Optimal Graph Coverings with Connected Subgraphs

Stephan Schwartz

Dissertation

zur Erlangung des Grades eines
Doktors der Naturwissenschaften

am Fachbereich Mathematik und Informatik
der Freien Universität Berlin

Berlin, 2023

Erstgutachter & Betreuer:
Prof. Dr. Ralf Borndörfer

Zweitgutachter:
PD Dr. Timo Berthold

Tag der Disputation:
29. März 2023

Acknowledgments

Working at ZIB for the past eight years has been a great experience. I learned a lot about optimization, modeling, and coding, and I got to know so many nice people.

This is only possible because my supervisor Ralf Borndörfer took me as a Master's student and welcomed me in his group. Thank you, Ralf, for your trust and support, for your motivational speeches that can fill half-empty glasses up to the rim, and for sharing some of your great skills with me. I am thankful for the discussions with you on the math, the structure, and the presentation of our papers and lectures.

I would also like to thank Timo Berthold for his willingness to be my co-supervisor for this thesis. When I started working at ZIB, your open-minded nature (and your taste in music) helped me integrating into the optimization group. Therefore, I am especially glad and thankful that you accepted right away when I asked you.

Most of the topics that I worked on within the research project with the Bundesamt für Güterverkehr are disjoint to the present thesis. I would like to thank my colleagues who worked with me on this project over the years: Elmar Swarat, Guillaume Sagnol, Boris Grimm, Markus Reuther, and Torsten Klug. Special thanks go to Thomas Schlechte for being the social anchor of the network optimization group, looking out for everybody, and for being the person I could always ask for advice.

I am also grateful for all other past and present members of the (network) optimization group for enjoying (lunch) breaks with discussions on various (non)mathematical topics.

I am indebted to my student collaborators, the "End-of-Alphabet-Crew": Vanessa Schreck, William Surau, and Ziena Zeif. Joint work with either one of you is part of this thesis, and working together with you was always fun and productive.

Finally, I would like to thank my family and friends for their unconditional support over the years and, especially, towards the end. A special thank you goes out to my parents and to Leo in this context. Most of all, however, I want to thank my wonderful wife, Tine. You and I both know that this thesis would simply not be finished now, if it weren't for your support. Thank you for everything! And thank you, Timo and Antonia, for persistently pulling me away from my computer to free up my mind.

Contents

1 Introduction

Graph covering problems are among the most classical and central subjects in graph theory. They also play a huge role in many mathematical models for various real-world applications. Be it the design of police patrol areas, sales territories, or voting districts, or be it a terrain coverage with multiple robots, e.g., for cleaning, harvesting, or security: All these problems come down to a covering problem of a graph. Moreover, as these examples show, it is often desired that each subgraph of the covering is connected. In most cases, additional constraints on the covering subgraphs, which are also called *templates*, are imposed. Most notable are lower or upper bounds on the "size" of each template, or the requirement that the subgraphs have to be disjoint. The latter results in a graph partitioning problem which we also consider in this thesis.

But clearly, we are not content with just any graph covering. The objective of the covering differs from application to application. Police districts could be designed with regard to minimal expected response time, while for cleaning robots the total completion time would be important. In other applications, the objective is to minimize the number of templates that is necessary for the covering. We will come back to the notion of "optimal" covering later, and first differentiate the graph covering problem between two fundamental concepts.

The two variants of the graph covering problem are concerned with covering the edges or, respectively, the vertices of a graph. Both draw a lot of scientific attention and are subject to prolific research. Interestingly, however, the field is divided into two practically independent parts: The edge covering problem belongs to the graph theory community, and the contributions often comprise theoretical bounds or (in)approximability results. Vertex covering problems, on the other hand, are mostly motivated by applications, and advances are mainly driven by the operations research community. Consequently, most contributions contain heuristic or approximation algorithms, integer programming (IP) formulations, or polyhedral studies to strengthen the respective linear programming (LP) relaxations.

While we will review both fields in this thesis, our focus clearly lies on the vertex covering problem, and the reasons therefore are threefold: First, the industry project funding and driving this research is concerned with a districting problem, which clearly falls into the realm of vertex covering, at least from the standpoint of the literature. Second, observe that by considering the line graph, an edge covering with connected subgraphs corresponds to a vertex covering with trees. While this is not a one-to-one relation, the problems are equivalent in the sense that any solution of one problem can be transformed into an equivalent solution of the other problem (cf. Section 2.7). And third, the author of this thesis must admit that he finds the techniques of the operations research community far more appealing than the seemingly dry topics of graph theory.

