}(v)$ sent along arc (j,l). Define the arc variables $\hat{y}(j,l)$, $(j,l) \in A$, of the RSA formulation as follows:

$$\hat{y}(j,l) := \begin{cases} \max_{v \in V \setminus \{r\}} f^v(j_2, l_1), & j, l \in V \setminus \{r\} \\ \max_{v \in V \setminus \{r\}} f^v(j, l_1), & j = r, l \in V \setminus \{r\}. \end{cases}$$

Hence, the arc variables of the instance in Figure A.2 are given by $\hat{y}(j,l) = 0.5$ for each $(j,l) \in A$. Moreover, define the node variables as $\hat{x}(v) = \hat{y}(\delta^-(v))$. Thus, in our case, it holds $\hat{x}(a)$, $\hat{x}(b)$, $\hat{x}(c)$, $\hat{x}(e) = 0.5$, and $\hat{x}(d) = 1$. The proof from Álvarez-Miranda et al. (2013b) claims that we can follow $\bar{x}(v) = \hat{x}(v)$, $v \in V$, by this definition of the variables. However, this claim is not true because of $0.5 = \bar{x}(d) \neq \hat{x}(d) = 1$, and therefore, no solution can be constructed from the solution $\bar{x}$ to the RNCut model.

In summary, and somewhat broadly speaking, the weaker LP relaxation can be explained as follows. The RNCut formulation can be interpreted as a multi-commodity flow problem in an enlarged graph. However, enlarging the graph opens new possibilities for what is sometimes called *rejoining of flows* (Polzin and Daneshmand, 2001a): Flows for different commodities enter a node on different arcs, but leave on the same arc. Such a rejoining can lead to an increased integrality gap.

A.3 Prize-collecting Steiner tree problem

A.3.1 Proof of Theorem 4.1

The proof of Theorem 4.1 is based on the well-known dynamic programming algorithm for SPG by Dreyfus and Wagner (1971) that runs in $O(3^{|T|}n + 2^{|T|}n^2 + n^2 \log n + mn)$, where T is the set of terminals. We will refer to this algorithm as *Dreyfus-Wagner* for short. See also Buchanan et al. (2018) for an extension of Dreyfus-Wagner to the node-weighted Steiner tree problem. Dreyfus-Wagner exploits the fact that any optimal Steiner tree S for an SPG (G, T, c), with $T \neq \emptyset$ and positive c, can be split at any $v \in V(S)$ into two non-empty trees S_1 and S_2 such that $T_1 := V(S_1) \cap T \neq \emptyset$, $T_2 := V(S_2) \cap T \neq \emptyset$, and:

1. $T_1 \cap T_2 \subseteq \{v\}$ and $T_1 \cup T_2 = T$,
2. S_1 is optimal for $(G, T_1 \cup \{v\}, c)$, and S_2 is optimal for $(G, T_2 \cup \{v\}, c)$.

We show that a similar property holds for PCSTP. To this end, consider the RPCSTP I_f, defined in Section 4.2.1. Moreover, we will consider only optimal (PCSTP and RPCSTP) solutions that contain only proper potential or fixed terminals as leafs. If no such solution exists, there is a trivial optimal solution, which can be found in linear time. For any $T \subseteq T_f$ we denote by $I_f(T)$ the RPCSTP (G, T, c, p); so in particular $I_f(T_f) = I_f$. For I_f we obtain the following result.

Lemma A.1. *Let $T_1, T_2 \subseteq T_f$ be non-empty with $T_1 \cup T_2 = T_f$. Let S_1, S_2 be trees in G such that all leaves of S_1 are contained in T_1, all leaves of S_2 are contained in T_2, and $S_1 \cap S_2 \neq \emptyset$. In this case, there is a tree $S \subseteq S_1 \cup S_2$ such that $T_f \subseteq V(S)$, and*

$$C(S) \leqslant C(S_1) + C(S_2) - \sum_{u \in V} p(u) + \min_{u \in V(S_1 \cap S_2)} p(u). \tag{A.1}$$

Proof. Initially, set $S := S_1 \cup S_2$ and $\hat{S} := S_1 \cap S_2$. Let $v_0 \in V(\hat{S})$ such that $p(v_0) = \min_{u \in V(\hat{S})} p(u)$. Let $\vec{S}_1$ be the arborescence corresponding to S_1 that is rooted in v_0. Denote its arcs by $A(\vec{S}_1)$. For any $w \in V(\hat{S}) \setminus \{v_0\}$ there is an (incoming) arc $(u, w) \in A(\vec{S}_1)$. Let $E'_S := \{\{u, w\} \in E(S_1) \setminus E(S_2) \mid w \in V(\hat{S}) \wedge (u, w) \in A(\vec{S}_1)\}$ and $E''_S := \{\{u, w\} \in E(\hat{S}) \mid (u, w) \in A(\vec{S}_1)\}$. Note that $|E'_S \dot{\cup} E''_S| = |V(\hat{S})| - 1$. Because of $T_p^+ = \emptyset$ it holds that

$$\sum_{e \in E'_S} c(e) + \sum_{e \in E''_S} c(e) \geqslant \sum_{u \in V(\hat{S}) \setminus \{v_0\}} p(u). \tag{A.2}$$

Because of $E''_S \subseteq E(\hat{S})$, inequality (A.2) implies

$$\sum_{e \in E'_S} c(e) + \sum_{e \in \hat{S}} c(e) \geqslant \sum_{u \in V(\hat{S}) \setminus \{v_0\}} p(u). \tag{A.3}$$

Note that any $e \in E'_S$ lies in a cycle of S and that each cycle of S contains an $e \in E'_S$. Remove E'_S from S to obtain a new tree $\tilde{S}$ (which contains T_f). It holds that:

$$\begin{aligned}
C(S_1) + C(S_2) &= C(S) + \sum_{e \in \hat{S}} c(e) + \sum_{u \in V} p(u) - \sum_{u \in V(\hat{S})} p(u) && \text{(A.4)}\\
&= C(\tilde{S}) + \sum_{e \in E'_S} c(e) + \sum_{e \in \hat{S}} c(e) + \sum_{u \in V} p(u) - \sum_{u \in V(\hat{S})} p(u) && \text{(A.5)}\\
&\overset{\text{(A.3)}}{\geqslant} C(\tilde{S}) + \sum_{u \in V} p(u) - p(v_0) && \text{(A.6)}\\
&= C(\tilde{S}) + \sum_{u \in V} p(u) - \min_{u \in V(S_1 \cap S_2)} p(u). && \text{(A.7)}
\end{aligned}$$

Thus, $\tilde{S}$ satisfies (A.1). □

This lemma sets the stage for the desired result:

Lemma A.2. *Let S be an optimal solution to I_f and choose any, arbitrary but fixed, $v \in V(S)$. Further, let $S_1, S_2 \subseteq S$ be trees such that $V(S_1 \cap S_2) = \{v\}$ and $S_1 \cup S_2 = S$. Define $T_1 := (T_f \cap V(S_1)) \cup \{v\}$ and $T_2 := (T_f \cap V(S_2)) \cup \{v\}$. It holds that S_1 is an optimal solution to $I_f(T_1)$, and S_2 to $I_f(T_2)$. Furthermore:*

$$C(S) = C(S_1) + C(S_2) - \sum_{u \in V \setminus \{v\}} p(u) \tag{A.8}$$

holds.

Proof. First, observe that (A.8) holds because of $V(S_1 \cap S_2) = \{v\}$. Suppose S_1 is not optimal. Thus, there exists a tree $\tilde{S}_1$ such that all its leaves are contained in T_1 and such that

$$C(\tilde{S}_1) < C(S_1). \tag{A.9}$$

We also assume that all leaves of S_2 are contained in T_2; note that because of $T_p^+ = \emptyset$ one can always modify S_2 to satisfy this property without increasing $C(S_2)$. By Lemma A.1

there exists a $\tilde{S} \subseteq \tilde{S}_1 \cup S_2$ such that $T_f \subseteq V(\tilde{S})$ and

$$C(\tilde{S}) \leqslant C(\tilde{S}_1) + C(S_2) - \sum_{u \in V} p(u) + p(v) \tag{A.10}$$

$$\overset{(A.9)}{<} C(S_1) + C(S_2) - \sum_{u \in V} p(u) + p(v) \tag{A.11}$$

$$= C(S), \tag{A.12}$$

which is a contradiction to the assumption that S is optimal. □

Based on the proceeding lemma, we can apply an extension of Dreyfus-Wagner to solve I_f. We define an slight modification of the prize-collecting cost from Section **??**. For a (v, w)-walk W let

$$c'_{pc}(W) := \sum_{e \in E(W)} c(e) - \sum_{u \in V(W) \setminus \{w\}} p(u). \tag{A.13}$$

Let $\mathcal{W}(v, w)$ be the set of all finite walks from v to w and define

$$d'_{pc}(v, w) := \min\{c'_{pc}(W) \mid W \in \mathcal{W}(v, w)\}. \tag{A.14}$$

Note that if $T_p^+ = \emptyset$, it is sufficient to consider only simple paths instead of walks. The next subsection concludes the proof of Theorem 4.1.

Proof of Proposition 4.2

Proof. Initially, choose an arbitrary $t_0 \in T_f$ and set $T_f^- := T_f \setminus \{t_0\}$. For every pair (v, w) of vertices, set

$$g(\{w\}, v) := d'_{pc}(v, w) + \sum_{u \in V \setminus \{w\}} p(u). \tag{A.15}$$

For $i = 2, ..., |T_f^-|$ define the functions f and g recursively as follows. For $T_i \subseteq T_f^-$ with $|T_i| = i$ set

$$f(T_i, w) = \min_{T \subsetneq T_i | T \neq \emptyset} \big(g(T, w) + g(T_i \setminus T, w) - \sum_{u \in V \setminus \{w\}} p(u)\big) \tag{A.16}$$

and

$$g(T_i, v) = \min_{u \in V} \big(f(T_i, u) + d'_{pc}(v, u)\big). \tag{A.17}$$

These values can be computed by a dynamic programming algorithm.

Claim 1: *After the termination of the above dynamic programming algorithm it holds that $g(T_f^-, t_0) = C(S)$ for any optimal solution S to I_f.*

We will show by induction on $i \in \{1, ..., |T_f^-|\}$ that for any $T_i \subseteq T_f^-$ with $|T_i| = i$, and any $v \in V \setminus T_i$ it holds that

$$g(T_i, v) = C(S) \tag{A.18}$$

for any optimal solution S to $I_f(T_i \cup \{v\})$. First, one observes from the definition of d'_{pc} that (A.18) holds for any $t \in T_f^-$ and any $v \in V \setminus \{t\}$. Next, let $i \in \{2, ..., |T_f^-|\}$. Assume that (A.18) holds for all non-empty $T \subset T_f^-$ with $|T| < i$. Choose any $T_i \subseteq T_f^-$ with $|T_i| = i$, and choose any $v \in V \setminus T_i$. Let S be an optimal solution to $I_f(T_i \cup \{v\})$. Split S as follows:

If $\delta_S(v) = 1$ let $P := (v, \emptyset)$, otherwise let $P \subseteq S$ be the path from v to the first vertex $w \in V(S)$ with $\delta_S(w) > 2$ or $w \in T_i$. Observe that

$$C(P) = d'_{pc}(v,w) + \sum_{u \in V \setminus \{w\}} p(u). \tag{A.19}$$

Let $\tilde{S} := (V(S \setminus P) \cup \{w\}, E(S \setminus P))$. Because of Lemma A.2, $\tilde{S}$ is an optimal solution to $I_f(T_i \cup \{w\})$ and P to $I_f(\{v,w\})$. Further:

$$C(S) = C(\tilde{S}) + C(P) - \sum_{u \in V \setminus \{w\}} \overset{\text{(A.19)}}{=} C(\tilde{S}) + d'_{pc}(v,w). \tag{A.20}$$

Moreover, $\tilde{S}$ can be split into two trees $\tilde{S}_1$ and $\tilde{S}_2$ such that $\tilde{S}_1 \cap \tilde{S}_2 = \{w\}$, $\tilde{S}_1 \cup \tilde{S}_2 = \tilde{S}$, and $\tilde{S}_1 \cap T_i \neq \emptyset$, $\tilde{S}_2 \cap T_i \neq \emptyset$. With $\tilde{T}_1 := (T_i \cap \tilde{S}_1) \cup \{w\}$ and $\tilde{T}_2 := (T_i \cap \tilde{S}_2) \cup \{w\}$, it holds by Lemma A.2 that $\tilde{S}_1$ is an optimal solution to $T_f(\tilde{T}_1)$ and $\tilde{S}_2$ to $T_f(\tilde{T}_2)$. Lemma A.2 furthermore implies that:

$$f(T_i, w) \leqslant C(\tilde{S}_1) + C(\tilde{S}_2) - \sum_{u \in V \setminus \{w\}} p(u). \tag{A.21}$$

From the optimality of $\tilde{S}$ combined with Lemma A.1 we obtain:

$$f(T_i, w) = C(\tilde{S}_1) + C(\tilde{S}_2) - \sum_{u \in V \setminus \{w\}} p(u). \tag{A.22}$$

Similarly, from Lemma A.1 and Lemma A.2 we obtain.

$$\begin{aligned} g(T_i, w) &\overset{\text{(A.22)}}{\leqslant} C(\tilde{S}_1) + C(\tilde{S}_2) + d'_{pc}(v,w) && \text{(A.23)} \\ &= C(\tilde{S}) - \sum_{u \in V \setminus \{w\}} p(u) + d'_{pc}(v,w) && \text{(A.24)} \\ &\overset{\text{(A.20)}}{=} C(S). && \text{(A.25)} \end{aligned}$$

Equality follows from Lemma A.1 and the optimality of S.

Claim 2: *The above dynamic programming algorithm terminates in time* $O(3^{|T_f|}n + 2^{|T_f|}n^2 + n^2 \log n + mn)$.

For $i \geqslant 2$ the algorithm differs from Dreyfus-Wagner essentially only in the weight functions of the trees. For $i = 1$ one observes the following. For a given $v \in V$, the distances $d'_{pc}(v,w)$ on I_f to all $w \in V$ can be computed in time $O(n \log n + m)$ by using an adaptation of Dijkstra's algorithm, similar to Algorithm 4.1, that runs in time $O(n \log n + m)$. Thus, the distances for all pairs (v,w) can be computed in $O(n^2 \log n + mn)$. Consequently, the overall dynamic programming algorithm algorithm has the same run time as Dreyfus-Wagner. □

A.3.2 Proof of Proposition 4.11

Proof. First, one can verify from the definition of Algorithm 4.1 that if it returns *deletable*, then $c(\{v,w\}) \geqslant d^-_{pc}(v,w)$ holds—also without condition (4.19). To show the converse, assume in the following that $c(\{v,w\}) \geqslant d^-_{pc}(v,w)$. To simply the presentation it will also be assumed that

$$p(v) = 0, \tag{A.26}$$

which does neither change $d^-_{pc}(v,w)$, nor the behavior of Algorithm 4.1. Further, note that because of (4.19) one can assume that W is a (simple) path. Otherwise, replace W by a shortest path (with respect to the edge costs c) between v and w in the subgraph corresponding to W. Indeed, because of (4.19) the prize-constrained length of this shortest path is not higher than that of W. As before, write $W = (v_1, e_1, v_2, e_2, ..., e_r, v_r)$ with $v_1 = v$ and $v_r = w$. Condition (4.19) furthermore implies that

$$l_{pc}(W(v, v_{k+1})) = l_{pc}(W(v, v_k)) + c(\{v_k, v_{k+1}\}) - p(v_k) \tag{A.27}$$

for any $k \in \{1, 2, ..., r-1\}$.

In the following, we will show that

$$dist_{pc}[v_k] \leqslant l_{pc}(W(v, v_k)) - p(v_k) \tag{A.28}$$

holds for any $k \in \{1, ..., r-1\}$. Thereby, the proof is concluded: Algorithm 4.1 can in this case reach w from v_{r-1} due to

$$dist_{pc}[v_{r-1}] + c(\{v_{r-1}, v_r\}) \overset{(A.28)}{\leqslant} l_{pc}(W(v, v_{r-1})) - p(v_{r-1}) + c(\{v_{r-1}, v_r\}) \tag{A.29}$$

$$\overset{(A.27)}{=} l_{pc}(W(v, v_r)) \tag{A.30}$$

$$= d^-_{pc}(v, w) \tag{A.31}$$

$$\leqslant c(\{v, w\}). \tag{A.32}$$

Thus, the algorithm returns *deletable*.

We will show (A.28) by induction on $k = 1, ..., r-1$. First, one readily verifies that (A.28) holds for $k = 1$. Next, let $k \in \{1, ..., r-2\}$ and assume that (A.28) holds for k. Suppose

$$dist_{pc}[v_{k+1}] > l_{pc}(W(v, v_{k+1})) - p(v_{k+1}). \tag{A.33}$$

Now perform Algorithm 4.1 until $dist_{pc}[v_k]$ satisfies (A.28). At this point, $forbidden[v_{k+1}]$ must have been set to $true$, otherwise one could update $dist_{pc}[v_{k+1}]$ from vertex v_k: Indeed, $dist_{pc}[v_k] + c(\{v_k, v_{k+1}\}) \leqslant c(\{v, w\})$ holds, which can be shown equivalently to (A.29)-(A.32). Thus, also the second condition for updating $dist_{pc}[v_{k+1}]$ from vertex v_k is fulfilled. For the third (and last condition), one obtains:

$$dist_{pc}[v_k] + c(\{v_k, v_{k+1}\}) - p(v_{k+1}) \overset{(A.28)}{\leqslant} l_{pc}(W(v, v_k) - p(v_k) + c(\{v_k, v_{k+1}\}) - p(v_{k+1}) \tag{A.34}$$

$$\overset{(A.27)}{=} l_{pc}(W(v, v_{k+1}) - p(v_{k+1}) \tag{A.35}$$

$$\overset{(A.33)}{<} dist_{pc}[v_{k+1}]. \tag{A.36}$$

If $forbidden[v_{k+1}]$ is set to $true$, the vertex v_{k+1} must already have been removed from Q in Algorithm 4.1 (which happens exactly one time, because at this point v_{k+1} will be marked as forbidden). We will show (by induction) for $j = 1, ..., k$ that v_j satisfies (A.28) at the point when v_{k+1} is removed from Q. In this way, we obtain a contradiction, because if v_k satisfies (A.28), then

$$dist_{pc}[v_k] \overset{(A.28)}{\leqslant} l_{pc}(W(v, v_k) - p(v_k) \tag{A.37}$$

$$\overset{(A.27)}{=} l_{pc}(W(v, v_{k+1}) - c(\{v_k, v_{k+1}\}) \tag{A.38}$$

$$\overset{(4.19)}{\leqslant} l_{pc}(W(v, v_{k+1}) - p(v_{k+1}) \tag{A.39}$$

$$\overset{(A.33)}{<} dist_{pc}[v_{k+1}]. \tag{A.40}$$

This implies that v_k would have been removed before v_{k+1} from Q. Consequently, the algorithm would have updated $dist_{pc}(v_{k+1})$ to $dist_{pc}[v_k]+c(\{v_k, v_{k+1}\})-p(v_{k+1})$, and (A.28) would hold for v_{k+1}, as shown in (A.34),(A.35).

We conclude with the induction for $j = 1, ..., k$. By definition, v_1 satisfies (A.28) when v_{k+1} is removed from Q. Assume that the same holds for v_j with $j \in \{2, ..., k-1\}$. Then v_j must have been removed from Q before v_{k+1}, because $dist_{pc}[v_j] < dist_{pc}[v_{k+1}]$ holds, which can be shown similarly to (A.37)-(A.40). Thus, one could update $dist_{pc}[v_{j+1}]$ from v_j to

$$dist_{pc}[v_{j+1}] := dist_{pc}[v_j] + c(\{v_j, v_{j+1}\}) - p(v_{j+1}) \overset{(A.27),(A.28)}{\leqslant} l_{pc}(W(v, v_{j+1})) - p(v_{j+1}), \tag{A.41}$$

which shows that v_{j+1} satisfies (A.28).