(a) balanced templates (b) homogeneous templates

Figure 1.1.: Example for balanced and homogenous templates. The numbers indicate vertex weights.

The two most common goals for vertex covering problems are to use balanced or, respectively, homogeneous templates. For the balancing objective, the templates shall be as similar as possible, while for the homogeneity goal, each template graph is supposed to be as homogeneous as possible. Figure 1.1 illustrates solutions with respect to each objective. Note that homogeneity and balancing are not necessarily opposing objectives. In many cases both are pursued, e.g., if we are seeking a cover that is homogeneous with respect to given vertex weights but balanced with respect to the cardinality of the templates.

Motivation: Our research is motivated by a concrete real-world application of designing optimal toll control sections for inspectors on German motorways. A detailed introduction and analysis of this Toll Section Design Problem is given in Section 6.1. Here, we give a brief overview: Given a road network with lengths and traffic volumes on the edges, our goal is to find a covering of the network with smaller, contiguous control areas. Homogeneous traffic within each control area is an operative goal. Furthermore, the areas are subject to lower and upper length bounds. As formulated, the problem is an edge covering problem with various side constraints and a fuzzy homogeneity objective. For now, we take this problem as a motivation for a journey through the field of graph covering, and revisit this application in Chapter 6.

Contributions and Structure of this Thesis

In this thesis, we give a holistic overview of graph covering and partitioning problems, together with their different formulations and solution techniques. While graph covering is a central topic in graph theory, there has not been a similar attempt to survey and categorize the whole field. Even for the subfield of vertex covering (or rather partitioning) problems, we are not aware of another systematical review of problems and approaches. Our overview of the field and its division into three practically independent and methodically different parts is shown in Figure 1.2.

Apart from this structuring of the field, we provide contributions to different variants of the problem in the form of newly proposed approximation algorithms and primal heuristics, preprocessing techniques and exact formulations. Concerning the critical subproblem of finding a single connected subgraph subject to various constraints, we compare known IP formulations for modeling connectivity within subgraphs, and we propose a number of valid inequalities to strengthen the LP relaxation of our problem formulation. Finally, we combine all these contributions and put them into practice by modeling and solving the aforementioned Toll Section Design Problem within a branch-and-price-and-cut framework. In detail, this thesis is structured as follows.

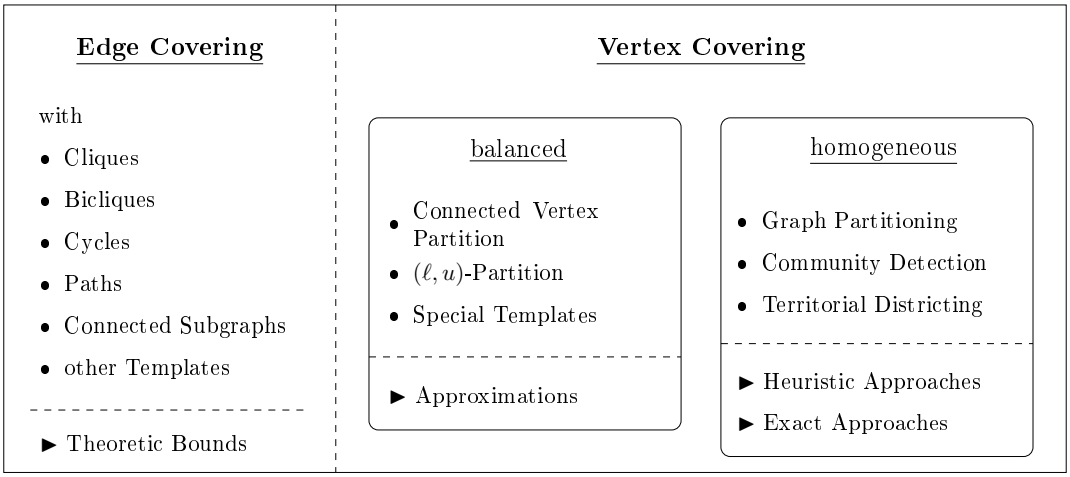


Figure 1.2.: An overview of graph covering problems and corresponding predominantly used mathematical techniques or objectives.