□

A.3.3 Proof of Lemma 4.14

Proof. Assume there is spanning tree S such that $\{v, w\} \notin S$. Remove from $E(S)$ an edge on the (unique) path between t_i and t_j in S of maximum cost. By definition of $b_{\{v,w\}}(t_i, t_j)$ it holds that

$$c(E(S_i)) + c(E(S_j)) + b_{\{v,w\}}(t_i, t_j) \leqslant c(E(S)). \tag{A.42}$$

This operation results in two disjoint trees: S_i with $t_i \in S_i$ and S_j with $t_j \in S_j$. If v and w are in different trees, one can add $\{v, w\}$ to connect S_i and S_j and obtain a spanning tree of no higher cost than S. Otherwise assume that $v, w \in V(S_j)$. Let W_i be a prize-constrained (v, t_i)-walk with $l_{pc}(W_i) = d_{pc}(v, t_i)$. There is at least one edge $\{p, q\} \in E(W_i)$ such that $p \in V(S_i)$ and $q \in V(S_j)$. By definition of the prize-constrained length it holds that $c(\{p, q\}) \leqslant l_{pc}(W_i)$. Thus, one can add both $\{p, q\}$ and $\{v, w\}$ to S_i, S_j to obtain a connected spanning subgraph S'. Because of condition (4.22) and (A.42) it holds that

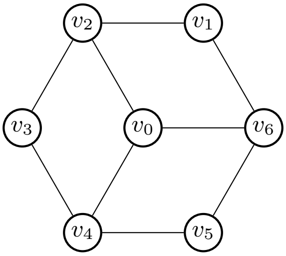
$$c(E(S')) \leqslant c(E(S)). \tag{A.43}$$

Delete any edge other than $\{v, w\}$ on the cycle in $E(S')$ that includes $\{v, w\}$. In this way one obtains a spanning tree S'' of no higher cost than S. □

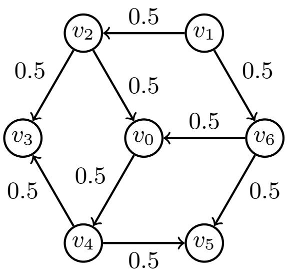
A.3.4 Proof of Proposition 4.30

Proof. We only show the second part of the proposition. First it follows from the construction of Transformation 4.20 and 4.25 that each optimal solution x^0, y^0 to the LP relaxation of $TransRCut(I_{T_0})$ can be transformed to a solution x, y to the LP relaxation of $TransCut(I_{PC})$ without changing the objective value: By setting $x((v_i, v_j)) :=$

$x^0((v_i, v_j))$ and $x((v_j, v_i)) := x^0((v_j, v_i))$ for all $\{v_i, v_j\} \in E$, $x((r', t_0)) := 1$ for any $t_0 \in T_0$, $x((t_i, t_i')) := 1$ for all $t_i \in T_0$, and by setting the remaining $x((v_i, v_j))$ accordingly. Thus $v_{LP}(PrizeCut(I_{PC})) \leqslant v_{LP}(PrizeRCut(I_{T_0}))$. To see that the inequality can be strict, consider the following wheel instance (which is well-known to have an integrality gap for DCut on SPG):



Set $c(e) = 1$ for all edges e. Further, set $p(v_0) = p(v_1) = p(v_3) = p(v_5) = 4, p(v_2) = p(v_6) = 0$, and $p(v_4) = \varepsilon$ with $0 < \varepsilon < 1$. Let $T_0 := \{v_0, v_1, v_3, v_4, v_5\}$. Let I be the PCSTP and I_{T_0} the corresponding RPCSTP. It holds that $v_{LP}(TransCut(I)) = 4.5 + \dfrac{\varepsilon}{2} < 5 = v_{LP}(TransRCut(I_{T_0}))$. Part of the solution corresponding to $v_{LP}(TransCut(I))$ is shown below (with numbers next to the arcs denoting the x values), the remaining x and y are set accordingly (e.g., $x((r', v_1)) = 1$).



□

Appendix B

Detailed computational results

This appendix provides detailed computational results on the problem instances discussed in this thesis.

All following tables are structured as follows: First, the name of the respective instance is given. The next three columns give the number of vertices, arcs, and terminals of the instance, but only after the respective graph transformation to SAP. The subsequent segment, labelled "Presolved", provides the size of the preprocessed problem along with the preprocessing time. The last segment provides first the dual and primal bound, or the optimal solution value if the problem could be solved to proven optimality. Moreover, the number of branch-and-bound nodes (N) and the total run time is given. A time-out is signified by a ">" in front of the termination time. We stress that the reported final execution times include both the preprocessing time and the reading time.

B.1 Steiner tree problem in graphs

B.1.1 PACE 2018 instances

The time limit for the following instances is 1620 seconds (which roughly corresponds to the 30 minutes time-limit on the machines used at the PACE Challenge). For all instances SoPlex 5.0 was used as LP solver.

Table B.1. Detailed computational results for SPG, test-set Pace (Track A).

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Dual	Primal	Gap %	N	t [s]
instance001	53	160	4	0	0	0	0.0	**503**			1	0.0
instance003	2500	10000	5	0	0	0	0.0	**73**			1	0.0
instance005	2500	125000	5	0	0	0	0.2	**15**			1	0.2
instance007	157	532	6	0	0	0	0.0	**1239**			1	0.0
instance009	57	168	8	0	0	0	0.0	**926**			1	0.0
instance011	64	576	8	64	576	8	0.0	**23**			1	0.0
instance013	640	1920	9	0	0	0	0.0	**4033**			1	0.0
instance015	640	1920	9	0	0	0	0.0	**3438**			1	0.0
instance017	640	1920	9	0	0	0	0.0	**4006**			1	0.0
instance019	640	8270	9	0	0	0	0.0	**2465**			1	0.0
instance021	640	8270	9	0	0	0	0.0	**2171**			1	0.0
instance023	640	408960	9	0	0	0	0.8	**1749**			1	0.8
instance025	640	408960	9	0	0	0	0.8	**1754**			1	0.8
instance027	90	270	10	0	0	0	0.0	**188**			1	0.0

cont. next page

Instance	Original $\|V\|$	$\|A\|$	$\|T\|$	Presolved $\|V\|$	$\|A\|$	$\|T\|$	t [s]	Dual	Primal	Gap %	N	t [s]
instance029	179	586	10	0	0	0	0.0	**245**			1	0.0
instance031	298	1006	10	0	0	0	0.0	**311**			1	0.0
instance033	331	1120	10	0	0	0	0.0	**319**			1	0.0
instance035	609	1864	10	40	118	7	0.0	**581**			1	0.0
instance037	777	2478	10	0	0	0	0.0	**566**			1	0.0
instance039	875	3044	10	0	0	0	0.0	**604**			1	0.0
instance041	898	3124	10	0	0	0	0.0	**594**			1	0.0
instance043	918	3368	10	0	0	0	0.0	**604**			1	0.0
instance045	1290	4540	10	0	0	0	0.0	**823**			1	0.0
instance047	2500	10000	10	0	0	0	0.0	**145**			1	0.0
instance049	3221	11876	10	0	0	0	0.1	**1550**			1	0.1
instance051	2500	25000	10	0	0	0	0.1	**67**			1	0.1
instance053	128	454	11	0	0	0	0.0	**1100361**			1	0.0
instance055	191	604	11	0	0	0	0.0	**311**			1	0.0
instance057	237	780	11	0	0	0	0.0	**353**			1	0.0
instance059	278	956	11	0	0	0	0.0	**564**			1	0.0
instance061	353	1216	11	0	0	0	0.0	**350**			1	0.0
instance063	572	1926	11	0	0	0	0.0	**621**			1	0.0
instance065	720	2538	11	0	0	0	0.0	**508**			1	0.0
instance067	3675	13418	11	0	0	0	0.1	**6673**			1	0.1
instance069	64	384	12	64	384	12	0.0	**3271**			1	0.1
instance071	233	772	12	0	0	0	0.0	**344**			1	0.0
instance073	386	1306	12	0	0	0	0.0	**386**			1	0.0
instance075	818	2924	12	127	408	9	0.0	**5250**			1	0.0
instance077	1981	7266	12	0	0	0	0.0	**6618**			1	0.0
instance079	4045	14188	12	155	506	11	0.2	**1459**			1	0.2
instance081	110	376	13	0	0	0	0.0	**1300798**			1	0.0
instance083	346	1166	13	0	0	0	0.0	**457**			1	0.0
instance085	125	1500	13	125	1032	13	0.0	**20**			1	0.4
instance087	125	1500	13	125	1500	13	0.0	**36**			1	0.8
instance089	933	3264	13	0	0	0	0.0	**550**			1	0.0
instance091	1359	4916	13	0	0	0	0.0	**714**			1	0.0
instance093	165	548	14	0	0	0	0.0	**1348**			1	0.0
instance095	418	1446	14	0	0	0	0.0	**399**			1	0.0
instance097	1196	4168	14	0	0	0	0.0	**745**			1	0.0
instance099	193	738	15	0	0	0	0.0	**1500405**			1	0.0
instance101	311	1158	16	0	0	0	0.0	**1601190**			1	0.0
instance103	402	1380	16	0	0	0	0.0	**393**			1	0.0
instance105	712	2434	16	0	0	0	0.0	**847**			1	0.0
instance107	837	2876	16	183	602	11	0.0	**848**			1	0.0
instance109	1051	3582	16	0	0	0	0.0	**939**			1	0.0
instance111	1848	6572	16	0	0	0	0.0	**914**			1	0.0
instance113	6405	20908	16	3076	10642	11	1.0	**2256**			1	1.0
instance115	122	388	17	0	0	0	0.0	**210**			1	0.0
instance117	220	748	17	0	0	0	0.0	**254**			1	0.0
instance119	310	1028	17	0	0	0	0.0	**370**			1	0.0
instance121	343	1118	17	0	0	0	0.0	**454**			1	0.0
instance123	2039	7096	17	0	0	0	0.0	**1104**			1	0.0
instance125	211	760	18	0	0	0	0.0	**1801464**			1	0.0
instance127	1709	5926	18	0	0	0	0.1	**926**			1	0.1
instance129	3738	14026	18	462	1726	12	0.3	**1570**			1	0.3
instance131	189	706	19	0	0	0	0.0	**1900439**			1	0.0
instance133	321	1080	20	0	0	0	0.0	**4132**			1	0.0
instance135	3683	13434	20	380	1306	12	0.2	**9143**			1	0.2
instance137	529	2064	21	212	788	14	0.1	**2103283**			1	0.1
instance139	770	2766	21	115	384	14	0.0	**750**			1	0.0
instance141	233	862	22	186	686	20	0.0	**2200557**			1	0.1

cont. next page

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	Presolved t [s]	Dual	Primal	Gap %	N	t [s]
instance143	828	2944	22	0	0	0	0.0	**5824**			1	0.0
instance145	132	460	23	0	0	0	0.0	**2300245**			1	0.0
instance147	3983	14216	23	0	0	0	0.2	**1488**			1	0.2
instance149	493	1926	24	299	1136	16	0.1	**2403332**			1	0.1
instance151	8007	29486	24	985	3474	20	7.3	**17803**			1	7.3
instance153	246	936	25	0	0	0	0.0	**2500540**			1	0.0
instance155	58	3306	25	0	0	0	0.0	**13655**			1	0.0
instance157	2213	8270	25	0	0	0	0.1	**1098**			1	0.1
instance159	3636	13578	25	0	0	0	0.1	**1362**			1	0.1
instance161	640	81792	25	331	7034	25	0.7	**5199**			81	339.6
instance163	640	81792	25	184	3264	25	1.9	**5194**			39	49.9
instance165	640	81792	25	521	15368	25	5.4	**5218**			87	1137.3
instance167	396	1562	26	310	1224	26	0.6	**2600443**			1	4.5
instance169	243	994	27	75	272	16	0.1	**2700441**			1	0.1
instance171	243	2430	27	241	2132	25	0.2	**42**			1	28.0
instance173	243	2430	27	243	2430	27	0.1	69.1389127	71	2.7	27	>1620.0
instance175	307	1118	28	0	0	0	0.0	**2800379**			1	0.0
instance177	245	872	29	0	0	0	0.0	**2900479**			1	0.0
instance179	1724	5950	29	0	0	0	0.0	**1244**			1	0.0
instance181	8013	29498	30	0	0	0	6.9	**21757**			1	6.9
instance183	1199	4156	31	0	0	0	0.0	**1068**			1	0.0
instance185	437	1676	33	190	700	20	0.0	**3300513**			1	0.0
instance187	1244	4948	34	1069	4256	32	2.2	**3400646**			1	3.3
instance189	8017	29506	36	0	0	0	7.3	**20678**			1	7.3
instance191	2132	7404	37	0	0	0	0.1	**1590**			1	0.1
instance193	603	2414	38	444	1810	37	1.3	**3800656**			1	5.5
instance195	550	10026	50	550	10026	50	0.2	**54**			111	209.6
instance197	10393	36086	104	0	0	0	0.4	**4292**			1	0.4
instance199	6163	20980	130	10	26	8	3.0	**5099**			1	3.0

Table B.2. Detailed computational results for SPG, test-set PaceTree (Track B).

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	Presolved t [s]	Dual	Primal	Gap %	N	t [s]
instance001	74	292	25	0	0	0	0.0	**1086**			1	0.0
instance003	87	352	30	2	2	1	0.0	**41350**			1	0.0
instance005	201	506	100	0	0	0	0.0	**764269099**			1	0.0
instance007	216	576	100	12	34	7	0.0	**20437**			1	0.0
instance009	229	624	100	11	30	6	0.0	**75952202**			1	0.0
instance011	244	674	100	45	146	18	0.0	**20492**			1	0.0
instance013	1906	4166	1655	33	100	16	0.0	**584948**			1	0.0
instance015	114	456	33	0	0	0	0.0	**1341**			1	0.0
instance017	210	552	100	17	56	7	0.0	**73033178**			1	0.0
instance019	231	638	100	0	0	0	0.0	**70446493**			1	0.0
instance021	247	972	100	30	92	12	0.0	**74070**			1	0.0
instance023	990	2516	574	23	66	12	0.0	**3509275**			1	0.0
instance025	8790	19630	7397	97	296	51	0.1	**22481625**			1	0.1
instance027	15	70	8	15	70	8	0.0	**10**			1	0.0
instance029	197	500	100	0	0	0	0.0	**20401**			1	0.0
instance031	269	798	99	84	268	35	0.0	**1225**			1	0.0
instance033	480	1340	200	108	348	48	0.1	**28803**			1	0.1
instance035	543	1454	250	46	146	19	0.0	**114650399**			1	0.0
instance037	1172	3254	500	0	0	0	0.2	**160586161**			1	0.2
instance039	1912	4446	1173	17	48	10	0.0	**53301**			1	0.0
instance041	2382	5348	1889	70	228	27	0.0	**295208**			1	0.0
instance043	246	936	25	0	0	0	0.0	**2500540**			1	0.0
instance045	389	1124	150	126	404	49	0.1	**25700**			1	0.1
instance047	585	1598	250	33	102	18	0.0	**115277351**			1	0.0
instance049	623	1752	250	93	298	39	0.1	**116609813**			1	0.1

cont. next page

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	t [s]	Dual	Primal	Gap %	N	t [s]
instance051	1416	3956	657	150	474	63	0.4	**471589**			1	0.5
instance053	181	578	51	113	374	39	0.0	**409**			1	0.1
instance055	233	862	22	186	686	20	0.0	**2200557**			1	0.1
instance057	245	908	20	0	0	0	0.0	**2000271**			1	0.0
instance059	560	1740	195	248	814	96	0.2	**2386**			1	0.3
instance061	528	2034	85	509	1958	85	0.5	**8500739**			1	16.7
instance063	1214	3400	500	168	536	68	0.4	**160124233**			1	0.7
instance065	2856	7282	1748	196	614	91	0.5	**3194670**			1	0.6
instance067	200	740	20	0	0	0	0.0	**39067**			1	0.0
instance069	200	740	100	0	0	0	0.0	**86268**			1	0.0
instance071	339	1044	100	0	0	0	0.1	**75176630**			1	0.1
instance073	262	1480	10	0	0	0	0.0	**2146**			1	0.0
instance075	557	2160	49	145	536	19	0.6	**4901968**			1	0.7
instance077	2413	6824	1577	94	320	48	0.1	**19825626**			1	0.2
instance079	36415	291270	16808	36415	291270	16808	50.3	21950.4867	25210	14.8	1	>1620.0
instance081	237	756	76	69	222	28	0.1	**513**			1	0.1
instance083	311	1264	32	246	998	29	0.1	**3200554**			1	3.6
instance085	1274	3616	500	113	366	46	0.4	**161674316**			1	0.4
instance087	1337	3866	500	97	318	34	0.6	**162664661**			1	0.6
instance089	5829	15104	3038	122	374	49	0.5	**131895**			1	0.5
instance091	304	1142	33	101	368	15	0.0	**3300655**			1	0.0
instance093	1001	2838	400	117	376	48	0.3	**1490972010**			1	0.4
instance095	1335	3864	500	122	370	56	0.4	**164685074**			1	0.4
instance097	2629	7586	1000	399	1256	166	1.6	**227886471**			1	1.8
instance099	1294	3706	500	312	990	133	0.5	**162100435**			1	0.7
instance101	2778	8166	1000	491	1600	189	0.9	**230200846**			1	1.2
instance103	1055	2946	493	74	220	39	0.1	**320137**			1	0.1
instance105	1408	4112	500	243	804	88	0.3	**160756854**			1	0.5
instance107	160	480	24	0	0	0	0.0	**7068**			1	0.0
instance109	257	1030	23	131	478	18	0.0	**2300376**			1	0.0
instance111	2763	8076	1000	217	730	69	1.0	**231605619**			1	2.0
instance113	80	320	16	0	0	0	0.0	**4354**			1	0.0
instance115	243	994	27	75	272	16	0.1	**2700441**			1	0.1
instance117	398	1576	44	156	590	27	0.3	**4401504**			113	1.0
instance119	678	2060	318	70	214	35	0.1	**39335**			1	0.1
instance121	1005	3462	18	0	0	0	0.0	**780**			1	0.0
instance123	2762	8094	1000	728	2356	283	1.1	**227807756**			1	1.5
instance125	160	480	24	0	0	0	0.0	**6923**			1	0.0
instance127	294	1136	22	114	416	15	0.1	**2200394**			1	0.1
instance129	323	1184	31	0	0	0	0.1	**3100635**			1	0.1
instance131	499	1722	16	0	0	0	0.0	**594**			1	0.0
instance133	766	3070	76	145	570	16	0.4	**7602040**			1	0.5
instance135	2532	7230	1000	169	544	68	0.7	**228031092**			1	0.8
instance137	2676	7788	1000	784	2538	305	1.1	**228330602**			1	1.7
instance139	2984	8968	1000	232	768	80	2.9	**230904712**			1	3.6
instance141	294	1236	38	48	154	15	0.1	**3800606**			1	0.1
instance143	388	1630	45	0	0	0	0.3	**4500728**			1	0.3
instance145	437	1676	33	190	700	20	0.0	**3300513**			1	0.0
instance147	564	2224	50	159	586	22	0.6	**5001625**			1	0.7
instance149	615	2464	53	361	1392	48	0.9	**5301351**			1	3.5
instance151	314	1300	34	0	0	0	0.1	**3400525**			1	0.1
instance153	670	2632	62	588	2286	62	1.5	**6201016**			1	15.1
instance155	770	2766	21	115	384	14	0.0	**750**			1	0.0
instance157	938	3738	75	236	888	30	2.0	**7501712**			1	2.2
instance159	2865	8534	1000	564	1834	219	1.6	**230535806**			1	2.1
instance161	2502	12488	100	1991	8880	90	1.2	**79714**			11	9.9
instance163	402	1560	26	0	0	0	0.0	**2600484**			1	0.0

cont. next page

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	Presolved t [s]	Dual	Primal	Gap %	N	t [s]
instance165	422	1616	25	152	552	10	0.0	**2500828**			1	0.1
instance167	632	2174	10	0	0	0	0.0	**580**			1	0.0
instance169	720	2538	11	0	0	0	0.0	**508**			1	0.0
instance171	802	3106	39	128	456	9	0.1	**3903734**			1	0.1
instance173	788	3876	50	163	682	19	0.4	**45852**			1	0.4
instance175	2397	7430	783	982	3252	367	1.1	**8883**			1	2.8
instance177	904	3612	59	94	318	14	0.3	**5901592**			1	0.3
instance179	752	2528	26	0	0	0	0.0	**806**			1	0.0
instance181	757	2986	58	354	1396	40	0.8	**5801466**			1	3.6
instance183	838	3526	60	764	3190	60	4.2	**6001164**			1	152.6
instance185	2834	8414	1000	663	2172	247	1.9	**230639115**			1	2.6
instance187	529	2064	21	212	788	14	0.1	**2103283**			1	0.1
instance189	467	1792	30	89	322	14	0.1	**3000569**			1	0.1
instance191	5096	16210	1379	2354	7884	745	5.0	**56207**			1	36.3
instance193	1848	6572	16	0	0	0	0.0	**914**			1	0.0
instance195	1724	5950	29	0	0	0	0.0	**1244**			1	0.0
instance197	6128	30528	200	5279	24204	168	4.0	**111005**			1	370.6
instance199	1011	4020	37	866	3432	37	3.6	**3700485**			1	5.0

B.1.2 SteinLib instances

The time limit for the following instances is 54340 seconds. This corresponds to 24 hours on the machine used by Polzin and Vahdati-Daneshmand (2014).

Table B.3. Detailed computational results for SPG, test-set 2R.

	Original			Presolved						
Instance	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]	Optimum	N	t [s]
2r111	2000	11600	9	0	0	0	0.1	**28000**	1	0.1
2r112	2000	11600	9	0	0	0	0.1	**32000**	1	0.1
2r113	2000	11600	9	0	0	0	0.1	**28000**	1	0.1
2r121	2000	11532	9	0	0	0	0.1	**28000**	1	0.1
2r122	2000	11544	9	0	0	0	0.1	**29000**	1	0.1
2r123	2000	11508	9	0	0	0	0.1	**25000**	1	0.1
2r131	2000	11452	9	0	0	0	0.1	**27000**	1	0.1
2r132	2000	11450	9	636	5426	9	0.4	**33000**	1	0.4
2r133	2000	11458	9	0	0	0	0.1	**29000**	1	0.1
2r211	2000	11600	50	571	4988	34	2.8	**89000**	3	8.2
2r212	2000	11600	49	132	818	17	0.9	**80000**	1	1.1
2r213	2000	11600	48	279	2104	29	2.0	**76000**	1	2.5
2r221	2000	11528	50	0	0	0	1.1	**83000**	1	1.1
2r222	2000	11530	50	0	0	0	1.9	**84000**	1	1.9
2r223	2000	11540	49	562	4606	40	1.8	**84000**	1	4.1
2r231	2000	11474	50	0	0	0	2.3	**86000**	1	2.3
2r232	2000	11466	49	453	3358	37	2.0	**87000**	1	3.7
2r233	2000	11460	47	0	0	0	1.3	**83000**	1	1.3
2r311	2000	11600	95	372	2586	47	1.2	**129000**	1	2.0
2r312	2000	11600	92	482	3802	45	1.0	**126000**	1	2.4
2r313	2000	11600	97	306	2124	38	1.0	**128000**	1	2.0
2r321	2000	11542	92	0	0	0	0.3	**125000**	1	0.3
2r322	2000	11506	92	397	2784	43	1.6	**130000**	1	2.5
2r323	2000	11528	96	651	4856	64	1.7	**142000**	52	13.3
2r331	2000	11472	93	260	1584	40	1.8	**134000**	1	2.0
2r332	2000	11490	95	544	3820	50	1.7	**136000**	1	4.8
2r333	2000	11482	98	449	2938	54	2.1	**143000**	1	3.1

Table B.4. Detailed computational results for SPG, test-set LIN.

	Original			Presolved								
Instance	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]	Dual	Primal	Gap %	N	t [s]
lin01	53	160	4	0	0	0	0.0	**503**			1	0.0
lin02	55	164	6	0	0	0	0.0	**557**			1	0.0
lin03	57	168	8	0	0	0	0.0	**926**			1	0.0
lin04	157	532	6	0	0	0	0.0	**1239**			1	0.0
lin05	160	538	9	0	0	0	0.0	**1703**			1	0.0
lin06	165	548	14	0	0	0	0.0	**1348**			1	0.0
lin07	307	1052	6	0	0	0	0.0	**1885**			1	0.0
lin08	311	1060	10	0	0	0	0.0	**2248**			1	0.0
lin09	313	1064	12	0	0	0	0.0	**2752**			1	0.0
lin10	321	1080	20	0	0	0	0.0	**4132**			1	0.0
lin11	816	2920	10	0	0	0	0.0	**4280**			1	0.0
lin12	818	2924	12	47	144	8	0.0	**5250**			1	0.0
lin13	822	2932	16	0	0	0	0.0	**4609**			1	0.0
lin14	828	2944	22	0	0	0	0.0	**5824**			1	0.0
lin15	840	2968	34	0	0	0	0.0	**7145**			1	0.0
lin16	1981	7266	12	0	0	0	0.1	**6618**			1	0.1
lin17	1989	7282	20	0	0	0	0.1	**8405**			1	0.1
lin18	1994	7292	25	13	34	6	0.5	**9714**			1	0.5
lin19	2010	7324	41	105	350	11	0.1	**13268**			1	0.1

cont. next page

	Original			Presolved								
Instance	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]	Dual	Primal	Gap %	N	t [s]
lin20	3675	13418	11	0	0	0	0.1	**6673**			1	0.1
lin21	3683	13434	20	129	422	9	0.2	**9143**			1	0.2
lin22	3692	13452	28	0	0	0	0.2	**10519**			1	0.2
lin23	3716	13500	52	84	278	17	0.8	**17560**			1	0.8
lin24	7998	29468	16	2970	10706	16	1.4	**15076**			1	1.5
lin25	8007	29486	24	931	3254	20	4.4	**17803**			1	4.4
lin26	8013	29498	30	0	0	0	3.4	**21757**			1	3.4
lin27	8017	29506	36	94	302	18	3.3	**20678**			1	3.3
lin28	8062	29596	81	354	1200	44	11.8	**32584**			1	12.0
lin29	19083	71272	24	1022	3714	19	12.0	**23765**			1	12.0
lin30	19091	71288	31	190	650	19	15.9	**27684**			1	15.9
lin31	19100	71306	40	6577	24148	37	39.7	**31696**			1	44.4
lin32	19112	71330	53	6902	25308	52	49.1	**39832**			1	66.7
lin33	19177	71460	117	5711	20688	97	52.4	**56061**			1	540.8
lin34	38282	143042	34	11461	42424	33	157.1	**45018**			1	158.2
lin35	38294	143066	45	14095	51962	45	99.1	**50559**			1	120.3
lin36	38307	143092	58	17931	66096	57	102.2	**55608**			1	259.9
lin37	38418	143314	172	23971	88916	169	129.2	97134.5228	99773	2.7	1	>54340

Table B.5. Detailed computational results for SPG, test-set PUC.

	Original			Presolved								
Instance	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]	Dual	Primal	Gap %	N	t [s]
bip42p	1200	7964	200	990	7234	200	0.2	**24657**			134338	18296.0
bip42u	1200	7964	200	989	7216	200	0.2	**236**			98275	10918.0
bip52p	2200	15994	200	1817	14650	200	0.4	24317.3892	24563	1.0	263924	>54340
bip52u	2200	15994	200	1818	14646	200	0.6	230.569282	234	1.5	274715	>54340
bip62p	1200	20004	200	1199	20000	200	0.3	22589.1452	22882	1.3	168402	>54340
bip62u	1200	20004	200	1199	20000	200	0.5	214.924104	219	1.9	122591	>54340
bipa2p	3300	36146	300	3139	35588	300	1.0	34789.4576	35427	1.8	91345	>54340
bipa2u	3300	36146	300	3138	35590	300	1.2	330.629545	340	2.8	118047	>54340
bipe2p	550	10026	50	550	10026	50	0.1	**5616**			1778	156.6
bipe2u	550	10026	50	550	10026	50	0.2	**54**			121	75.1
cc10-2p	1024	10240	135	1024	10240	135	0.4	34531.666	35235	2.0	4227	>54340
cc10-2u	1024	10240	135	1024	10240	135	0.8	334.695705	344	2.8	3902	>54340
cc11-2p	2048	22526	244	2048	22526	244	1.3	62119.5137	63775	2.7	1606	>54340
cc11-2u	2048	22526	244	2048	22526	244	1.9	602.628793	617	2.4	2794	>54340
cc12-2p	4096	49148	473	4096	49148	473	4.4	118630.032	121609	2.5	99	>54340
cc12-2u	4096	49148	473	4096	49148	473	4.3	1150.13492	1180	2.6	267	>54340
cc3-10p	1000	27000	50	1000	27000	50	0.6	12701.8929	12774	0.6	7970	>54340
cc3-10u	1000	27000	50	1000	27000	50	1.0	120.24651	126	4.8	540	>54340
cc3-11p	1331	39930	61	1331	39930	61	1.0	15463.0715	15600	0.9	4291	>54340
cc3-11u	1331	39930	61	1331	39930	61	1.1	144.304692	154	6.7	1	>54340
cc3-12p	1728	57024	74	1728	57024	74	1.5	18735.8687	18847	0.6	3835	>54340
cc3-12u	1728	57024	74	1728	57024	74	1.7	174.415419	186	6.6	65	>54340
cc3-4p	64	576	8	64	576	8	0.0	**2338**			1	0.0
cc3-4u	64	576	8	64	576	8	0.0	**23**			1	0.0
cc3-5p	125	1500	13	125	1500	13	0.0	**3661**			1	0.8
cc3-5u	125	1500	13	125	1500	13	0.0	**36**			1	0.8
cc5-3p	243	2430	27	243	2430	27	0.1	**7299**			1617	3908.1
cc5-3u	243	2430	27	243	2430	27	0.1	**71**			195	631.5
cc6-2p	64	384	12	64	384	12	0.0	**3271**			1	0.1
cc6-2u	64	384	12	64	384	12	0.0	**32**			1	0.2
cc6-3p	729	8736	76	729	8736	76	0.3	20195.8499	20285	0.4	21389	>54340
cc6-3u	729	8736	76	729	8736	76	0.6	**197**			480	24751.0
cc7-3p	2187	30616	222	2187	30616	222	1.7	55409.2249	57103	3.1	693	>54340
cc7-3u	2187	30616	222	2187	30616	222	1.8	536.649655	553	3.0	803	>54340
cc9-2p	512	4608	64	512	4608	64	0.2	16910.9215	17221	1.8	7616	>54340

cont. next page

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Dual	Primal	Gap %	N	t [s]
cc9-2u	512	4608	64	512	4608	64	0.3	163.900276	168	2.5	7493	>54340
hc10p	1024	10240	512	1024	10240	512	0.4	59290.8921	59840	0.9	45960	>54340
hc10u	1024	10240	512	1024	10240	512	0.8	567.888889	576	1.4	64217	>54340
hc11p	2048	22528	1024	2048	22528	1024	1.2	117451.059	119795	2.0	14368	>54340
hc11u	2048	22528	1024	2048	22528	1024	2.1	1125.4	1158	2.9	12159	>54340
hc12p	4096	49152	2048	4096	49152	2048	4.7	232922.491	236506	1.5	1720	>54340
hc12u	4096	49152	2048	4096	49152	2048	7.5	2233.09091	2301	3.0	370	>54340
hc6p	64	384	32	64	384	32	0.0	**4003**			1497	16.1
hc6u	64	384	32	64	384	32	0.0	**39**			695	5.9
hc7p	128	896	64	128	896	64	0.0	**7905**			260991	3550.8
hc7u	128	896	64	128	896	64	0.1	**77**			597413	5143.4
hc8p	256	2048	128	256	2048	128	0.1	**15322**			307517	23599.8
hc8u	256	2048	128	256	2048	128	0.1	145.714286	148	1.6	1069424	>54340
hc9p	512	4608	256	512	4608	256	0.2	29982.8096	30252	0.9	152094	>54340
hc9u	512	4608	256	512	4608	256	0.3	287.125	292	1.7	374988	>54340

Table B.6. Detailed computational results for SPG, test-set SP.

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Optimum	N	t [s]
antiwheel5	10	30	5	0	0	0	0.0	**7**	1	0.0
design432	8	40	4	0	0	0	0.0	**9**	1	0.0
oddcycle3	6	18	3	0	0	0	0.0	**4**	1	0.0
oddwheel3	7	18	4	0	0	0	0.0	**5**	1	0.0
se03	13	42	4	0	0	0	0.0	**12**	1	0.0
w13c29	783	4524	406	783	4524	406	0.4	**507**	62	95.7
w23c23	1081	6348	552	1081	6348	552	0.7	**689**	62	1061.8
w3c571	3997	20556	2284	3997	20556	2284	3.1	**2854**	1	54.9

Table B.7. Detailed computational results for SPG, test-set TSPFST.

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Optimum	N	t [s]
a280fst	313	656	279	0	0	0	0.0	**2502**	1	0.0
att48fst	139	404	48	58	186	23	0.0	**30236**	1	0.0
att532fst	1468	4304	532	270	854	103	0.4	**84009**	1	0.5
berlin52fst	89	208	52	0	0	0	0.0	**6760**	1	0.0
bier127fst	258	714	127	0	0	0	0.0	**104284**	1	0.0
d1291fst	1365	2912	1291	0	0	0	0.0	**481421**	1	0.0
d1655fst	1906	4166	1655	33	100	16	0.0	**584948**	1	0.0
d198fst	232	512	198	0	0	0	0.0	**129175**	1	0.0
d2103fst	2206	4544	2103	0	0	0	0.0	**769797**	1	0.0
d493fst	1055	2946	493	92	280	46	0.1	**320137**	1	0.2
d657fst	1416	3956	657	131	418	56	0.3	**471589**	1	0.7
dsj1000fst	2562	7310	1000	69	220	28	0.2	**17564659**	1	0.3
eil101fst	330	1076	101	166	550	56	0.1	**605**	1	0.2
eil51fst	181	578	51	114	376	39	0.0	**409**	1	0.1
eil76fst	237	756	76	92	302	35	0.1	**513**	1	0.1
fl1400fst	2694	9092	1400	441	1648	221	0.6	**17980523**	1	7.9
fl1577fst	2413	6824	1577	94	320	48	0.1	**19825626**	1	0.3
fl3795fst	4859	13078	3795	444	1656	222	3.2	**25529856**	1	8.7
fl417fst	732	2168	417	124	438	52	0.1	**10883190**	1	0.2
fnl4461fst	17127	54704	4461	7720	25748	2484	25.2	**182361**	55	165.5
gil262fst	537	1446	262	34	104	22	0.0	**2306**	1	0.0
kroA100fst	197	500	100	0	0	0	0.0	**20401**	1	0.0
kroA150fst	389	1124	150	126	404	49	0.1	**25700**	1	0.1
kroA200fst	500	1428	200	0	0	0	0.0	**28652**	1	0.0

cont. next page

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	Presolved t [s]	Optimum	N	t [s]
kroB100fst	230	626	100	0	0	0	0.0	**21211**	1	0.0
kroB150fst	420	1238	150	62	204	25	0.1	**25217**	1	0.1
kroB200fst	480	1340	200	102	330	45	0.1	**28803**	1	0.1
kroC100fst	244	674	100	45	146	18	0.0	**20492**	1	0.0
kroD100fst	216	576	100	12	34	7	0.0	**20437**	1	0.0
kroE100fst	226	612	100	46	140	21	0.0	**21245**	1	0.0
lin105fst	216	646	105	43	134	20	0.0	**13429**	1	0.0
lin318fst	678	2060	318	78	252	38	0.1	**39335**	1	0.1
linhp318fst	678	2060	318	78	252	38	0.1	**39335**	1	0.1
nrw1379fst	5096	16210	1379	2370	7936	751	4.6	**56207**	1	16.5
p654fst	777	1734	654	0	0	0	0.0	**314925**	1	0.0
pcb1173fst	1912	4446	1173	17	48	10	0.0	**53301**	1	0.0
pcb3038fst	5829	15104	3038	77	232	40	0.4	**131895**	1	0.5
pcb442fst	503	1062	442	0	0	0	0.0	**47675**	1	0.0
pla7397fst	8790	19630	7397	97	296	51	0.1	**22481625**	1	0.1
pr1002fst	1473	3430	1002	0	0	0	0.0	**243176**	1	0.0
pr107fst	111	220	107	0	0	0	0.0	**34850**	1	0.0
pr124fst	154	330	124	0	0	0	0.0	**52759**	1	0.0
pr136fst	196	500	136	0	0	0	0.0	**86811**	1	0.0
pr144fst	221	570	144	0	0	0	0.0	**52925**	1	0.0
pr152fst	308	862	152	0	0	0	0.0	**64323**	1	0.0
pr226fst	255	538	226	0	0	0	0.0	**70700**	1	0.0
pr2392fst	3398	7932	2392	0	0	0	0.0	**358989**	1	0.0
pr264fst	280	574	264	0	0	0	0.0	**41400**	1	0.0
pr299fst	420	1000	299	0	0	0	0.0	**44671**	1	0.0
pr439fst	572	1324	439	0	0	0	0.0	**97400**	1	0.0
pr76fst	168	494	76	33	94	16	0.0	**95908**	1	0.0
rat195fst	560	1740	195	182	594	70	0.2	**2386**	1	0.3
rat575fst	1986	6352	575	892	2940	319	2.0	**6808**	1	2.7
rat783fst	2397	7430	783	900	2976	338	2.2	**8883**	1	3.1
rat99fst	269	798	99	0	0	0	0.0	**1225**	1	0.0
rd100fst	201	506	100	0	0	0	0.0	**764269099**	1	0.0
rd400fst	1001	2838	400	161	514	65	0.1	**1490972010**	1	0.2
rl11849fst	13963	30630	11849	88	266	50	0.1	**8779590**	1	0.1
rl1304fst	1562	3388	1304	18	54	10	0.0	**236649**	1	0.0
rl1323fst	1598	3500	1323	0	0	0	0.0	**253620**	1	0.0
rl1889fst	2382	5348	1889	67	216	27	0.0	**295208**	1	0.0
rl5915fst	6569	13960	5915	34	102	19	0.0	**533226**	1	0.0
rl5934fst	6827	14730	5934	31	100	19	0.0	**529890**	1	0.0
st70fst	133	338	70	0	0	0	0.0	**626**	1	0.0
ts225fst	225	448	225	0	0	0	0.0	**1120**	1	0.0
tsp225fst	242	504	225	0	0	0	0.0	**356850**	1	0.0
u1060fst	1835	4858	1060	63	220	32	0.1	**21265372**	1	0.6
u1432fst	1432	2862	1432	0	0	0	0.0	**1465**	1	0.0
u159fst	184	372	159	0	0	0	0.0	**390**	1	0.0
u1817fst	1831	3692	1817	0	0	0	0.0	**5513053**	1	0.0
u2152fst	2167	4368	2152	0	0	0	0.0	**6253305**	1	0.0
u2319fst	2319	4636	2319	0	0	0	0.0	**2322**	1	0.0
u574fst	990	2516	574	0	0	0	0.1	**3509275**	1	0.1
u724fst	1180	3074	724	11	32	7	0.0	**4069628**	1	0.1
vm1084fst	1679	4116	1084	26	80	15	0.1	**2248390**	1	0.1
vm1748fst	2856	7282	1748	191	612	85	0.5	**3194670**	1	0.6

Table B.8. Detailed computational results for SPG, test-set VLSI.

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	Presolved t [s]	Optimum	N	t [s]
alue2087	1244	3942	34	0	0	0	0.0	**1049**	1	0.0
alue2105	1220	3716	34	0	0	0	0.0	**1032**	1	0.0
alue3146	3626	11738	64	0	0	0	0.1	**2240**	1	0.1
alue5067	3524	11120	68	50	146	16	0.1	**2586**	1	0.1
alue5345	5179	16330	68	62	202	17	0.9	**3507**	1	0.9

cont. next page

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	Presolved t [s]	Optimum	N	t [s]
alue5623	4472	13876	68	20	56	9	0.6	**3413**	1	0.6
alue5901	11543	36858	68	41	124	18	0.7	**3912**	1	0.7
alue6179	3372	10426	67	0	0	0	0.1	**2452**	1	0.1
alue6457	3932	12274	68	0	0	0	0.1	**3057**	1	0.1
alue6735	4119	13392	68	140	446	20	0.1	**2696**	1	0.1
alue6951	2818	8838	67	57	172	19	0.1	**2386**	1	0.1
alue7065	34046	109682	544	41	122	17	19.5	**23881**	1	19.5
alue7066	6405	20908	16	2281	7962	9	0.7	**2256**	1	0.8
alue7080	34479	110988	2344	862	2736	344	9.8	**62449**	1	10.3
alue7229	940	2948	34	0	0	0	0.0	**824**	1	0.0
alut0787	1160	4178	34	0	0	0	0.0	**982**	1	0.0
alut0805	966	3332	34	0	0	0	0.0	**958**	1	0.0
alut1181	3041	11386	64	0	0	0	0.2	**2353**	1	0.2
alut2010	6104	22022	68	52	162	13	0.3	**3307**	1	0.3
alut2288	9070	33190	68	0	0	0	0.9	**3843**	1	0.9
alut2566	5021	18110	68	32	96	14	0.7	**3073**	1	0.7
alut2610	33901	125632	204	119	408	9	41.4	**12239**	1	41.4
alut2625	36711	136234	879	2875	10224	426	45.1	**35459**	1	52.4
alut2764	387	1252	34	0	0	0	0.0	**640**	1	0.0
diw0234	5349	20172	25	0	0	0	0.1	**1996**	1	0.1
diw0250	353	1216	11	0	0	0	0.0	**350**	1	0.0
diw0260	539	1970	12	0	0	0	0.0	**468**	1	0.0
diw0313	468	1644	14	0	0	0	0.0	**397**	1	0.0
diw0393	212	762	11	0	0	0	0.0	**302**	1	0.0
diw0445	1804	6622	33	21	60	12	0.1	**1363**	1	0.1
diw0459	3636	13578	25	14	40	5	0.1	**1362**	1	0.1
diw0460	339	1158	13	0	0	0	0.0	**345**	1	0.0
diw0473	2213	8270	25	0	0	0	0.1	**1098**	1	0.1
diw0487	2414	8772	25	0	0	0	0.0	**1424**	1	0.0
diw0495	938	3310	10	0	0	0	0.0	**616**	1	0.0
diw0513	918	3368	10	0	0	0	0.0	**604**	1	0.0
diw0523	1080	4030	10	0	0	0	0.0	**561**	1	0.0
diw0540	286	930	10	0	0	0	0.0	**374**	1	0.0
diw0559	3738	14026	18	171	608	12	0.2	**1570**	1	0.2
diw0778	7231	27454	24	0	0	0	0.6	**2173**	1	0.6
diw0779	11821	45032	50	32	100	8	2.7	**4440**	1	2.7
diw0795	3221	11876	10	0	0	0	0.1	**1550**	1	0.1
diw0801	3023	11150	10	0	0	0	0.1	**1587**	1	0.1
diw0819	10553	40132	32	0	0	0	0.2	**3399**	1	0.2
diw0820	11749	44768	37	88	310	12	3.8	**4167**	1	3.9
dmxa0296	233	772	12	0	0	0	0.0	**344**	1	0.0
dmxa0368	2050	7352	18	16	40	10	0.1	**1017**	1	0.1
dmxa0454	1848	6572	16	0	0	0	0.0	**914**	1	0.0
dmxa0628	169	560	10	0	0	0	0.0	**275**	1	0.0
dmxa0734	663	2308	11	0	0	0	0.0	**506**	1	0.0
dmxa0848	499	1722	16	0	0	0	0.0	**594**	1	0.0
dmxa0903	632	2174	10	0	0	0	0.0	**580**	1	0.0
dmxa1010	3983	14216	23	0	0	0	0.1	**1488**	1	0.1
dmxa1109	343	1118	17	0	0	0	0.0	**454**	1	0.0
dmxa1200	770	2766	21	33	94	13	0.0	**750**	1	0.0
dmxa1304	298	1006	10	0	0	0	0.0	**311**	1	0.0
dmxa1516	720	2538	11	0	0	0	0.0	**508**	1	0.0
dmxa1721	1005	3462	18	0	0	0	0.0	**780**	1	0.0
dmxa1801	2333	8274	17	209	716	16	0.1	**1365**	1	0.1
gap1307	342	1104	17	0	0	0	0.0	**549**	1	0.0
gap1413	541	1812	10	0	0	0	0.0	**457**	1	0.0
gap1500	220	748	17	0	0	0	0.0	**254**	1	0.0