In Chapter 2, we review the literature on edge covering. Our presentation is organized along the lines of possible template classes such as cliques, cycles, or connected subgraphs. The contribution of this chapter is a comprehensive overview and categorization of the problems, techniques and notable results in this topic. It is the first attempt to organize and survey the literature on edge covering problems since Pyber's article in 1991 [Pyb91].

With Chapter 3, we shift our focus to vertex covering and we start with the balanced version. We survey the most important variants and results which mostly concern approximations. In line with this, we examine the main variant of this field and present new approximation algorithms for this case. In addition, we provide an elegant way to transform and improve this approximation result to the respective edge covering problem.

The homogeneous variant of vertex covering is the topic of Chapter 4. We begin by reviewing different options to measure homogeneity of (sub)graphs. Our comprehensive overview of the related literature is divided into heuristic and exact solution approaches. Finally, one might say, we arrive at the discussion of different IP formulations and improving techniques. These IP formulations fall into two categories: We can use a standard covering IP with one variable per possible template and, since enumerating all feasible templates is impossible, generate promising templates dynamically. Alternatively, we have formulations that assign each node to one (or more) templates, resulting in a covering of the vertex set. To close this chapter, we present a method to efficiently incorporate connectivity constraints into the latter formulation.

Chapter 5 follows the direction of generating promising templates dynamically and is concerned with finding a single feasible template (that is optimal in some sense). Due to the connectivity condition of a single template, this problem results in the well-known Maximum Weight Connected Subgraph Problem (MWCS) with additional constraints. We focus on a specific variant of this problem, the Balanced, Rooted, and Capacitated MWCS (BRCMWCS), and study this problem in-depth. We present different methods to model connectivity within an IP formulation, propose powerful reduction techniques, and provide different families of inequalities that strengthen the LP relaxation. In an extensive computational study we compare the connectivity formulations on different classes of instances, and demonstrate the effectiveness of the proposed enhancements.

Finally, Chapter 6 is devoted to modeling and solving the Toll Section Design Problem. The broad literature review from the previous chapters reveals that this problem fits best into the category of homogeneous vertex covers. In particular, our covering problem is surprisingly related to the class of districting problems. Employing and extending the approaches from this field, we find a column generation approach to be best suited. The pricing problem boils down to a capacitated MWCS with a complex objective function. We take a novel perspective and split up the pricing into many subproblems with a fixed root node. As it turns out, we can reduce each subproblem to the BRCMWCS considered in Chapter 5 and employ all improvements from there. In addition, we propose two further enhancements, a heuristic pricing and a local search, and confirm their positive impact on all real-world instances and additional artificial instances that mimic these. Finally, we demonstrate that our methods that were developed with regard to the TSDP also translate to general districting problems, and that the results of this thesis propel the state of the art for the whole field.

Preliminaries and Notation

In a graph covering problem we are given a graph G and a set of possible subgraphs of G. Following the terminology of Knauer and Ueckerdt [KU16], we call G the *host graph* while the set of possible subgraphs forms the *template class*. Elements of the template class are called template graphs or, for short, templates. An edge (resp. vertex) covering is a set of templates such that each edge (resp. vertex) of the host graph belongs to at least one template. In addition, there is a measure for the quality of a covering, e.g., the number of used templates. The graph covering problem is to find a covering that optimizes this quality measure among all coverings.

If the template graphs of a covering are pairwise disjoint, the covering is called a partitioning. In many cases the analogously defined graph partitioning problem is considered instead of the graph covering problem.

Since we can consider the connected components of a graph separately, we assume throughout this thesis that the host graph is connected, unless explicitly stated otherwise.

General Notation: In the vast majority of cases, we use the standard notation of graph theory and follow the book of Diestel [Die16]. In particular, we use the terms vertices and nodes synonymously and adopt the notion of writing $G - H$ where G is a graph and H can be a graph, a node set, or an edge set that is removed from G.

Apart from that, we use the standard notation $[i] := \{1, \dots, i\}$ for $i \in \mathbb{N}$, and for variables or parameters φ defined on a finite set X and for $X' \subseteq X$, we denote by $\varphi(X')$ the sum $\sum_{x \in X'} \varphi_x$. In addition, for a graph $G = (V, E)$ and weights φ defined on V, we also write $\varphi(G)$ instead of $\varphi(V)$. Notation beyond this is used locally and introduced when appropriate.