cont. next page

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	Presolved t [s]	Optimum	N	t [s]
gap1810	429	1404	17	0	0	0	0.0	**482**	1	0.0
gap1904	735	2512	21	0	0	0	0.0	**763**	1	0.0
gap2007	2039	7096	17	0	0	0	0.0	**1104**	1	0.0
gap2119	1724	5950	29	0	0	0	0.0	**1244**	1	0.0
gap2740	1196	4168	14	0	0	0	0.0	**745**	1	0.0
gap2800	386	1306	12	0	0	0	0.0	**386**	1	0.0
gap2975	179	586	10	0	0	0	0.0	**245**	1	0.0
gap3036	346	1166	13	0	0	0	0.0	**457**	1	0.0
gap3100	921	3116	11	0	0	0	0.0	**640**	1	0.0
gap3128	10393	36086	104	0	0	0	0.2	**4292**	1	0.2
msm0580	338	1082	11	0	0	0	0.0	**467**	1	0.0
msm0654	1290	4540	10	0	0	0	0.0	**823**	1	0.0
msm0709	1442	4806	16	0	0	0	0.0	**884**	1	0.0
msm0920	752	2528	26	0	0	0	0.0	**806**	1	0.0
msm1008	402	1390	11	0	0	0	0.0	**494**	1	0.0
msm1234	933	3264	13	0	0	0	0.0	**550**	1	0.0
msm1477	1199	4156	31	0	0	0	0.0	**1068**	1	0.0
msm1707	278	956	11	0	0	0	0.0	**564**	1	0.0
msm1844	90	270	10	0	0	0	0.0	**188**	1	0.0
msm1931	875	3044	10	0	0	0	0.0	**604**	1	0.0
msm2000	898	3124	10	0	0	0	0.0	**594**	1	0.0
msm2152	2132	7404	37	0	0	0	0.1	**1590**	1	0.1
msm2326	418	1446	14	0	0	0	0.0	**399**	1	0.0
msm2492	4045	14188	12	0	0	0	0.1	**1459**	1	0.1
msm2525	3031	10478	12	0	0	0	0.1	**1290**	1	0.1
msm2601	2961	10200	16	0	0	0	0.1	**1440**	1	0.1
msm2705	1359	4916	13	0	0	0	0.0	**714**	1	0.0
msm2802	1709	5926	18	0	0	0	0.0	**926**	1	0.0
msm2846	3263	11566	89	52	162	22	0.3	**3135**	1	0.3
msm3277	1704	5982	12	0	0	0	0.0	**869**	1	0.0
msm3676	957	3108	10	0	0	0	0.0	**607**	1	0.0
msm3727	4640	16510	21	0	0	0	0.1	**1376**	1	0.1
msm3829	4221	14510	12	0	0	0	0.3	**1571**	1	0.3
msm4038	237	780	11	0	0	0	0.0	**353**	1	0.0
msm4114	402	1380	16	0	0	0	0.0	**393**	1	0.0
msm4190	391	1332	16	0	0	0	0.0	**381**	1	0.0
msm4224	191	604	11	0	0	0	0.0	**311**	1	0.0
msm4312	5181	17786	10	672	2332	10	0.5	**2016**	1	0.5
msm4414	317	952	11	0	0	0	0.0	**408**	1	0.0
msm4515	777	2716	13	0	0	0	0.0	**630**	1	0.0
taq0014	6466	22092	128	0	0	0	0.5	**5326**	1	0.5
taq0023	572	1926	11	0	0	0	0.0	**621**	1	0.0
taq0365	4186	14148	22	61	198	9	0.1	**1914**	1	0.1
taq0377	6836	23430	136	58	160	34	1.6	**6393**	1	1.7
taq0431	1128	3810	13	0	0	0	0.0	**897**	1	0.0
taq0631	609	1864	10	0	0	0	0.0	**581**	1	0.0
taq0739	837	2876	16	0	0	0	0.0	**848**	1	0.0
taq0741	712	2434	16	53	170	9	0.0	**847**	1	0.0
taq0751	1051	3582	16	0	0	0	0.0	**939**	1	0.0
taq0891	331	1120	10	0	0	0	0.0	**319**	1	0.0
taq0903	6163	20980	130	0	0	0	1.4	**5099**	1	1.5
taq0910	310	1028	17	0	0	0	0.0	**370**	1	0.0
taq0920	122	388	17	0	0	0	0.0	**210**	1	0.0
taq0978	777	2478	10	0	0	0	0.0	**566**	1	0.0

Table B.9. Detailed computational results for SPG, test-set WRP3.

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	Presolved t [s]	Optimum	N	t [s]
wrp3-11	128	454	11	0	0	0	0.0	**1100361**	1	0.0
wrp3-12	84	298	12	0	0	0	0.0	**1200237**	1	0.0
wrp3-13	311	1226	13	131	492	13	0.1	**1300497**	1	0.1
wrp3-14	128	494	14	108	422	13	0.0	**1400250**	1	0.0
wrp3-15	138	514	15	0	0	0	0.0	**1500422**	1	0.0
wrp3-16	204	748	16	0	0	0	0.0	**1600208**	1	0.0
wrp3-17	177	708	17	151	622	13	0.0	**1700442**	1	0.0
wrp3-19	189	706	19	0	0	0	0.0	**1900439**	1	0.0
wrp3-20	245	908	20	0	0	0	0.0	**2000271**	1	0.0
wrp3-21	237	888	21	0	0	0	0.0	**2100522**	1	0.0
wrp3-22	233	862	22	186	686	20	0.0	**2200557**	1	0.1
wrp3-23	132	460	23	0	0	0	0.0	**2300245**	1	0.0
wrp3-24	262	974	24	120	416	17	0.0	**2400623**	1	0.0
wrp3-25	246	936	25	0	0	0	0.0	**2500540**	1	0.0
wrp3-26	402	1560	26	0	0	0	0.0	**2600484**	1	0.0
wrp3-27	370	1442	27	58	202	14	0.2	**2700502**	1	0.2
wrp3-28	307	1118	28	2	2	1	0.0	**2800379**	1	0.0
wrp3-29	245	872	29	0	0	0	0.0	**2900479**	1	0.0
wrp3-30	467	1792	30	73	248	14	0.1	**3000569**	1	0.1
wrp3-31	323	1184	31	55	188	16	0.1	**3100635**	1	0.1
wrp3-33	437	1676	33	100	382	13	0.0	**3300513**	1	0.0
wrp3-34	1244	4948	34	1057	4206	32	2.4	**3400646**	1	3.6
wrp3-36	435	1636	36	99	332	15	0.4	**3600610**	1	0.4
wrp3-37	1011	4020	37	847	3356	37	3.2	**3700485**	1	4.7
wrp3-38	603	2414	38	437	1780	37	1.0	**3800656**	1	2.0
wrp3-39	703	3232	39	609	2822	38	2.4	**3900450**	1	4.2
wrp3-41	178	614	41	129	448	36	0.2	**4100466**	1	0.2
wrp3-42	705	2746	42	572	2214	41	0.8	**4200598**	1	1.3
wrp3-43	173	596	43	0	0	0	0.1	**4300457**	1	0.1
wrp3-45	1414	5626	45	1204	4786	45	2.9	**4500860**	1	3.3
wrp3-48	925	3476	48	491	1816	45	1.2	**4800552**	1	2.0
wrp3-49	886	3600	49	693	2798	46	1.8	**4900882**	1	9.9
wrp3-50	1119	4502	50	915	3716	49	2.5	**5000673**	1	4.3
wrp3-52	701	2704	52	581	2250	49	1.6	**5200825**	1	5.2
wrp3-53	775	2942	53	148	534	12	0.3	**5300847**	1	0.3
wrp3-55	1645	6372	55	1487	5844	55	2.1	**5500888**	1	69.4
wrp3-56	853	3180	56	590	2238	52	0.9	**5600872**	1	2.7
wrp3-60	838	3526	60	785	3300	60	2.2	**6001164**	1	28.2
wrp3-62	670	2632	62	586	2278	62	1.1	**6201016**	1	5.5
wrp3-64	1822	7220	64	1592	6402	59	3.4	**6400931**	1	9.9
wrp3-66	2521	9716	66	2269	8946	62	3.0	**6600922**	1	368.1
wrp3-67	987	3846	67	467	1848	36	1.8	**6700776**	1	3.5
wrp3-69	856	3242	69	447	1674	61	1.6	**6900841**	1	2.0
wrp3-70	1468	5862	70	964	3810	56	2.7	**7000890**	1	11.2
wrp3-71	1221	4828	71	947	3754	62	2.7	**7101028**	1	20.3
wrp3-73	1890	7226	73	1679	6534	63	2.2	**7301207**	1	37.4
wrp3-74	1019	3882	74	861	3326	65	1.1	**7400759**	1	13.1
wrp3-75	729	2790	75	551	2054	75	1.6	**7501020**	1	2.5
wrp3-76	1761	6740	76	1049	4066	46	3.1	**7601028**	1	4.7
wrp3-78	2346	9312	78	1993	7980	71	3.6	**7801094**	1	232.2
wrp3-79	833	3190	79	0	0	0	1.1	**7900444**	1	1.1
wrp3-80	1491	5662	80	1214	4650	75	3.6	**8000849**	1	29.7
wrp3-83	3168	12440	83	2961	11852	80	3.2	**8300906**	1	3073.2
wrp3-84	2356	9094	84	1915	7600	73	3.4	**8401094**	1	18.6
wrp3-85	528	2034	85	509	1958	85	0.5	**8500739**	1	7.7
wrp3-86	1360	5214	86	1157	4444	86	2.9	**86000746**	1	44.0
wrp3-88	743	2818	88	390	1470	58	1.6	**88001175**	1	2.4

cont. next page

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Optimum	N	t [s]
wrp3-91	1343	5188	91	873	3356	78	3.1	**91000866**	1	5.5
wrp3-92	1765	7226	92	1265	5254	70	3.5	**92000764**	1	36.5
wrp3-94	1976	7672	94	1504	6002	79	3.9	**94001181**	5	53.1
wrp3-96	2518	9970	96	2193	8800	87	3.9	**96001172**	1	185.4
wrp3-98	2265	9090	98	1893	7712	83	4.0	**98001224**	1	347.2
wrp3-99	2076	8144	99	1689	6612	94	2.0	**99001097**	1	118.1

Table B.10. Detailed computational results for SPG, test-set WRP4.

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Optimum	N	t [s]
wrp4-11	123	466	11	0	0	0	0.0	**1100179**	1	0.0
wrp4-13	110	376	13	0	0	0	0.0	**1300798**	1	0.0
wrp4-14	145	566	14	0	0	0	0.0	**1400290**	1	0.0
wrp4-15	193	738	15	0	0	0	0.0	**1500405**	1	0.0
wrp4-16	311	1158	16	0	0	0	0.0	**1601190**	1	0.0
wrp4-17	223	808	17	138	486	13	0.0	**1700525**	1	0.0
wrp4-18	211	760	18	0	0	0	0.0	**1801464**	1	0.0
wrp4-19	119	412	19	0	0	0	0.0	**1901446**	1	0.0
wrp4-21	529	2064	21	167	644	15	0.1	**2103283**	1	0.1
wrp4-22	294	1136	22	108	392	15	0.1	**2200394**	1	0.1
wrp4-23	257	1030	23	131	478	18	0.0	**2300376**	1	0.0
wrp4-24	493	1926	24	0	0	0	0.1	**2403332**	1	0.1
wrp4-25	422	1616	25	92	332	9	0.1	**2500828**	1	0.1
wrp4-26	396	1562	26	310	1224	26	0.5	**2600443**	1	1.7
wrp4-27	243	994	27	71	260	16	0.1	**2700441**	1	0.1
wrp4-28	272	1090	28	190	756	28	0.2	**2800466**	1	0.5
wrp4-29	247	1010	29	105	394	22	0.2	**2900484**	1	0.2
wrp4-30	361	1448	30	296	1190	29	0.1	**3000526**	1	2.2
wrp4-31	390	1572	31	318	1280	30	0.3	**3100526**	1	2.6
wrp4-32	311	1264	32	246	998	29	0.1	**3200554**	1	1.2
wrp4-33	304	1142	33	103	372	19	0.0	**3300655**	1	0.0
wrp4-34	314	1300	34	45	154	9	0.1	**3400525**	1	0.1
wrp4-35	471	1908	35	320	1240	35	0.4	**3500601**	1	1.2
wrp4-36	363	1500	36	310	1276	36	0.2	**3600596**	1	1.3
wrp4-37	522	2108	37	438	1726	37	0.4	**3700647**	1	2.9
wrp4-38	294	1236	38	0	0	0	0.1	**3800606**	1	0.1
wrp4-39	802	3106	39	163	600	14	0.1	**3903734**	1	0.1
wrp4-40	538	2176	40	440	1774	39	0.3	**4000758**	1	6.9
wrp4-41	465	1910	41	377	1540	41	0.4	**4100695**	1	4.1
wrp4-42	552	2262	42	502	2038	42	0.4	**4200701**	1	9.6
wrp4-43	596	2296	43	277	1054	33	0.1	**4301508**	1	0.2
wrp4-44	398	1576	44	153	576	27	0.3	**4401504**	9	0.4
wrp4-45	388	1630	45	0	0	0	0.3	**4500728**	1	0.3
wrp4-46	632	2574	46	583	2356	46	0.4	**4600756**	1	8.6
wrp4-47	555	2196	47	0	0	0	0.9	**4701318**	1	0.9
wrp4-48	451	1650	48	0	0	0	0.1	**4802220**	1	0.1
wrp4-49	557	2160	49	158	582	22	0.5	**4901968**	1	0.6
wrp4-50	564	2224	50	223	860	24	0.4	**5001625**	1	0.6
wrp4-51	668	2612	51	407	1592	45	1.3	**5101616**	1	1.6
wrp4-52	547	2230	52	70	240	20	0.4	**5201081**	1	0.4
wrp4-53	615	2464	53	351	1370	46	0.7	**5301351**	1	1.5
wrp4-54	688	2776	54	356	1398	40	0.6	**5401534**	1	1.4
wrp4-55	610	2402	55	403	1562	51	0.7	**5501952**	1	1.0
wrp4-56	839	3234	56	489	1902	47	0.8	**5602299**	1	1.6
wrp4-58	757	2986	58	367	1446	41	0.6	**5801466**	1	1.3
wrp4-59	904	3612	59	154	506	29	0.2	**5901592**	1	0.2

cont. next page

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	Presolved t [s]	Optimum	N	t [s]
wrp4-60	693	2740	60	103	346	24	0.4	**6001782**	1	0.4
wrp4-61	775	3076	61	138	500	19	0.2	**6102210**	1	0.2
wrp4-62	1283	4986	62	313	1184	29	2.6	**6202100**	1	2.7
wrp4-63	1121	4454	63	943	3752	60	0.9	**6301479**	1	57.1
wrp4-64	632	2562	64	0	0	0	0.3	**6401996**	1	0.3
wrp4-66	844	3382	66	229	834	24	1.0	**6602931**	1	1.0
wrp4-67	1518	6120	67	208	770	28	2.5	**6702800**	1	2.6
wrp4-68	917	3700	68	793	3182	67	0.8	**6801753**	1	4.3
wrp4-69	574	2330	69	0	0	0	0.7	**6902328**	1	0.7
wrp4-70	637	2538	70	0	0	0	0.1	**7003022**	1	0.1
wrp4-71	802	3218	71	0	0	0	0.1	**7102320**	1	0.1
wrp4-72	1151	4548	72	538	2132	48	1.1	**7202807**	1	4.2
wrp4-73	1898	7232	73	1290	5112	73	1.9	**7302643**	1	23.7
wrp4-74	802	3240	74	610	2422	72	0.8	**7402046**	1	1.9
wrp4-75	938	3738	75	702	2784	75	1.1	**7501712**	1	2.2
wrp4-76	766	3070	76	140	504	30	0.5	**7602040**	1	0.6

B.1.3 DIMACS 2014 instances

The time limit for the following instances is 54340 seconds. This corresponds to 24 hours on the machine used by Polzin and Vahdati-Daneshmand (2014).

Table B.11. Detailed computational results for SPG, test-set Copenhagen14.

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	Presolved t [s]	Optimum	N	t [s]
ind1	18	62	10	0	0	0	0.0	**604**	1	0.0
ind2	31	114	10	0	0	0	0.0	**9500**	1	0.0
ind3	16	46	10	0	0	0	0.0	**600**	1	0.0
ind4	74	292	25	0	0	0	0.0	**1086**	1	0.0
ind5	114	456	33	0	0	0	0.0	**1341**	1	0.0
rc01	21	70	10	0	0	0	0.0	**25980**	1	0.0
rc02	87	352	30	2	2	1	0.0	**41350**	1	0.0
rc03	109	404	50	0	0	0	0.0	**54160**	1	0.0
rc04	121	394	70	0	0	0	0.0	**59070**	1	0.0
rc05	247	972	100	0	0	0	0.0	**74070**	1	0.0
rc06	2502	12488	100	1991	8880	90	1.2	**79714**	3	5.3
rc07	2740	13156	200	2001	8674	139	1.9	**108740**	9	7.8
rc08	7527	36340	200	6840	30894	186	4.8	**112564**	11	120.5
rc09	6128	30528	200	5290	24238	168	4.0	**111005**	1	83.1
rc10	1572	6490	500	572	2078	163	0.9	**164150**	1	1.2
rc11	2858	11638	1000	1055	3676	337	3.1	**230837**	1	3.9
rt01	262	1480	10	0	0	0	0.0	**2146**	1	0.0
rt02	788	3876	50	0	0	0	0.3	**45852**	1	0.3
rt03	1725	8184	100	1430	6198	82	0.9	**7964**	1	2.6
rt04	9469	45486	100	9035	41352	94	4.2	**9693**	9	379.2
rt05	15473	77856	200	14488	68570	190	7.2	**51313**	47	2418.4

Table B.12. Detailed computational results for SPG, test-set ES10000FST.

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	Presolved t [s]	Optimum	N	t [s]
es10000fst01	27019	78814	10000	4080	13246	1621	39.0	**716174280**	1	52.2

Table B.13. Detailed computational results for SPG, test-set geo-original.

	Original			Presolved						
Instance	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]	Optimum	N	t [s]
G101	67966	164970	100	669	2544	66	4.7	**3492405**	1	5.2
G102	111707	321008	2052	9285	30744	1592	20.1	**15187538**	1	65.6
G103	135543	403606	3033	12804	42348	2277	26.9	**19938744**	1	110.1
G104	158212	480044	3914	16492	54266	2929	36.4	**26165528**	1	199.3
G105	79244	202378	550	3103	10712	409	8.7	**12507877**	1	25.3
G106	204621	636272	5556	1379	4618	233	70.1	**44547208**	1	486.0
G107	85568	228226	938	1177	3824	247	7.5	**7325530**	1	9.8
G201	44624	112410	190	775	2588	123	3.4	**3484028**	1	3.8
G202	62174	175124	1015	1773	5766	357	7.5	**6849423**	1	9.4
G203	88728	267250	2041	1817	5962	323	19.9	**13155210**	1	42.6
G204	50002	130406	386	905	2916	181	4.9	**5313548**	1	5.2
G205	120866	374624	3224	3873	12768	639	31.6	**24819583**	1	171.7
G206	60446	165880	803	254	834	56	7.8	**9175622**	1	10.6
G207	42481	105104	97	0	0	0	1.8	**2265834**	1	1.8
G301	80736	197500	191	1313	4932	152	6.7	**4797441**	1	8.1
G302	117756	330306	1879	354	1112	91	13.9	**13300990**	1	22.8
G303	147718	428352	2992	10545	33992	1853	35.2	**27941456**	1	68.7
G304	86413	217744	419	77	242	24	6.7	**6721180**	1	6.7
G305	172687	511650	3902	2540	8404	421	42.7	**40632152**	1	128.2
G306	196404	600072	4937	2329	7616	425	49.8	**33949874**	1	335.4
G307	235686	732186	6313	3647	12044	610	79.1	**51219090**	1	529.6
G308	78834	191464	88	1120	4464	72	6.5	**4699474**	3	10.9
G309	97928	257264	902	576	1872	127	12.1	**11256303**	1	14.6

Table B.14. Detailed computational results for SPG, test-set geo-advanced.

	Original			Presolved						
Instance	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]	Optimum	N	t [s]
G101a	10734	32690	96	286	1050	44	4.3	**3492405**	1	4.5
G102a	27896	87850	2003	8975	29614	1543	22.2	**15187538**	1	64.7
G103a	36270	114740	2930	5448	17926	970	26.9	**19938744**	1	98.8
G104a	44251	140058	3776	16048	52674	2830	37.9	**26165528**	1	208.2
G105a	14586	44900	525	3029	10410	402	7.6	**12507877**	1	21.8
G106a	62618	200134	5373	1380	4618	233	63.0	**44547208**	1	473.8
G107a	15536	47716	893	1181	3850	247	6.3	**7325530**	1	8.8
G201a	8286	25234	188	772	2580	124	3.3	**3484028**	1	3.6
G202a	14028	43220	985	1771	5752	360	6.7	**6849423**	1	8.7
G203a	25651	81220	1999	1803	5910	320	18.2	**13155210**	1	39.7
G204a	9939	30498	376	868	2806	176	2.8	**5313548**	1	3.2
G205a	37398	118646	3146	3815	12590	624	28.4	**24819583**	1	156.6
G206a	13688	42394	789	294	974	60	6.9	**9175622**	1	9.7
G207a	7565	23042	98	2	2	1	1.7	**2265834**	1	1.7
G301a	13291	40522	181	952	3484	125	6.3	**4797441**	1	6.9
G302a	24951	77294	1797	403	1274	101	12.6	**13300990**	1	21.6
G303a	37085	115422	2915	1308	4250	231	29.7	**27941456**	1	73.7
G304a	15213	46658	403	117	380	30	5.7	**6721180**	1	5.8
G305a	47016	147722	3809	14024	45076	2425	47.2	**40632152**	1	130.4
G306a	55423	175558	4766	2329	7614	425	50.9	**33949874**	1	374.3
G307a	71184	227232	6107	3648	12048	610	69.1	**51219090**	1	554.0
G308a	13298	40702	86	702	2740	67	6.4	**4699474**	1	7.3
G309a	18704	57702	868	2259	7478	402	11.3	**11256303**	1	13.2

Table B.15. Detailed computational results for SPG, test-set vienna-i-simple.

	Original			Presolved						
Instance	$\|V\|$	$\|A\|$	$\|T\|$	$\|V\|$	$\|A\|$	$\|T\|$	t [s]	Optimum	N	t [s]
I001	30190	95496	1184	211	646	75	2.1	**253921201**	1	2.2
I002	49920	155742	1665	642	1940	186	6.4	**399809303**	1	7.2
I003	44482	146838	3222	443	1342	126	9.6	**788774494**	1	13.2
I004	5556	17104	570	0	0	0	0.1	**279512692**	1	0.1

cont. next page

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	t [s]	Optimum	N	t [s]
I005	10284	31960	1017	0	0	0	0.2	**390876350**	1	0.2
I006	31754	105750	2202	544	1640	175	7.2	**504526035**	1	10.8
I007	15122	48742	737	136	400	42	0.9	**177909660**	1	1.0
I008	15714	51134	871	136	404	46	1.9	**201788202**	1	2.0
I009	33188	104014	1262	297	902	100	2.5	**275558727**	1	2.6
I010	29905	94914	943	151	450	58	1.4	**207889674**	1	1.4
I011	25195	82596	1428	734	2258	207	2.5	**317589880**	1	3.3
I012	12355	39924	503	32	98	16	0.3	**118893243**	1	0.3
I013	18242	57952	891	66	188	32	1.3	**193190339**	1	1.4
I014	12715	41264	475	10	26	5	0.3	**105173465**	1	0.3
I015	48833	159974	2493	424	1330	123	7.4	**592240832**	1	8.6
I016	72038	230110	4391	742	2290	207	17.5	**1110914620**	1	19.4
I017	15095	48182	478	76	234	21	0.5	**109739695**	1	0.5
I018	31121	102226	1898	982	2914	274	4.7	**463887832**	1	6.5
I019	25946	83290	866	320	970	90	1.8	**217647693**	1	1.9
I020	21808	69842	594	98	308	35	1.0	**146515460**	1	1.0
I021	16013	50538	392	17	46	7	0.5	**106470644**	1	0.5
I022	16224	51382	437	54	156	19	0.7	**106799980**	1	0.7
I023	22805	70614	582	92	294	31	0.7	**131044872**	1	0.7
I024	68464	217464	3001	275	848	79	10.9	**758483415**	1	12.3
I025	23412	75904	945	474	1488	153	3.6	**232790758**	1	3.7
I026	47429	158614	3334	1420	4372	409	11.5	**928032223**	1	13.2
I027	85085	277776	3954	1166	3564	291	16.9	**976812226**	1	18.2
I028	72701	230860	1790	176	546	59	16.6	**384053191**	1	16.6
I029	69988	223608	2162	349	1100	93	13.0	**492193565**	1	13.2
I030	33188	107360	1263	148	450	39	3.4	**321646787**	1	3.4
I031	54351	176422	2182	155	482	42	5.5	**578284709**	1	5.5
I032	56023	182798	3017	800	2404	244	6.6	**773096651**	1	7.7
I033	18555	59460	636	59	174	25	1.6	**134461857**	1	1.6
I034	22311	71032	735	64	186	21	1.9	**165115148**	1	1.9
I035	30585	100908	1704	129	386	49	3.7	**414440370**	1	4.2
I036	37208	120712	1411	125	402	36	6.5	**375260864**	1	7.0
I037	13694	44252	427	13	36	7	1.2	**105720727**	1	1.2
I038	18747	61278	967	679	2106	169	1.9	**255767543**	1	2.7
I039	8755	28898	347	88	258	38	0.7	**85566290**	1	0.7
I040	40389	131640	1762	398	1236	121	6.2	**431498867**	1	6.3
I041	47197	150614	1193	181	554	65	5.5	**301914840**	1	5.6
I042	51896	171100	2171	131	394	39	7.2	**532131412**	1	7.3
I043	10398	33574	367	108	328	41	0.9	**95722094**	1	1.0
I044	68905	227778	3358	352	1082	90	11.3	**804532332**	1	14.0
I045	14685	46932	421	80	234	26	0.6	**105944062**	1	0.6
I046	70843	234418	3598	172	516	50	12.8	**925470052**	1	14.4
I047	28524	92502	2354	2176	6606	622	5.8	**695163406**	1	8.4
I048	13189	42438	358	0	0	0	0.5	**91509264**	1	0.5
I049	30857	99182	990	159	468	51	2.7	**294811505**	1	2.7
I050	43073	142552	2868	3449	10540	920	11.1	**792599114**	1	20.6
I051	27028	90812	1524	137	406	42	5.0	**357230839**	1	5.8
I052	2363	7522	40	0	0	0	0.0	**13309487**	1	0.0
I053	3224	10570	126	19	52	8	0.1	**30854904**	1	0.1
I054	3803	12426	38	0	0	0	0.0	**15841596**	1	0.0
I055	13332	43160	570	112	338	46	0.8	**144164924**	1	0.8
I056	1991	6352	51	0	0	0	0.0	**14171206**	1	0.0
I057	33231	110298	1569	112	340	40	3.4	**412746415**	1	4.1
I058	23527	79256	1256	169	538	42	1.2	**305024188**	1	1.3
I059	9287	29950	363	49	134	22	0.2	**107617854**	1	0.2
I060	42008	135144	1242	160	504	54	6.0	**337290460**	1	6.0
I061	39160	127318	1458	171	532	46	7.5	**363042722**	1	8.1

cont. next page

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	Presolved t [s]	Optimum	N	t [s]
I062	66048	220982	3343	122	374	43	7.7	**792941137**	1	8.1
I063	26840	87322	1645	777	2366	214	4.0	**459801704**	1	4.8
I064	63158	214690	3458	6440	20058	1597	21.0	**863103567**	1	38.2
I065	3898	12712	144	12	36	9	0.2	**32965718**	1	0.2
I066	15038	49192	551	70	212	28	0.4	**174219813**	1	0.4
I067	20547	66460	627	403	1256	121	1.6	**175540750**	1	1.8
I068	33118	110254	1553	353	1066	100	2.8	**420730046**	1	3.0
I069	9574	32416	543	258	804	71	0.9	**135161583**	1	1.0
I070	15079	49216	550	123	364	48	1.8	**136700139**	1	1.8
I071	33203	108854	1494	233	684	70	3.1	**382539099**	1	3.2
I072	26948	88388	993	110	338	24	2.1	**289019226**	1	2.1
I073	21653	70342	1847	115	336	44	3.0	**663004987**	1	3.7
I074	13316	44066	653	17	50	9	0.8	**165573383**	1	0.8
I075	57551	190762	2973	110	336	33	8.5	**815404026**	1	9.0
I076	14023	45790	598	71	208	31	0.9	**166249692**	1	0.9
I077	20856	68474	1787	3514	10400	882	5.0	**472503150**	1	11.4
I078	13294	43896	835	86	244	37	1.2	**185525490**	1	1.2
I079	19867	62542	565	757	2598	213	2.6	**150506933**	1	3.2
I080	18695	59416	548	313	966	92	1.8	**164299652**	1	2.0
I081	25081	81478	888	53	154	27	2.5	**247527679**	1	2.6
I082	15592	49576	515	0	0	0	1.0	**147407632**	1	1.0
I083	89596	297166	4991	65	202	21	12.5	**1405593860**	1	14.1
I084	44934	147454	2319	95	318	26	5.0	**627187559**	1	7.0
I085	9113	28982	301	98	340	29	0.4	**80628079**	1	0.4

Table B.16. Detailed computational results for SPG, test-set vienna-i-advanced.

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	Presolved t [s]	Optimum	N	t [s]
I001a	14675	44110	941	212	638	72	1.9	**253921201**	1	2.0
I002a	23800	71516	1282	635	1918	186	5.0	**399809303**	1	5.9
I003a	16270	47838	2336	440	1332	125	8.0	**788774494**	1	11.3
I004a	867	2476	263	19	48	11	0.1	**279512692**	1	0.1
I005a	1677	4860	491	0	0	0	0.1	**390876350**	1	0.1
I006a	13339	39064	1842	104	316	28	6.0	**504526035**	1	9.7
I007a	6873	20598	599	128	370	42	0.8	**177909660**	1	0.9
I008a	6522	19258	708	101	296	33	1.6	**201788202**	1	1.7
I009a	14977	44870	1053	306	924	101	1.9	**275558727**	1	2.1
I010a	13041	39090	782	156	470	59	0.9	**207889674**	1	1.0
I011a	9298	27370	1202	709	2172	200	2.4	**317589880**	1	3.0
I012a	3500	10428	387	0	0	0	0.1	**118893243**	1	0.1
I013a	7147	21216	670	67	192	33	1.0	**193190339**	1	1.1
I014a	3577	10622	364	0	0	0	0.1	**105173465**	1	0.1
I015a	20573	61082	2119	407	1270	120	6.3	**592240832**	1	7.6
I016a	27214	79648	3434	507	1548	154	12.4	**1110914620**	1	14.9
I017a	7571	23142	386	0	0	0	0.3	**109739695**	1	0.3
I018a	12258	36028	1549	992	2942	276	3.5	**463887832**	1	5.0
I019a	11693	35248	732	278	846	79	1.4	**217647693**	1	1.5
I020a	6405	19128	508	58	180	18	0.5	**146515460**	1	0.5
I021a	5195	15722	295	102	306	27	0.2	**106470644**	1	0.2
I022a	8869	27102	356	64	188	24	0.5	**106799980**	1	0.5
I023a	13724	41726	403	222	672	64	0.5	**131044872**	1	0.5
I024a	32357	96500	2511	73	214	28	9.3	**758483415**	1	10.1
I025a	10055	29922	833	73	228	28	3.0	**232790758**	1	3.2
I026a	18155	53136	2661	1687	5180	496	9.1	**928032223**	1	10.9
I027a	40772	121110	3490	109	346	33	15.6	**976812226**	1	17.3
I028a	43690	132922	1597	255	790	85	15.4	**384053191**	1	15.5
I029a	32979	99254	1946	270	856	73	9.6	**492193565**	1	9.8
I030a	12941	38558	1093	151	460	39	2.3	**321646787**	1	2.4

cont. next page

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	Presolved t [s]	Optimum	N	t [s]
I031a	21054	62820	1832	156	484	42	3.8	**578284709**	1	3.8
I032a	21345	62706	2454	344	1058	90	5.7	**773096651**	1	6.7
I033a	8500	25400	548	252	770	76	1.1	**134461857**	1	1.2
I034a	9128	27336	606	142	412	48	1.2	**165115148**	1	1.2
I035a	13129	38840	1428	118	352	47	2.9	**414440370**	1	3.3
I036a	17036	50964	1258	318	984	74	5.4	**375260864**	1	6.0
I037a	5886	17738	392	60	180	21	0.8	**105720727**	1	0.8
I038a	7733	22956	798	693	2152	180	1.4	**255767543**	1	1.9
I039a	3719	11066	306	34	104	10	0.4	**85566290**	1	0.4
I040a	18837	56312	1501	165	512	49	5.9	**431498867**	1	6.0
I041a	22466	67736	1014	92	272	36	3.0	**301914840**	1	3.0
I042a	23925	71612	1923	116	346	34	5.7	**532131412**	1	5.9
I043a	4511	13480	335	99	288	35	0.8	**95722094**	1	0.8
I044a	31500	93514	2954	1327	4108	296	9.6	**804532332**	1	12.2
I045a	6775	20454	378	83	244	26	0.4	**105944062**	1	0.4
I046a	32376	96108	3154	163	482	50	9.9	**925470052**	1	11.5
I047a	10622	30880	1791	1365	4126	392	8.2	**695163406**	1	9.1
I048a	4920	14712	320	0	0	0	0.3	**91509264**	1	0.3
I049a	15045	45426	821	157	460	51	2.3	**294811505**	1	2.4
I050a	17787	52352	2232	3357	10250	902	10.0	**792599114**	1	18.3
I051a	12130	35784	1337	146	440	43	4.2	**357230839**	1	5.1
I052a	160	474	23	0	0	0	0.0	**13309487**	1	0.0
I053a	693	2046	102	26	72	13	0.0	**30854904**	1	0.0
I054a	540	1634	25	0	0	0	0.0	**15841596**	1	0.0
I055a	4701	13958	483	100	284	45	0.6	**144164924**	1	0.6
I056a	290	878	34	0	0	0	0.0	**14171206**	1	0.0
I057a	13078	38736	1346	178	546	64	3.0	**412746415**	1	3.6
I058a	7877	23314	997	156	494	39	0.9	**305024188**	1	1.0
I059a	2800	8314	286	31	86	11	0.1	**107617854**	1	0.1
I060a	18991	57072	1158	191	582	70	4.8	**337290460**	1	4.8
I061a	20958	62930	1337	153	464	49	6.1	**363042722**	1	6.6
I062a	23714	70610	2812	94	280	30	6.7	**792941137**	1	7.0
I063a	9600	28084	1291	950	2898	255	3.4	**459801704**	1	4.1
I064a	31712	93422	3182	6460	20152	1609	19.5	**863103567**	1	37.1
I065a	1185	3512	119	62	194	26	0.2	**32965718**	1	0.2
I066a	4551	13642	417	59	182	24	0.3	**174219813**	1	0.3
I067a	10318	31176	579	407	1272	123	1.4	**175540750**	1	1.6
I068a	12191	36046	1302	321	976	91	2.0	**420730046**	1	2.3
I069a	3508	10312	452	269	844	73	0.7	**135161583**	1	0.9
I070a	6739	20128	511	147	438	52	1.4	**136700139**	1	1.5
I071a	12772	37772	1281	117	362	36	2.3	**382539099**	1	2.5
I072a	11628	34822	851	92	268	38	1.1	**289019226**	1	1.1
I073a	7510	21746	1337	1069	3244	324	2.8	**663004987**	1	3.3
I074a	4441	13124	548	37	110	13	0.3	**165573383**	1	0.3
I075a	23195	68724	2498	102	300	33	6.7	**815404026**	1	7.1
I076a	4909	14536	498	20	54	11	0.6	**166249692**	1	0.6
I077a	9153	26726	1490	3509	10388	880	4.4	**472503150**	1	11.8
I078a	5864	17324	692	168	486	58	1.1	**185525490**	1	1.1
I079a	7933	23614	497	732	2516	205	2.1	**150506933**	1	2.6
I080a	7589	22512	499	307	950	92	1.1	**164299652**	1	1.2
I081a	10747	32058	751	85	246	45	2.0	**247527679**	1	2.0
I082a	5850	17386	435	29	82	14	0.7	**147407632**	1	0.7
I083a	34221	100602	4138	326	1010	86	9.4	**1405593860**	1	10.8
I084a	17050	50402	1918	1265	3922	306	4.3	**627187559**	1	6.2
I085a	2780	8246	243	0	0	0	0.2	**80628079**	1	0.2

B.2 Maximum-weight connected subgraph problem

The time limit for the following instances is two hours.

Table B.17. Detailed computational results for MWCSP, test-set MWCS-ACTMOD.

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	Presolved t [s]	Optimum	N	t [s]
drosophila001	5298	187214	72	3	6	2	0.2	**24.3855064**	1	0.2
drosophila005	5421	187952	195	0	0	0	0.3	**178.663952**	1	0.3
drosophila0075	5477	188288	251	3	6	2	0.2	**260.523557**	1	0.2
HCMV	3919	58916	56	3	6	2	0.1	**7.55431486**	1	0.1
lymphoma	2102	15914	68	3	6	2	0.0	**70.1663087**	1	0.0
metabol_expr_mice_1	3674	9590	151	9	26	4	0.0	**544.94837**	1	0.0
metabol_expr_mice_2	3600	9174	86	3	6	2	0.0	**241.077524**	1	0.0
metabol_expr_mice_3	2968	7354	115	6	16	3	0.0	**508.260877**	1	0.0

Table B.18. Detailed computational results for MWCSP, test-set MWCS-HANDB.

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	Presolved t [s]	Dual	Primal	Gap %	N	t [s]
handbd01	171596	687872	1796	33	108	12	2.5	**728.963591**			1	2.6
handbd02	176996	720272	7196	1123	2624	436	8.9	**296.496486**			1	9.1
handbd03	171946	689972	2146	3	6	2	0.6	**135.070605**			1	0.6
handbd04	175099	708890	5299	1353	3290	462	6.9	**1813.95916**			1	7.0
handbd05	172125	691046	2325	3	6	2	0.6	**105.474688**			1	0.6
handbd06	176275	715946	6475	1715	4672	485	6.5	**1528.76544**			1	7.2
handbd07	172641	694142	2841	3	6	2	0.7	**77.861959**			1	0.7
handbd08	176911	719762	7111	919	2350	305	5.9	**1368.16677**			1	6.1
handbd09	172409	692750	2609	3	6	2	0.7	**62.71716**			1	0.7
handbd10	177713	724574	7913	225	754	74	3.7	**1137.42973**			1	3.8
handbd11	172111	690962	2311	3	6	2	0.6	**46.772533**			1	0.6
handbd12	178656	730232	8856	3	6	2	3.1	**321.204744**			1	3.1
handbd13	172681	694382	2881	43367	170616	792	32.9	13.1776581	13.185228	0.1	1	>7200.3
handbd14	169950	677996	150	3	6	2	0.2	**4379.10424**			1	0.2
handbi01	160177	642272	1777	17	52	7	0.7	**1358.56338**			1	0.7
handbi02	165361	673376	6961	1055	2462	417	4.8	**531.810883**			1	5.0
handbi03	160336	643226	1936	3	6	2	0.6	**243.134201**			1	0.6
handbi04	163630	662990	5230	8267	28384	718	19.9	**3202.18574**			1	23.8
handbi05	160691	645356	2291	3	6	2	0.6	**184.467331**			1	0.6
handbi06	164158	666158	5758	3190	9982	493	8.1	**2921.54472**			1	9.0
handbi07	160657	645152	2257	3	6	2	0.7	**150.974258**			1	0.7
handbi08	165259	672764	6859	595	1480	211	4.4	**2270.28462**			1	4.6
handbi09	160674	645254	2274	5	12	3	0.7	**107.768806**			1	0.7
handbi10	166033	677408	7633	72	236	25	2.6	**1874.29296**			1	2.6
handbi11	160843	646268	2443	3	6	2	0.6	**68.944709**			1	0.6
handbi12	166538	680438	8138	3	6	2	1.3	**138.257023**			1	1.3
handbi13	161089	647744	2689	98606	391402	1554	120.2	4.022194	4.250363	5.7	1	>7202.0
handbi14	166371	679436	7971	3	6	2	0.7	**7881.76874**			1	0.7

Table B.19. Detailed computational results for MWCSP, test-set MWCS-HANDS.

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	Presolved t [s]	Optimum	N	t [s]
handsi01	40033	160000	433	3	6	2	0.1	**295.453616**	1	0.1
handsi02	41304	167626	1704	321	1066	108	0.7	**125.429411**	1	0.7
handsi03	40220	161122	620	15	46	6	0.1	**56.149422**	1	0.1
handsi04	41030	165982	1430	474	1052	204	0.8	**722.508197**	1	0.8
handsi05	40188	160930	588	3	6	2	0.1	**35.043506**	1	0.1
handsi06	41513	168880	1913	182	604	61	0.5	**452.953621**	1	0.5

cont. next page

Instance	Original $\|V\|$	$\|A\|$	$\|T\|$	Presolved $\|V\|$	$\|A\|$	$\|T\|$	t [s]	Optimum	N	t [s]
handsi07	40203	161020	603	3	6	2	0.1	**18.410135**	1	0.1
handsi08	41597	169384	1997	6	16	3	0.2	**229.52993**	1	0.2
handsi09	40213	161080	613	3	6	2	0.3	**5.962166**	1	0.3
handsi10	40966	165598	1366	1011	2642	428	1.1	**1803.69751**	1	1.6
handsd01	43024	172088	524	3	6	2	0.1	**171.636766**	1	0.1
handsd02	44084	178448	1584	691	1882	174	0.8	**159.751395**	1	1.0
handsd03	43213	173222	713	3	6	2	0.1	**31.306275**	1	0.1
handsd04	43842	176996	1342	507	1686	170	0.7	**491.733164**	1	0.8
handsd05	43205	173174	705	3	6	2	0.1	**21.937611**	1	0.1
handsd06	44477	180806	1977	326	1088	103	0.6	**279.90313**	1	0.7
handsd07	43176	173000	676	3	6	2	0.1	**11.80412**	1	0.1
handsd08	44624	181688	2124	30	96	11	0.3	**143.237729**	1	0.3
handsd09	43183	173042	683	3	6	2	0.1	**3.818683**	1	0.1
handsd10	42806	170780	306	3	6	2	0.1	**1034.76736**	1	0.1

Table B.20. Detailed computational results for MWCSP, test-set MWCS-JMPALMK.

Instance	Original $\|V\|$	$\|A\|$	$\|T\|$	Presolved $\|V\|$	$\|A\|$	$\|T\|$	t [s]	Optimum	N	t [s]
MWCS-I-D-n-10-0-a-0-6-d-0-25-e-0-25	1193	11024	193	3	6	2	0.0	**931.538552**	1	0.0
MWCS-I-D-n-10-0-a-0-6-d-0-25-e-0-5	1388	12194	388	3	6	2	0.0	**1872.2754**	1	0.0
MWCS-I-D-n-10-0-a-0-6-d-0-25-e-0-75	1564	13250	564	3	6	2	0.0	**2789.57911**	1	0.0
MWCS-I-D-n-10-0-a-0-6-d-0-5-e-0-25	1114	10550	114	3	6	2	0.0	**522.525615**	1	0.0
MWCS-I-D-n-10-0-a-0-6-d-0-5-e-0-5	1250	11366	250	3	6	2	0.0	**1197.85102**	1	0.0
MWCS-I-D-n-10-0-a-0-6-d-0-5-e-0-75	1374	12110	374	3	6	2	0.0	**1762.70747**	1	0.0
MWCS-I-D-n-10-0-a-0-6-d-0-75-e-0-25	1062	10238	62	3	6	2	0.0	**332.791924**	1	0.0
MWCS-I-D-n-10-0-a-0-6-d-0-75-e-0-5	1141	10712	141	3	6	2	0.0	**754.300601**	1	0.0
MWCS-I-D-n-10-0-a-0-6-d-0-75-e-0-75	1196	11042	196	3	6	2	0.0	**998.215414**	1	0.0
MWCS-I-D-n-10-0-a-1-d-0-25-e-0-25	1193	27710	193	3	6	2	0.0	**939.39337**	1	0.0
MWCS-I-D-n-10-0-a-1-d-0-25-e-0-5	1388	28880	388	3	6	2	0.0	**1883.21361**	1	0.0
MWCS-I-D-n-10-0-a-1-d-0-25-e-0-75	1564	29936	564	3	6	2	0.0	**2789.57911**	1	0.0
MWCS-I-D-n-10-0-a-1-d-0-5-e-0-25	1114	27236	114	3	6	2	0.0	**533.4294**	1	0.0
MWCS-I-D-n-10-0-a-1-d-0-5-e-0-5	1250	28052	250	3	6	2	0.0	**1205.42131**	1	0.0
MWCS-I-D-n-10-0-a-1-d-0-5-e-0-75	1374	28796	374	3	6	2	0.0	**1770.27776**	1	0.0
MWCS-I-D-n-10-0-a-1-d-0-75-e-0-25	1062	26924	62	3	6	2	0.0	**336.829944**	1	0.0
MWCS-I-D-n-10-0-a-1-d-0-75-e-0-5	1141	27398	141	3	6	2	0.0	**760.284581**	1	0.0
MWCS-I-D-n-10-0-a-1-d-0-75-e-0-75	1196	27728	196	3	6	2	0.0	**1004.19939**	1	0.0
MWCS-I-D-n-150–a-0-6-d-0-25-e-0-25	1785	17028	285	3	6	2	0.0	**1333.47643**	1	0.0
MWCS-I-D-n-150–a-0-6-d-0-25-e-0-5	2078	18786	578	3	6	2	0.0	**2799.67722**	1	0.0
MWCS-I-D-n-150–a-0-6-d-0-25-e-0-75	2353	20436	853	3	6	2	0.0	**4230.25112**	1	0.0
MWCS-I-D-n-150–a-0-6-d-0-5-e-0-25	1680	16398	180	3	6	2	0.0	**847.452011**	1	0.0
MWCS-I-D-n-150–a-0-6-d-0-5-e-0-5	1881	17604	381	3	6	2	0.0	**1858.0926**	1	0.0
MWCS-I-D-n-150–a-0-6-d-0-5-e-0-75	2060	18678	560	3	6	2	0.0	**2697.45876**	1	0.0
MWCS-I-D-n-150–a-0-6-d-0-75-e-0-25	1594	15882	94	3	6	2	0.0	**502.17599**	1	0.0
MWCS-I-D-n-150–a-0-6-d-0-75-e-0-5	1705	16548	205	3	6	2	0.0	**1089.77117**	1	0.0
MWCS-I-D-n-150–a-0-6-d-0-75-e-0-75	1779	16992	279	3	6	2	0.0	**1423.61063**	1	0.0
MWCS-I-D-n-150–a-1-d-0-25-e-0-25	1785	42758	285	3	6	2	0.0	**1377.0144**	1	0.0
MWCS-I-D-n-150–a-1-d-0-25-e-0-5	2078	44516	578	3	6	2	0.0	**2820.05174**	1	0.0
MWCS-I-D-n-150–a-1-d-0-25-e-0-75	2353	46166	853	3	6	2	0.0	**4230.25112**	1	0.0
MWCS-I-D-n-150–a-1-d-0-5-e-0-25	1680	42128	180	3	6	2	0.0	**860.618961**	1	0.0
MWCS-I-D-n-150–a-1-d-0-5-e-0-5	1881	43334	381	3	6	2	0.0	**1865.66289**	1	0.0
MWCS-I-D-n-150–a-1-d-0-5-e-0-75	2060	44408	560	3	6	2	0.0	**2707.70001**	1	0.0
MWCS-I-D-n-150–a-1-d-0-75-e-0-25	1594	41612	94	3	6	2	0.0	**502.17599**	1	0.0
MWCS-I-D-n-150–a-1-d-0-75-e-0-5	1705	42278	205	3	6	2	0.0	**1089.77117**	1	0.0
MWCS-I-D-n-150–a-1-d-0-75-e-0-75	1779	42722	279	3	6	2	0.0	**1423.61063**	1	0.0
MWCS-I-D-n-50–a-0-62-d-0-25-e-0-25	590	5728	90	3	6	2	0.0	**460.577357**	1	0.0
MWCS-I-D-n-50–a-0-62-d-0-25-e-0-5	696	6364	196	3	6	2	0.0	**992.967111**	1	0.0
MWCS-I-D-n-50–a-0-62-d-0-25-e-0-75	788	6916	288	3	6	2	0.0	**1447.54452**	1	0.0

cont. next page

Instance	Original $\vert V\vert$	$\vert A\vert$	$\vert T\vert$	Presolved $\vert V\vert$	$\vert A\vert$	$\vert T\vert$	t [s]	Optimum	N	t [s]
MWCS-I-D-n-50–a-0-62-d-0-5-e-0-25	556	5524	56	3	6	2	0.0	**280.832378**	1	0.0
MWCS-I-D-n-50–a-0-62-d-0-5-e-0-5	629	5962	129	3	6	2	0.0	**655.623217**	1	0.0
MWCS-I-D-n-50–a-0-62-d-0-5-e-0-75	696	6364	196	3	6	2	0.0	**965.554694**	1	0.0
MWCS-I-D-n-50–a-0-62-d-0-75-e-0-25	531	5374	31	3	6	2	0.0	**171.628785**	1	0.0
MWCS-I-D-n-50–a-0-62-d-0-75-e-0-5	566	5584	66	3	6	2	0.0	**362.188212**	1	0.0
MWCS-I-D-n-50–a-0-62-d-0-75-e-0-75	593	5746	93	3	6	2	0.0	**490.623986**	1	0.0
MWCS-I-D-n-50–a-1-d-0-25-e-0-25	590	13572	90	3	6	2	0.0	**471.393285**	1	0.0
MWCS-I-D-n-50–a-1-d-0-25-e-0-5	696	14208	196	3	6	2	0.0	**995.313181**	1	0.0
MWCS-I-D-n-50–a-1-d-0-25-e-0-75	788	14760	288	3	6	2	0.0	**1447.54452**	1	0.0
MWCS-I-D-n-50–a-1-d-0-5-e-0-25	556	13368	56	3	6	2	0.0	**286.920868**	1	0.0
MWCS-I-D-n-50–a-1-d-0-5-e-0-5	629	13806	129	3	6	2	0.0	**661.711707**	1	0.0
MWCS-I-D-n-50–a-1-d-0-5-e-0-75	696	14208	196	3	6	2	0.0	**965.554694**	1	0.0
MWCS-I-D-n-50–a-1-d-0-75-e-0-25	531	13218	31	3	6	2	0.0	**171.628785**	1	0.0
MWCS-I-D-n-50–a-1-d-0-75-e-0-5	566	13428	66	3	6	2	0.0	**362.188212**	1	0.0
MWCS-I-D-n-50–a-1-d-0-75-e-0-75	593	13590	93	3	6	2	0.0	**490.623986**	1	0.0
MWCS-I-D-n-750-a-0-647-d-0-25-e-0-25	891	9278	141	3	6	2	0.0	**702.644057**	1	0.0
MWCS-I-D-n-750-a-0-647-d-0-25-e-0-5	1041	10178	291	3	6	2	0.0	**1419.77986**	1	0.0
MWCS-I-D-n-750-a-0-647-d-0-25-e-0-75	1176	10988	426	3	6	2	0.0	**2116.58233**	1	0.0
MWCS-I-D-n-750-a-0-647-d-0-5-e-0-25	830	8912	80	3	6	2	0.0	**403.177763**	1	0.0
MWCS-I-D-n-750-a-0-647-d-0-5-e-0-5	939	9566	189	3	6	2	0.0	**946.129495**	1	0.0
MWCS-I-D-n-750-a-0-647-d-0-5-e-0-75	1036	10148	286	3	6	2	0.0	**1382.77203**	1	0.0
MWCS-I-D-n-750-a-0-647-d-0-75-e-0-25	799	8726	49	3	6	2	0.0	**266.983922**	1	0.0
MWCS-I-D-n-750-a-0-647-d-0-75-e-0-5	856	9068	106	3	6	2	0.0	**580.407832**	1	0.0
MWCS-I-D-n-750-a-0-647-d-0-75-e-0-75	895	9302	145	3	6	2	0.0	**764.156726**	1	0.0
MWCS-I-D-n-750-a-1-d-0-25-e-0-25	891	20484	141	3	6	2	0.0	**708.143835**	1	0.0
MWCS-I-D-n-750-a-1-d-0-25-e-0-5	1041	21384	291	3	6	2	0.0	**1426.44904**	1	0.0
MWCS-I-D-n-750-a-1-d-0-25-e-0-75	1176	22194	426	3	6	2	0.0	**2116.58233**	1	0.0
MWCS-I-D-n-750-a-1-d-0-5-e-0-25	830	20118	80	3	6	2	0.0	**403.177763**	1	0.0
MWCS-I-D-n-750-a-1-d-0-5-e-0-5	939	20772	189	3	6	2	0.0	**946.129495**	1	0.0
MWCS-I-D-n-750-a-1-d-0-5-e-0-75	1036	21354	286	3	6	2	0.0	**1382.77203**	1	0.0
MWCS-I-D-n-750-a-1-d-0-75-e-0-25	799	19932	49	3	6	2	0.0	**266.983922**	1	0.0
MWCS-I-D-n-750-a-1-d-0-75-e-0-5	856	20274	106	3	6	2	0.0	**580.407832**	1	0.0
MWCS-I-D-n-750-a-1-d-0-75-e-0-75	895	20508	145	3	6	2	0.0	**764.156726**	1	0.0

Table B.21. Detailed computational results for MWCSP, test-set MWCS-PUCNU.

Instance	Original $\vert V\vert$	$\vert A\vert$	$\vert T\vert$	Presolved $\vert V\vert$	$\vert A\vert$	$\vert T\vert$	t [s]	Dual	Primal	Gap %	N	t [s]
transformed_bip42nu	1307	8600	107	991	7224	106	0.6	**226**			168	58.0
transformed_bip52nu	2303	16606	103	1819	14652	102	0.8	**222**			555	387.8
transformed_bip62nu	1303	20616	103	1199	20000	102	0.5	211.022467	214	1.4	4003	>7200.0
transformed_bipa2nu	3439	36974	139	3281	36426	139	1.8	320.430498	327	2.1	1	>7200.0
transformed_bipe2nu	576	10176	26	550	10026	25	0.1	**53**			7	35.8
transformed_cc10-2nu	1090	10630	66	981	9360	51	0.3	**167**			23	578.8
transformed_cc11-2nu	2174	23276	126	1970	20752	101	0.9	300.580262	304	1.1	94	>7200.1
transformed_cc12-2nu	4323	50504	227	3923	45066	165	1.5	559.295815	564	0.8	19	>7200.1
transformed_cc3-10nu	1019	27108	19	1007	16228	18	0.3	**61**			1	84.0
transformed_cc3-11nu	1366	40134	35	1347	23524	34	0.5	**79**			1	2.8
transformed_cc3-12nu	1769	57264	41	1739	32648	40	0.7	**95**			1	5.6
transformed_cc3-4nu	70	606	6	3	6	2	0.0	**10**			1	0.0
transformed_cc3-5nu	134	1548	9	3	6	2	0.0	**17**			1	0.0
transformed_cc5-3nu	257	2508	14	0	0	0	0.0	**36**			1	0.0
transformed_cc6-2nu	70	414	6	3	6	2	0.0	**15**			1	0.0
transformed_cc6-3nu	768	8964	39	701	7050	29	0.1	**95**			1	0.6
transformed_cc7-3nu	2303	31306	116	2108	25686	91	0.4	268.268191	270	0.6	58	>7201.9
transformed_cc9-2nu	542	4782	30	0	0	0	0.1	**83**			1	0.1

Table B.22. Detailed computational results for MWCSP, test-set MWCS-SHINY.

Instance	Original			Presolved						
	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Optimum	N	t [s]
25e814a792c4	4186	13210	872	3	6	2	0.1	**1083.30811**	1	0.1
25e81700dead	4186	13210	872	3	6	2	0.0	**2527.27139**	1	0.0
25e83661bc4	3350	8194	36	3	6	2	0.0	**65.5501822**	1	0.0
25e83d7dbeea	3319	8008	5	3	6	2	0.0	**11.0598178**	1	0.0
25e857e14393	4186	13210	872	3	6	2	0.0	**1660.33065**	1	0.0
3a0d1335fe78	3555	7862	168	29	94	10	0.0	**29.0905466**	1	0.0
3a0d151a8ee0	3686	8314	230	9	26	4	0.0	**9.51295927**	1	0.0
3a0d17a83362	3357	7786	276	33	104	11	0.0	**111.428553**	1	0.0
3a0d1a1e31cf	3302	8924	281	37	124	12	0.0	**141.063702**	1	0.0
3a0d2255a681	3754	8296	177	3	6	2	0.0	**5.07786653**	1	0.0
3a0d226a0a5c	3259	7390	198	21	66	8	0.0	**63.5028802**	1	0.0
3a0d25c9a738	3552	7844	165	9	26	4	0.0	**28.4732591**	1	0.0
3a0d25f9bda3	3870	11056	809	12	36	5	0.0	**119.083963**	1	0.0
3a0d2875c8cf	3517	7634	130	21	64	7	0.0	**22.8385099**	1	0.0
3a0d325af5cc	3347	7402	157	27	86	9	0.0	**37.0853611**	1	0.0
3a0d32b18854	3214	7120	153	6	16	3	0.0	**40.960459**	1	0.0
3a0d33d2aa32	3905	9202	328	3	6	2	0.0	**16.6661429**	1	0.0
3a0d390c537e	286	722	54	3	6	2	0.0	**38.6838961**	1	0.0
3a0d435ee480	3657	8140	201	9	26	4	0.0	**12.0074874**	1	0.0
3a0d4427fe32	1955	4308	71	3	6	2	0.0	**38.0000646**	1	0.0
3a0d4ccc9b37	2754	6616	76	3	6	2	0.0	**1162.91998**	1	0.0
3a0d4dac5319	1955	4308	71	3	6	2	0.0	**15.29986**	1	0.0
3a0d52ee8185	3617	7900	161	3	6	2	0.0	**5.02525142**	1	0.0
3a0d55ddd0a5	3649	8092	193	3	6	2	0.0	**5.32580928**	1	0.0
3a0d568fbd87	3519	7646	132	21	64	7	0.0	**24.8508763**	1	0.0
3a0d5dc4a759	3615	7888	159	3	6	2	0.0	**4.93513445**	1	0.0
3a0d5e4aac27	3843	9256	387	17	52	7	0.0	**18.2490709**	1	0.0
3a0d5e4aac27x	3843	9256	387	17	52	7	0.0	**18.2490709**	1	0.0
3a0d610beb4c	3208	6812	107	15	46	6	0.0	**66.8976568**	1	0.0
3a0d6505353b	3345	7906	284	3	6	2	0.0	**79.3302586**	1	0.0
3a0d6a21bbd5	3237	7258	176	29	94	10	0.0	**53.9189217**	1	0.0
3a0d6e97602a	1955	4308	71	3	6	2	0.0	**13.29986**	1	0.0
3a0d724ffec9	1955	4308	71	3	6	2	0.0	**6.32757816**	1	0.0
3a0d73143aeb	3704	8160	168	9	26	4	0.0	**10.8861513**	1	0.0
3a0dff0eb70	1955	4308	71	3	6	2	0.0	**19.5375417**	1	0.0
48e7452da6ba	3905	8232	77	3	6	2	0.0	**406.044087**	1	0.0
48e7526364af	3091	7658	70	3	6	2	0.0	**346.728578**	1	0.0
48e76a6886bc	3691	9282	50	3	6	2	0.0	**37.561934**	1	0.0
795313fd138b	3259	8666	238	3	6	2	0.0	**119.552962**	1	0.0

B.3 Prize-collecting Steiner tree problem

The time limit for the following instances is two hours.

Table B.23. Detailed computational results for PCSTP, test-set PCSPG-ACTMOD.

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	Presolved t [s]	Optimum	N	t [s]
drosophila001	5298	187214	72	3	6	2	0.6	**8273.98263**	1	0.6
drosophila005	5421	187952	195	199	934	65	0.7	**8121.31358**	1	0.8
drosophila0075	5477	188288	251	0	0	0	0.5	**8039.85946**	1	0.5
HCMV	3919	58916	56	3	6	2	0.2	**7371.53637**	1	0.2
lymphoma	2102	15914	68	3	6	2	0.0	**3341.89024**	1	0.0
metabol_expr_mice_1	3674	9590	151	3	6	2	0.0	**11346.9272**	1	0.0
metabol_expr_mice_2	3600	9174	86	3	6	2	0.0	**16250.2352**	1	0.0
metabol_expr_mice_3	2968	7354	115	3	6	2	0.0	**16919.6204**	1	0.0

Table B.24. Detailed computational results for PCSTP, test-set PCSPG-CRR.

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	Presolved t [s]	Optimum	N	t [s]
C01-A	505	1274	5	3	6	2	0.0	**18**	1	0.0
C01-B	506	1280	6	3	6	2	0.0	**85**	1	0.0
C02-A	509	1298	9	3	6	2	0.0	**50**	1	0.0
C02-B	511	1310	11	3	6	2	0.0	**141**	1	0.0
C03-A	552	1556	52	3	6	2	0.0	**414**	1	0.0
C03-B	584	1748	84	3	6	2	0.0	**737**	1	0.0
C04-A	570	1664	70	3	6	2	0.0	**618**	1	0.0
C04-B	621	1970	121	3	6	2	0.0	**1063**	1	0.0
C05-A	644	2108	144	3	6	2	0.0	**1080**	1	0.0
C05-B	740	2684	240	3	6	2	0.0	**1528**	1	0.0
C06-A	504	2018	4	3	6	2	0.0	**18**	1	0.0
C06-B	506	2030	6	3	6	2	0.0	**55**	1	0.0
C07-A	510	2054	10	3	6	2	0.0	**50**	1	0.0
C07-B	511	2060	11	3	6	2	0.0	**102**	1	0.0
C08-A	561	2360	61	3	6	2	0.0	**361**	1	0.0
C08-B	583	2492	83	3	6	2	0.0	**500**	1	0.0
C09-A	587	2516	87	3	6	2	0.0	**533**	1	0.0
C09-B	622	2726	122	3	6	2	0.0	**694**	1	0.0
C10-A	664	2978	164	3	6	2	0.0	**859**	1	0.0
C10-B	742	3446	242	3	6	2	0.0	**1069**	1	0.0
C11-A	505	5024	5	3	6	2	0.0	**18**	1	0.0
C11-B	506	5030	6	3	6	2	0.0	**32**	1	0.0
C12-A	510	5054	10	3	6	2	0.0	**38**	1	0.0
C12-B	511	5060	11	3	6	2	0.1	**46**	1	0.1
C13-A	572	5426	72	3	6	2	0.0	**236**	1	0.0
C13-B	584	5498	84	3	6	2	0.0	**258**	1	0.0
C14-A	603	5612	103	3	6	2	0.0	**293**	1	0.0
C14-B	623	5732	123	3	6	2	0.0	**318**	1	0.0
C15-A	705	6224	205	3	6	2	0.0	**501**	1	0.0
C15-B	748	6482	248	3	6	2	0.0	**551**	1	0.0
C16-A	506	25030	6	3	6	2	0.1	**11**	1	0.1
C16-B	506	25030	6	3	6	2	0.1	**11**	1	0.1
C17-A	511	25060	11	3	6	2	0.1	**18**	1	0.1
C17-B	511	25060	11	3	6	2	0.1	**18**	1	0.1
C18-A	577	25456	77	114	438	40	0.3	**111**	1	0.3
C18-B	584	25498	84	184	820	46	0.3	**113**	1	0.3
C19-A	611	25660	111	3	6	2	0.0	**146**	1	0.0

cont. next page

Instance	Original $\|V\|$	$\|A\|$	$\|T\|$	Presolved $\|V\|$	$\|A\|$	$\|T\|$	t [s]	Optimum	N	t [s]
C19-B	625	25744	125	3	6	2	0.0	**146**	1	0.0
C20-A	718	26302	218	3	6	2	0.0	**266**	1	0.0
C20-B	748	26482	248	3	6	2	0.0	**267**	1	0.0
D01-A	1003	2512	3	3	6	2	0.0	**18**	1	0.0
D01-B	1006	2530	6	3	6	2	0.0	**106**	1	0.0
D02-A	1009	2548	9	3	6	2	0.0	**50**	1	0.0
D02-B	1011	2560	11	3	6	2	0.0	**218**	1	0.0
D03-A	1096	3070	96	3	6	2	0.0	**807**	1	0.0
D03-B	1157	3436	157	3	6	2	0.0	**1509**	1	0.0
D04-A	1141	3340	141	3	6	2	0.0	**1203**	1	0.0
D04-B	1238	3922	238	3	6	2	0.0	**1881**	1	0.0
D05-A	1277	4156	277	3	6	2	0.0	**2157**	1	0.0
D05-B	1479	5368	479	3	6	2	0.0	**3135**	1	0.0
D06-A	1005	4024	5	3	6	2	0.0	**18**	1	0.0
D06-B	1006	4030	6	3	6	2	0.1	**67**	1	0.1
D07-A	1010	4054	10	3	6	2	0.0	**50**	1	0.0
D07-B	1011	4060	11	3	6	2	0.1	**103**	1	0.1
D08-A	1109	4648	109	3	6	2	0.0	**755**	1	0.0
D08-B	1160	4954	160	3	6	2	0.0	**1036**	1	0.0
D09-A	1165	4984	165	3	6	2	0.0	**1070**	1	0.0
D09-B	1245	5464	245	3	6	2	0.0	**1420**	1	0.0
D10-A	1345	6064	345	3	6	2	0.0	**1671**	1	0.0
D10-B	1486	6910	486	3	6	2	0.0	**2079**	1	0.0
D11-A	1006	10030	6	3	6	2	0.0	**18**	1	0.0
D11-B	1006	10030	6	3	6	2	0.0	**29**	1	0.0
D12-A	1011	10060	11	3	6	2	0.1	**42**	1	0.1
D12-B	1011	10060	11	3	6	2	0.1	**42**	1	0.1
D13-A	1137	10816	137	3	6	2	0.1	**445**	1	0.1
D13-B	1164	10978	164	3	6	2	0.0	**486**	1	0.0
D14-A	1206	11230	206	3	6	2	0.0	**602**	1	0.0
D14-B	1246	11470	246	3	6	2	0.0	**665**	1	0.0
D15-A	1404	12418	404	3	6	2	0.0	**1042**	1	0.0
D15-B	1490	12934	490	3	6	2	0.0	**1108**	1	0.0
D16-A	1006	50030	6	3	6	2	0.1	**13**	1	0.1
D16-B	1006	50030	6	3	6	2	0.1	**13**	1	0.1
D17-A	1011	50060	11	3	6	2	0.2	**23**	1	0.2
D17-B	1011	50060	11	3	6	2	0.2	**23**	1	0.2
D18-A	1145	50864	145	293	1324	79	0.6	**218**	1	0.7
D18-B	1165	50984	165	0	0	0	0.5	**223**	1	0.5
D19-A	1219	51308	219	3	6	2	0.1	**306**	1	0.1
D19-B	1248	51482	248	290	1300	83	0.3	**310**	1	0.3
D20-A	1437	52616	437	3	6	2	0.1	**536**	1	0.1
D20-B	1495	52964	495	3	6	2	0.1	**537**	1	0.1

Table B.25. Detailed computational results for PCSTP, test-set PCSPG-E.

Instance	Original $\|V\|$	$\|A\|$	$\|T\|$	Presolved $\|V\|$	$\|A\|$	$\|T\|$	t [s]	Optimum	N	t [s]
e01-A	2503	6262	3	3	6	2	0.0	**13**	1	0.0
e01-B	2506	6280	6	3	6	2	0.0	**109**	1	0.0
e02-A	2506	6280	6	3	6	2	0.0	**30**	1	0.0
e02-B	2511	6310	11	3	6	2	0.1	**170**	1	0.1
e03-A	2757	7786	257	3	6	2	0.0	**2231**	1	0.0
e03-B	2898	8632	398	3	6	2	0.0	**3806**	1	0.0
e04-A	2885	8554	385	3	6	2	0.0	**3151**	1	0.0
e04-B	3092	9796	592	3	6	2	0.0	**4888**	1	0.0
e05-A	3272	10876	772	3	6	2	0.0	**5657**	1	0.0
e05-B	3711	13510	1211	3	6	2	0.0	**7998**	1	0.0
e06-A	2505	10024	5	3	6	2	0.0	**19**	1	0.0
e06-B	2506	10030	6	3	6	2	0.0	**70**	1	0.0
e07-A	2506	10030	6	3	6	2	0.0	**40**	1	0.0

cont. next page

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Optimum	N	t [s]
e07-B	2511	10060	11	3	6	2	0.2	**136**	1	0.2
e08-A	2794	11758	294	0	0	0	0.1	**1878**	1	0.1
e08-B	2905	12424	405	3	6	2	0.1	**2555**	1	0.1
e09-A	2941	12640	441	3	6	2	0.1	**2787**	1	0.1
e09-B	3103	13612	603	3	6	2	0.1	**3541**	1	0.1
e10-A	3401	15400	901	3	6	2	0.1	**4586**	1	0.1
e10-B	3709	17248	1209	3	6	2	0.1	**5502**	1	0.1
e11-A	2505	25024	5	3	6	2	0.1	**21**	1	0.1
e11-B	2506	25030	6	3	6	2	0.1	**34**	1	0.1
e12-A	2510	25054	10	3	6	2	0.2	**49**	1	0.2
e12-B	2511	25060	11	3	6	2	0.7	**67**	1	0.7
e13-A	2857	27136	357	3	6	2	0.4	**1169**	1	0.5
e13-B	2911	27460	411	34	106	19	1.2	**1269**	1	2.3
e14-A	3016	28090	516	3	6	2	0.1	**1579**	1	0.1
e14-B	3118	28702	618	3	6	2	0.2	**1716**	1	0.2
e15-A	3553	31312	1053	3	6	2	0.1	**2610**	1	0.1
e15-B	3736	32410	1236	3	6	2	0.1	**2767**	1	0.1
e16-A	2506	125030	6	3	6	2	0.3	**15**	1	0.3
e16-B	2506	125030	6	3	6	2	0.3	**15**	1	0.3
e17-A	2511	125060	11	3	6	2	0.9	**25**	1	0.9
e17-B	2511	125060	11	3	6	2	0.9	**25**	1	1.0
e18-A	2872	127226	372	2080	11800	229	4.2	**555**	7	22.5
e18-B	2917	127496	417	2023	11196	245	2.7	**564**	8	21.7
e19-A	3058	128342	558	0	0	0	0.8	**747**	1	0.8
e19-B	3121	128720	621	669	3092	146	1.5	**758**	1	1.8
e20-A	3619	131708	1119	3	6	2	0.3	**1331**	1	0.3
e20-B	3743	132452	1243	3	6	2	0.3	**1342**	1	0.3

Table B.26. Detailed computational results for PCSTP, test-set PCSPG-H2.

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Dual	Primal	Gap %	N	t [s]
hc10p2	1531	13276	507	1452	11952	506	5.1	58914.488	59799	1.5	5781	>7200.0
hc10u2	1203	11308	179	1024	10240	178	4.0	375.958261	380	1.1	4120	>7200.0
hc11p2	3063	28612	1015	2916	26000	1014	17.0	116909.439	119051	1.8	1791	>7200.0
hc11u2	2398	24622	350	2048	22528	349	9.1	742.747344	755	1.6	1	>7200.0
hc12p2	6126	61326	2030	6126	61326	2030	37.0	232156.555	237133	2.1	282	>7200.0
hc12u2	4798	53358	702	4798	53358	702	9.7	1475.40845	1502	1.8	1	>7200.2
hc6p2	97	576	33	93	500	32	0.0	**3923**			13909	63.2
hc6u2	74	438	10	0	0	0	0.0	**20**			1	0.0
hc7p2	192	1274	64	183	1116	63	0.2	7638.82886	7711	0.9	422469	>7200.0
hc7u2	151	1028	23	99	530	22	0.2	**47**			3	0.3
hc8p2	384	2810	128	368	2496	127	0.6	15046.4656	15231	1.2	78732	>7200.0
hc8u2	297	2288	41	256	2032	40	0.5	**97**			7	18.8
hc9p2	766	6126	254	727	5468	253	1.5	29799.6858	30233	1.5	22759	>7200.0
hc9u2	595	5100	83	512	4596	82	1.6	**190**			5366	2601.4

Table B.27. Detailed computational results for PCSTP, test-set PCSPG-HANDB.

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Dual	Primal	Gap %	N	t [s]
handbi01	160177	642272	1777	3	6	2	1.4	**1358.56338**			1	1.4
handbi02	165361	673376	6961	0	0	0	5.0	**531.810883**			1	5.1
handbi03	160336	643226	1936	3	6	2	1.1	**243.134201**			1	1.1
handbi04	163630	662990	5230	63	214	30	9.8	**3202.18574**			1	9.8
handbi05	160691	645356	2291	3	6	2	1.0	**184.467331**			1	1.0
handbi06	164158	666158	5758	0	0	0	6.8	**2921.55784**			1	6.8
handbi07	160657	645152	2257	0	0	0	1.4	**150.974258**			1	1.4
handbi08	165259	672764	6859	0	0	0	4.5	**2270.28462**			1	4.5

cont. next page

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Dual	Primal	Gap %	N	t [s]
handbi09	160674	645254	2274	0	0	0	1.3	**107.768806**			1	1.3
handbi10	166033	677408	7633	3	6	2	2.6	**1874.29296**			1	2.6
handbi11	160843	646268	2443	3	6	2	1.0	**68.944709**			1	1.0
handbi12	166538	680438	8138	3	6	2	1.6	**138.257023**			1	1.6
handbi13	161089	647744	2689	105707	422142	1724	112.6	3.748109	4.249969	13.4	1	>7201.3
handbi14	166371	679436	7971	3	6	2	1.5	**7881.76874**			1	1.5
handbd01	171596	687872	1796	3	6	2	2.9	**728.963591**			1	2.9
handbd02	176996	720272	7196	3	6	2	5.6	**296.496486**			1	5.6
handbd03	171946	689972	2146	3	6	2	1.2	**135.070605**			1	1.2
handbd04	175099	708890	5299	0	0	0	5.0	**1813.95916**			1	5.1
handbd05	172125	691046	2325	3	6	2	1.1	**105.474688**			1	1.1
handbd06	176275	715946	6475	0	0	0	6.6	**1528.76544**			1	6.7
handbd07	172641	694142	2841	3	6	2	1.1	**77.861959**			1	1.1
handbd08	176911	719762	7111	0	0	0	5.9	**1368.16677**			1	5.9
handbd09	172409	692750	2609	3	6	2	1.1	**62.71716**			1	1.2
handbd10	177713	724574	7913	3	6	2	3.6	**1137.42973**			1	3.6
handbd11	172111	690962	2311	3	6	2	1.1	**46.772533**			1	1.1
handbd12	178656	730232	8856	3	6	2	3.1	**321.204744**			1	3.1
handbd13	172681	694382	2881	33638	134186	699	29.7	13.1777737	13.185068	0.1	1	>7200.3
handbd14	169950	677996	150	3	6	2	0.9	**4379.10424**			1	0.9

Table B.28. Detailed computational results for PCSTP, test-set PCSPG-HANDS.

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Optimum	N	t [s]
handsd01	43024	172088	524	3	6	2	0.2	**171.636766**	1	0.2
handsd02	44084	178448	1584	0	0	0	0.8	**159.751395**	1	0.8
handsd03	43213	173222	713	3	6	2	0.2	**31.306275**	1	0.2
handsd04	43842	176996	1342	3	6	2	0.7	**491.733164**	1	0.7
handsd05	43205	173174	705	3	6	2	0.2	**21.937611**	1	0.2
handsd06	44477	180806	1977	3	6	2	0.6	**279.90313**	1	0.6
handsd07	43176	173000	676	3	6	2	0.2	**11.80412**	1	0.2
handsd08	44624	181688	2124	3	6	2	0.7	**143.237729**	1	0.7
handsd09	43183	173042	683	3	6	2	0.4	**3.818683**	1	0.4
handsd10	42806	170780	306	3	6	2	0.2	**1034.76736**	1	0.2
handsi01	40033	160000	433	3	6	2	0.2	**295.453616**	1	0.2
handsi02	41304	167626	1704	3	6	2	0.5	**125.429411**	1	0.5
handsi03	40220	161122	620	3	6	2	0.2	**56.149422**	1	0.2
handsi04	41030	165982	1430	3	6	2	0.7	**722.508197**	1	0.7
handsi05	40188	160930	588	3	6	2	0.2	**35.043506**	1	0.2
handsi06	41513	168880	1913	0	0	0	0.6	**452.953621**	1	0.6
handsi07	40203	161020	603	3	6	2	0.2	**18.410135**	1	0.2
handsi08	41597	169384	1997	3	6	2	0.3	**229.52993**	1	0.3
handsi09	40213	161080	613	3	6	2	0.3	**5.962166**	1	0.3
handsi10	40966	165598	1366	333	1010	160	1.5	**1803.69751**	1	1.5

Table B.29. Detailed computational results for PCSTP, test-set PCSPG-PUCNU.

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Dual	Primal	Gap %	N	t [s]
bip42nu	1307	8600	107	989	7216	106	1.9	**226**			1490	327.5
bip52nu	2303	16606	103	1818	14646	102	1.7	**222**			440	362.7
bip62nu	1303	20616	103	1199	20000	102	2.6	210.941435	214	1.4	3306	>7200.0
bipa2nu	3439	36974	139	3279	36418	139	2.8	320.429805	326	1.7	1	>7200.0
bipe2nu	576	10176	26	550	10026	25	0.7	**53**			13	16.2
cc10-2nu	1090	10630	66	981	9360	51	1.4	**167**			33	463.8
cc11-2nu	2174	23276	126	1970	20752	101	5.4	300.589765	304	1.1	234	>7200.0
cc12-2nu	4323	50504	227	3923	45066	165	8.4	559.264693	563	0.7	99	>7200.1
cc3-10nu	1019	27108	19	1007	16228	18	3.2	**61**			1	57.1

cont. next page

Instance	Original $\|V\|$	$\|A\|$	$\|T\|$	Presolved $\|V\|$	$\|A\|$	$\|T\|$	t [s]	Dual	Primal	Gap %	N	t [s]
cc3-11nu	1366	40134	35	1331	23460	34	6.5	**79**			1	11.7
cc3-12nu	1769	57264	41	1739	32648	40	9.0	**95**			1	16.9
cc3-4nu	70	606	6	3	6	2	0.0	**10**			1	0.0
cc3-5nu	134	1548	9	3	6	2	0.0	**17**			1	0.0
cc5-3nu	257	2508	14	3	6	2	0.0	**36**			1	0.0
cc6-2nu	70	414	6	3	6	2	0.0	**15**			1	0.0
cc6-3nu	768	8964	39	606	4938	29	1.4	**95**			1	1.9
cc7-3nu	2303	31306	116	2108	25686	91	3.5	268.232062	270	0.7	207	>7200.1
cc9-2nu	542	4782	30	385	2428	27	0.4	**83**			1	0.6

Table B.30. Detailed computational results for PCSTP, test-set PCSPG-Random.

Instance	Original $\|V\|$	$\|A\|$	$\|T\|$	Presolved $\|V\|$	$\|A\|$	$\|T\|$	t [s]	Optimum	N	t [s]
a0200RandGraph12	234	3470	34	3	6	2	0.0	**122.214525**	1	0.0
a0200RandGraph15	277	3606	77	3	6	2	0.0	**141.877157**	1	0.0
a0200RandGraph2	317	3906	117	3	6	2	0.0	**157.017004**	1	0.0
a0200RandGraph3	343	4084	143	3	6	2	0.0	**170.286354**	1	0.0
a0400RandGraph12	464	6766	64	3	6	2	0.0	**234.981814**	1	0.0
a0400RandGraph15	547	7338	147	3	6	2	0.0	**272.87495**	1	0.0
a0400RandGraph2	591	7724	191	3	6	2	0.0	**300.920525**	1	0.0
a0400RandGraph3	667	8040	267	3	6	2	0.0	**337.596008**	1	0.0
a0600RandGraph12	708	10284	108	3	6	2	0.0	**360.393503**	1	0.0
a0600RandGraph15	811	10950	211	3	6	2	0.0	**407.632071**	1	0.0
a0600RandGraph2	900	11456	300	3	6	2	0.0	**460.016305**	1	0.0
a0600RandGraph3	995	11980	395	3	6	2	0.0	**507.937086**	1	0.0
a0800RandGraph12	924	13644	124	3	6	2	0.0	**464.776231**	1	0.0
a0800RandGraph15	1066	14192	266	3	6	2	0.0	**530.442058**	1	0.0
a0800RandGraph2	1196	15300	396	3	6	2	0.0	**603.326182**	1	0.0
a0800RandGraph3	1336	15980	536	3	6	2	0.0	**663.607078**	1	0.0
a10000RandGraph12	11669	170604	1669	0	0	0	1.9	**5927.32057**	1	1.9
a10000RandGraph15	13371	180796	3371	3	6	2	1.8	**6775.54991**	1	1.8
a10000RandGraph2	15018	189918	5018	3	6	2	0.8	**7594.383**	1	0.8
a10000RandGraph3	16680	199630	6680	3	6	2	0.7	**8422.56095**	1	0.7
a1000RandGraph12	14039	204414	2039	5310	24894	993	8.8	**7073.94654**	1	16.4
a1000RandGraph15	15987	216698	3987	0	0	0	2.5	**8084.12787**	1	2.5
a1000RandGraph2	18002	227980	6002	3	6	2	1.2	**9064.24425**	1	1.2
a1000RandGraph3	19931	240478	7931	3	6	2	1.0	**10061.8205**	1	1.0
a1200RandGraph12	1383	19988	183	0	0	0	0.1	**705.672644**	1	0.1
a1200RandGraph15	1603	21662	403	3	6	2	0.0	**810.482934**	1	0.0
a1200RandGraph2	1787	22608	587	3	6	2	0.0	**906.792746**	1	0.0
a1200RandGraph3	2011	23762	811	3	6	2	0.0	**1012.4502**	1	0.0
a14000RandGraph12	16302	237838	2302	4082	17488	947	8.7	**8271.46523**	1	13.6
a14000RandGraph15	18688	252578	4688	3	6	2	3.7	**9475.59356**	1	3.7
a14000RandGraph2	20999	266726	6999	3	6	2	1.6	**10639.2035**	1	1.6
a14000RandGraph3	23275	279382	9275	3	6	2	1.3	**11776.8943**	1	1.3
a1400RandGraph12	1626	23734	226	3	6	2	0.1	**810.633664**	1	0.1
a1400RandGraph15	1848	25134	448	3	6	2	0.1	**938.932467**	1	0.1
a1400RandGraph2	2080	26274	680	3	6	2	0.0	**1051.01074**	1	0.0
a1400RandGraph3	2325	28070	925	3	6	2	0.0	**1158.95534**	1	0.0
a1600RandGraph12	1871	27358	271	51	158	29	0.1	**943.735583**	1	0.1
a1600RandGraph15	2114	28556	514	3	6	2	0.1	**1078.79731**	1	0.1
a1600RandGraph2	2369	30166	769	3	6	2	0.0	**1217.05199**	1	0.0
a1600RandGraph3	2677	32382	1077	3	6	2	0.0	**1351.98377**	1	0.0
a1800RandGraph12	2111	30806	311	3	6	2	0.1	**1061.39359**	1	0.1
a1800RandGraph15	2412	32110	612	3	6	2	0.1	**1218.77778**	1	0.1
a1800RandGraph2	2692	34004	892	3	6	2	0.0	**1364.89276**	1	0.0

cont. next page

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	Presolved t [s]	Optimum	N	t [s]
a1800RandGraph3	2993	36214	1193	3	6	2	0.0	**1507.26619**	1	0.0
a2000RandGraph12	2314	33894	314	0	0	0	0.1	**1151.95327**	1	0.1
a2000RandGraph15	2626	35420	626	3	6	2	0.1	**1330.77363**	1	0.1
a2000RandGraph2	2967	37920	967	3	6	2	0.1	**1483.8368**	1	0.1
a2000RandGraph3	3295	39266	1295	3	6	2	0.0	**1669.34571**	1	0.0
a3000RandGraph12	3490	51024	490	3	6	2	0.3	**1781.19442**	1	0.3
a3000RandGraph15	3995	53668	995	3	6	2	0.2	**2028.61995**	1	0.2
a3000RandGraph2	4516	57220	1516	3	6	2	0.1	**2282.91749**	1	0.1
a3000RandGraph3	4993	60004	1993	3	6	2	0.1	**2537.20275**	1	0.1
a4000RandGraph12	4669	68182	669	3	6	2	0.4	**2396.91987**	1	0.4
a4000RandGraph15	5378	72500	1378	3	6	2	0.3	**2735.1789**	1	0.3
a4000RandGraph2	6040	75994	2040	3	6	2	0.1	**3072.26147**	1	0.1
a4000RandGraph3	6692	80196	2692	3	6	2	0.1	**3406.61873**	1	0.1
a6000RandGraph12	6969	101606	969	292	1018	133	1.0	**3544.38604**	1	1.1
a6000RandGraph15	7959	107902	1959	3	6	2	0.8	**4059.18665**	1	0.8
a6000RandGraph2	8993	114090	2993	3	6	2	0.3	**4551.76667**	1	0.3
a6000RandGraph3	9982	119716	3982	3	6	2	0.3	**5049.26346**	1	0.3
a8000RandGraph12	9343	136798	1343	0	0	0	1.6	**4719.96527**	1	1.7
a8000RandGraph15	10646	143494	2646	0	0	0	1.7	**5394.56802**	1	1.7
a8000RandGraph2	11967	151544	3967	3	6	2	0.5	**6055.12642**	1	0.5
a8000RandGraph3	13300	160148	5300	3	6	2	0.5	**6710.61511**	1	0.5

B.4 Steiner arborescence problem

The time limit for the following and all remaining instances in this thesis is two hours.

Table B.31. Detailed computational results for SAP, test-set SAP-gene.

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	Presolved t [s]	Optimum	N	t [s]
gene41x	335	910	43	0	0	0	0.0	**126**	1	0.0
gene42	335	912	43	0	0	0	0.0	**126**	1	0.0
gene61a	395	1024	82	0	0	0	0.0	**205**	1	0.0
gene61b	570	1616	82	0	0	0	0.0	**199**	1	0.0
gene61c	549	1580	82	0	0	0	0.0	**196**	1	0.0
gene61f	412	1104	82	0	0	0	0.0	**198**	1	0.0
gene425	425	1108	86	0	0	0	0.0	**214**	1	0.0
gene442	442	1188	86	0	0	0	0.0	**207**	1	0.0
gene575	575	1648	86	0	0	0	0.0	**207**	1	0.0
gene602	602	1716	86	0	0	0	0.0	**209**	1	0.0

Table B.32. Detailed computational results for SAP, test-set SAP-gene2002.

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	Presolved t [s]	Optimum	N	t [s]
microtri1	347	952	47	0	0	0	0.0	**128**	1	0.0
microtri3	400	1112	47	0	0	0	0.0	**146**	1	0.0
microtri5	416	1124	47	0	0	0	0.0	**150**	1	0.0
microtri6	419	1164	47	0	0	0	0.0	**146**	1	0.0
microtri7	437	1172	47	0	0	0	0.0	**159**	1	0.0
microtri8	484	1412	47	0	0	0	0.0	**151**	1	0.0
microtri9	297	792	47	0	0	0	0.0	**131**	1	0.0
microtri10	319	836	47	0	0	0	0.0	**136**	1	0.0
microtri11	382	1024	47	0	0	0	0.0	**152**	1	0.0

Table B.33. Detailed computational results for SAP, test-set SAP-NET.

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	Presolved t [s]	Optimum	N	t [s]
ID141_1	77120	232714	8	0	0	0	3.2	**445143.514**	1	3.2
ID141_2	77120	232714	3	0	0	0	3.0	**127899.377**	1	3.0
ID141_3	77120	232714	3	0	0	0	3.2	**130314.802**	1	3.2
ID141_4	77120	232714	5	0	0	0	3.4	**806360.212**	1	3.5
ID141_5	77120	232714	11	1705	6674	11	2.7	**1371182.7**	1	2.8
ID313_0	88328	272968	62	0	0	0	11.7	**3e-08**	1	11.7
ID313_10	88328	272968	7	103	316	7	6.7	**785402.452**	1	6.7
ID313_11	88328	272968	8	0	0	0	4.7	**460175.719**	1	4.8
ID313_12	88328	272968	3	0	0	0	4.4	**60735.3624**	1	4.4
ID313_13	88328	272968	4	0	0	0	5.9	**378449.624**	1	5.9
ID313_14	88328	272968	4	0	0	0	5.3	**413106.771**	1	5.4
ID313_1	88328	272968	6	364	1450	4	4.6	**268092.533**	1	4.7
ID313_2	88328	272968	7	0	0	0	4.5	**363173.432**	1	4.5
ID313_3	88328	272968	7	162	638	6	4.5	**336031.786**	1	4.5
ID313_4	88328	272968	4	0	0	0	4.5	**108384.996**	1	4.5
ID313_5	88328	272968	4	0	0	0	4.4	**137775.062**	1	4.4
ID313_6	88328	272968	2	0	0	0	3.9	**40213.8773**	1	4.0
ID313_7	88328	272968	3	0	0	0	4.4	**209186.362**	1	4.5
ID313_8	88328	272968	10	0	0	0	4.6	**395117.997**	1	4.6
ID313_9	88328	272968	6	1926	7674	6	10.0	**1012291.08**	1	10.1
ID314_0	225739	642338	225	171637	452808	1	27.2	**1e-08**	1	28.8

cont. next page

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	Presolved t [s]	Optimum	N	t [s]
ID314_1	225739	642338	132	56330	213562	132	34.0	**20198383.1**	1	1486.8
ID314_2	225739	642338	26	23510	93554	26	29.4	**1604154.66**	1	450.8
ID314_3	225739	642338	30	16824	66820	30	31.2	**2689408.05**	1	101.6
ID314_4	225739	642338	40	0	0	0	7.5	**5723922.1**	1	7.5

B.5 Euclidean Steiner tree problem

Table B.34. Detailed computational results for ESMT, test-set ESMT-R25.

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	Presolved t [s]	Optimum	N	t [s]
R25K01EFST	39277	94524	25000	92	294	40	34.7	**98.9612134**	1	40.1
R25K02EFST	39306	94978	25000	59	180	30	38.4	**99.0370878**	1	47.1
R25K03EFST	39549	96348	25000	3893	12466	1785	35.0	**99.2157207**	1	45.8
R25K04EFST	39555	96260	25000	84	274	37	38.0	**98.9431392**	1	47.6
R25K05EFST	39153	93806	25000	49	146	26	28.9	**99.4912321**	1	39.1
R25K06EFST	39438	95690	25000	5990	19160	2804	12.7	**99.3728768**	1	29.6
R25K07EFST	39900	98180	25000	47	140	24	38.4	**99.5646105**	1	51.6
R25K08EFST	39529	95920	25000	65	200	32	39.6	**99.2662017**	1	48.5
R25K09EFST	39732	97060	25000	3807	12238	1773	38.1	**99.0968636**	1	44.7
R25K10EFST	39248	94668	25000	48	136	23	28.1	**99.1104801**	1	35.7
R25K11EFST	39425	95470	25000	2661	8418	1239	42.1	**99.1216345**	1	47.5
R25K12EFST	39293	94888	25000	3434	10960	1593	37.3	**99.1134447**	1	45.5
R25K13EFST	39284	94770	25000	3328	10524	1566	26.4	**99.4005526**	1	33.0
R25K14EFST	40063	98534	25000	3957	12746	1795	38.9	**99.2046414**	1	46.1
R25K15EFST	39498	95704	25000	44	130	21	43.7	**99.2521324**	1	54.6

Table B.35. Detailed computational results for ESMT, test-set ESMT-R50.

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	Presolved t [s]	Optimum	N	t [s]
R50K01EFST	79505	194746	50000	9521	30346	4427	120.2	**140.398764**	1	139.8
R50K02EFST	78754	190726	50000	254	798	119	116.2	**139.955781**	1	145.3
R50K03EFST	78964	191358	50000	7198	23066	3325	126.2	**140.006412**	1	140.3
R50K04EFST	78983	191484	50000	56	174	28	142.5	**140.093852**	1	154.3
R50K05EFST	79200	193418	50000	43	126	23	142.7	**139.995235**	1	196.5
R50K06EFST	79480	194744	50000	9705	31120	4495	133.9	**140.348542**	1	164.9
R50K07EFST	79046	192228	50000	13631	43506	6350	44.2	**140.249582**	1	86.0
R50K08EFST	79175	192822	50000	108	378	41	111.7	**140.351147**	1	126.8
R50K09EFST	78825	190952	50000	54	156	26	107.0	**140.363481**	1	120.6
R50K10EFST	78948	191740	50000	73	236	33	40.8	**140.321093**	1	80.0
R50K11EFST	79121	192608	50000	8136	25928	3797	122.1	**140.169756**	1	139.6
R50K12EFST	79133	192768	50000	51	156	24	29.3	**140.201234**	1	57.7
R50K13EFST	78972	191348	50000	7654	24410	3562	116.3	**140.03999**	1	131.6
R50K14EFST	79326	193440	50000	51	156	24	135.7	**140.209795**	1	151.8
R50K15EFST	79483	194414	50000	66	224	25	156.4	**140.447926**	1	170.4

Table B.36. Detailed computational results for ESMT, test-set ESMT-R100.

Instance	Original $\|V\|$	Original $\|A\|$	Original $\|T\|$	Presolved $\|V\|$	Presolved $\|A\|$	Presolved $\|T\|$	Presolved t [s]	Optimum	N	t [s]
R100K01EFST	157869	383172	100000	50	156	24	554.3	**198.306705**	1	631.7
R100K02EFST	158031	383994	100000	71	244	26	156.6	**197.970178**	1	370.1
R100K03EFST	158290	384990	100000	18337	58020	8630	477.9	**198.022313**	1	640.7
R100K04EFST	158205	385292	100000	64	204	29	550.8	**198.189607**	1	612.3
R100K05EFST	158587	386646	100000	47	146	22	123.6	**198.153237**	1	372.1

cont. next page

Instance	Original $\|V\|$	$\|A\|$	$\|T\|$	Presolved $\|V\|$	$\|A\|$	$\|T\|$	t [s]	Optimum	N	t [s]
R100K06EFST	158514	386086	100000	73	276	25	117.7	**198.174481**	1	328.3
R100K07EFST	157947	383296	100000	24847	79452	11545	112.4	**197.878416**	1	369.6
R100K08EFST	157839	382404	100000	17086	54446	7977	471.3	**197.994433**	1	541.8
R100K09EFST	158069	383470	100000	26909	85924	12571	152.3	**198.135832**	1	391.4
R100K10EFST	158575	386344	100000	18012	57070	8424	448.3	**198.039905**	1	546.1
R100K11EFST	158265	385490	100000	25924	83376	11924	124.7	**198.136332**	1	368.6
R100K12EFST	157806	382352	100000	50	158	22	472.6	**198.384696**	1	570.0
R100K13EFST	157660	381462	100000	27619	87894	12879	151.4	**198.053544**	1	457.4
R100K14EFST	158516	386662	100000	50	162	22	510.5	**198.226993**	1	729.7
R100K15EFST	158033	384044	100000	27235	87188	12652	151.3	**198.274048**	1	460.0

B.6 The degree constrained Steiner tree problem

Table B.37. Detailed computational results for DCSTP, test-set TreeFam.

Instance	Original $\|V\|$	$\|A\|$	$\|T\|$	Presolved $\|V\|$	$\|A\|$	$\|T\|$	t [s]	Dual	Primal	Gap %	N	t [s]
TF101057-t1	52	2652	35	52	1462	35	0.0	**infeasible**			1	0.0
TF101057-t3	52	2652	35	52	2648	35	0.0	**2756**			362	2.1
TF101125-t1	304	92112	155	304	68242	155	4.3	**infeasible**			1	5.4
TF101125-t3	304	92112	155	304	92112	155	3.0	**55185**			302	178.8
TF101202-t1	188	35156	72	188	30044	72	0.7	**79920**			47303	2542.0
TF101202-t3	188	35156	72	188	35156	72	0.6	**77978**			4894	390.2
TF102003-t1	832	691392	407	832	526150	407	84.0	193812.178	419049	116.2	122	>7200.4
TF102003-t3	832	691392	407	832	691392	407	65.5	181328.493	188671	4.0	56	>7200.1
TF105035-t1	237	55932	104	237	45220	104	1.8	35058.2504	36311	3.6	38058	>7200.0
TF105035-t3	237	55932	104	237	55932	104	1.3	**32916**			396	112.8
TF105272-t1	476	226100	223	476	176594	223	13.7	134857.053	211138	56.6	2065	>7200.2
TF105272-t3	476	226100	223	476	226100	223	8.7	126868.798	127352	0.4	5431	>7200.3
TF105419-t1	55	2970	24	55	2418	24	0.0	**18668**			4626	19.0
TF105419-t3	55	2970	24	55	2286	24	0.0	**18223**			5	0.2
TF105897-t1	314	98282	133	314	80726	133	3.4	108066.522	114092	5.6	14151	>7200.0
TF105897-t3	314	98282	133	314	98282	133	2.9	**97832**			2650	1134.3
TF106403-t1	119	14042	46	119	11972	46	0.2	**54124**			1054	15.4
TF106403-t3	119	14042	46	119	4710	46	0.3	**53760**			1	0.4
TF106478-t1	130	16770	54	130	13908	54	0.3	**55132**			3809	62.4
TF106478-t3	130	16770	54	130	16174	54	0.4	**54839**			390	13.6

B.7 The group Steiner tree problem

Table B.38. Detailed computational results for GSTP, test-set GSTP1.

Instance	Original $\|V\|$	$\|A\|$	$\|T\|$	Presolved $\|V\|$	$\|A\|$	$\|T\|$	t [s]	Optimum	N	t [s]
gstp30f2	474	1828	30	0	0	0	0.1	**569**	1	0.1
gstp31f2	349	1284	31	0	0	0	0.1	**635**	1	0.1
gstp33f2	452	1746	33	0	0	0	0.0	**513**	1	0.0
gstp34f2	1253	5000	34	1033	4132	31	3.8	**646**	1	5.1
gstp36f2	442	1672	36	134	484	20	0.6	**610**	1	0.6
gstp37f2	1054	4216	37	978	3916	37	1.4	**485**	1	3.8
gstp38f2	618	2504	38	460	1892	37	1.4	**656**	1	2.5
gstp39f2	707	3310	39	623	2938	38	2.5	**450**	1	5.0

Table B.39. Detailed computational results for GSTP, test-set GSTP2.

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Optimum	N	t [s]
gstp50f2	1142	4622	50	930	3794	49	3.3	**673**	1	5.0
gstp55f2	1751	6804	55	1565	6176	55	2.8	**888**	1	65.7
gstp60f2	838	3528	60	763	3184	60	4.5	**1164**	1	27.6
gstp64f2	1860	7380	64	1640	6604	60	3.2	**931**	1	9.7
gstp66f2	2623	10100	66	2339	9224	62	4.7	**920**	1	393.4
gstp73f2	1911	7308	73	1704	6644	63	3.0	**1207**	1	38.9
gstp76f2	1818	6990	76	1133	4434	47	3.1	**1026**	1	5.7
gstp78f2	2355	9384	78	2009	8080	71	4.7	**1094**	1	315.0
gstp83f2	3177	12530	83	2975	11954	80	4.4	**906**	1	3451.2
gstp84f2	2358	9134	84	1989	7912	73	4.0	**1094**	1	27.2

B.8 Hop constrained directed Steiner tree problems

Table B.40. Detailed computational results for HCDSTP, test-set gr12.

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Optimum	N	t [s]
wo11-gr12-cr100-se10	809	7432	10	106	686	10	0.0	**136516**	1	0.1
wo11-gr12-cr100-se11	809	7430	10	25	48	10	0.0	**145251**	1	0.0
wo11-gr12-cr100-se1	809	7444	10	318	2978	10	0.1	**182082**	1	0.2
wo11-gr12-cr100-se2	809	7394	10	31	60	10	0.0	**163872**	1	0.0
wo11-gr12-cr200-se11	809	15260	10	393	5708	10	0.1	**66786**	1	0.3
wo11-gr12-cr200-se1	809	15274	10	549	10626	10	0.1	**76353**	1	0.8
wo11-gr12-cr200-se2	809	15224	10	23	44	10	0.0	**75434**	1	0.0
wo12-gr12-cr100-se10	809	9360	10	52	198	10	0.0	**167223**	1	0.0
wo10-gr12-cr100-se10	809	14428	10	27	52	10	0.0	**117081**	1	0.0
wo12-gr12-cr100-se11	809	9852	10	359	3686	10	0.1	**199679**	1	0.2
wo12-gr12-cr100-se1	809	9446	10	209	1870	10	0.0	**164198**	1	0.1
wo12-gr12-cr100-se7	809	9702	10	24	46	10	0.0	**136232**	1	0.0
wo12-gr12-cr200-se9	809	28346	10	17	32	10	0.0	**46408**	1	0.0
wo10-gr12-cr100-se0	809	14396	10	774	13328	10	0.1	**171486**	1	6.0
wo10-gr12-cr100-se11	809	14386	10	427	6672	10	0.1	**125785**	1	0.8
wo10-gr12-cr200-se7	809	44696	10	131	1648	10	0.1	**46306**	1	0.1
wo10-gr12-cr200-se8	809	44654	10	779	34890	10	0.5	**61177**	1	6.8
wo10-gr12-cr200-se9	809	44670	10	363	8946	10	0.1	**51737**	1	0.6

Table B.41. Detailed computational results for HCDSTP, test-set gr14.

Instance	Original $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	Presolved $\lvert V\rvert$	$\lvert A\rvert$	$\lvert T\rvert$	t [s]	Optimum	N	t [s]
wo10-gr14-cr100-se0	3209	215940	10	3178	212040	10	5.8	**163373**	3	2130.8
wo10-gr14-cr100-se11	3209	215932	10	1408	75734	10	5.6	**120466**	1	28.9
wo10-gr14-cr200-se3	3209	643552	10	2615	317044	10	41.0	**52425**	1	358.9
wo10-gr14-cr200-se4	3209	643414	10	2884	479482	10	23.9	**51592**	1	699.9
wo11-gr14-cr100-se6	3209	115502	10	2687	108314	10	2.2	**211758**	81	1905.5
wo11-gr14-cr200-se2	3209	232858	10	2453	170432	10	3.4	**71134**	1	59.4
wo11-gr14-cr200-se3	3209	233104	10	19	36	10	0.4	**57930**	1	0.4
wo11-gr14-cr200-se4	3209	233038	10	2458	176768	10	5.2	**63313**	1	53.6
wo12-gr14-cr100-se0	3209	153366	10	687	25696	10	0.7	**118617**	1	3.6
wo12-gr14-cr100-se5	3209	156578	10	919	37024	10	0.8	**131631**	1	12.8
wo12-gr14-cr100-se6	3209	157214	10	1521	66260	10	2.5	**146049**	1	76.0
wo12-gr14-cr100-se7	3209	158984	10	812	35402	10	1.1	**122306**	1	13.3
wo12-gr14-cr100-se8	3209	157912	10	880	37954	10	1.1	**116077**	1	22.3
wo12-gr14-cr100-se9	3209	156658	10	163	1692	10	0.2	**100813**	1	0.2
wo12-gr14-cr200-se0	3209	445774	10	1029	71640	10	3.7	**53883**	1	19.2
wo12-gr14-cr200-se10	3209	446040	10	2160	262644	10	22.6	**64582**	3	1009.4
wo12-gr14-cr200-se11	3209	457496	10	2339	318702	10	8.4	**69143**	3	860.7

cont. next page

Instance	Original $\lvert V\rvert$	Original $\lvert A\rvert$	Original $\lvert T\rvert$	Presolved $\lvert V\rvert$	Presolved $\lvert A\rvert$	Presolved $\lvert T\rvert$	Presolved t [s]	Optimum	N	t [s]
wo12-gr14-cr200-se4	3209	460250	10	2288	263776	10	12.7	**72993**	7	1254.8
wo12-gr14-cr200-se5	3209	456998	10	1458	126506	10	5.3	**57694**	1	120.0
wo12-gr14-cr200-se6	3209	460500	10	2246	265226	10	20.6	**61925**	7	701.6
wo12-gr14-cr200-se7	3209	464090	10	1437	150164	10	5.0	**61370**	3	89.6