Publications

Significant parts of the work presented in this thesis have been published in refereed journals or conference proceedings. These include the following:

- Most of the literature review in Chapters 2, 3, and 4 has been published as a survey article on graph covering and partitioning in [1].
- The approximation results in Chapter 3 are published in [2] but our exposition mainly follows [3].
- The minimum balanced separator approach from Chapter 4 was proposed in [4].
- The BRCMWCS in Chapter 5 in combination with the preprocessing and conflict pairs has been studied in [5].
- The review on connectivity in IPs in Chapter 5 as well as the algorithmic enhancements, the branch-and-price approach, and the districting application in Chapter 6 have been published in [6].

[1] S. Schwartz. "An overview of graph covering and partitioning". In: *Discrete Mathematics* 345.8 (2022)

[2] R. Borndörfer et al. "Connected k-partition of k-connected graphs and c-claw-free graphs". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Ed. by M. Wootters and L. Sanità. Vol. 207. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 27:1–27:14

[3] R. Borndörfer, Z. Elijazyfer, and S. Schwartz. *Approximating Balanced Graph Partitions*. Tech. rep. 19-25. Zuse Institute Berlin, 2019

[4] W. Surau, R. Borndörfer, and S. Schwartz. "Finding Minimum Balanced Separators – An Exact Approach". In: *Operations Research Proceedings*. Springer. 2021, pp. 154–159

[5] R. Borndörfer, S. Schwartz, and W. Surau. "Rooted maximum weight connected subgraphs with balancing and capacity constraints". In: *Proceedings of the 10th International Network Optimization Conference (INOC), Aachen, Germany*. Springer. 2022, pp. 63–68

[6] R. Borndörfer, S. Schwartz, and W. Surau. "Vertex covering with capacitated trees". In: *Networks* 81.2 (2023), pp. 253–277. DOI: 10.1002/net.22130

2 Edge Covering

In this chapter, we overview the first part of graph coverings, namely, to cover the edges of a given host graph with templates of certain types. The investigation of edge covering problems in its present form was initiated by graph theory legends Paul Erdős [EGP66] and László Lovász [Lov68]. Since then, research is predominantly concerned with improving bounds on the covering number, i.e., the minimum number of templates necessary for an edge covering. The covering number is studied for different template classes and often, special host graphs or special coverings are considered.

As a special covering we single out the local covering number here. It is defined as the smallest number ℓ such that a covering exists where every vertex is contained in at most ℓ template graphs.

The only survey article on edge covering that we are aware of is over 30 years old and due to Pyber [Pyb91]. The focus of this chapter lies on an overview and the categorization of the literature on this topic. We will include only notable results, and organize the presentation along the lines of different template classes. Table 2.1 summarizes our classification of the relevant papers. In line with the standard notation, we denote by n the number of vertices and by m the number of edges of a graph.

2.1 Edge Covering with Cliques

The clique covering problem is the oldest variant and has been studied in different types and under different names such as *keyword conflict* or *intersection graph basis*. A number of survey articles summarize earlier results on this topic as well as a few applications [Pul83; Rob85; MPR95; Cav05].

The problem was first studied by Erdős, Goodman, and Pósa [EGP66] in terms of decomposing a graph into edges and triangles. They showed that such a decomposition can be done with at most $\lfloor n^2/4 \rfloor$ template graphs and that this bound is met for the complete bipartite graph $K_{n/2,n/2}$. Recently, Král et al. [Krá+19] extended this result and proved that any graph admits an edge partitioning into n_2 copies of K_2 and n_3 copies of K_3 such that $2n_2 + 3n_3 \leq (1/2 + o(1))n^2$, settling a conjecture of Győri and Tuza [GT87].

An upper bound for the clique covering number of general host graphs is given by Lovász [Lov68]: If $k = \binom{n}{2} - m$ and $t \in \mathbb{N}$ is maximal with $t^2 - t \leq k$, then there is always a covering with $k + t$ cliques. Another bound due to Alon [Alo86] depends on the minimum degree δ of the host graph. He proves that $2e^2(n - \delta)^2 \log_e n$ cliques suffice. Gyárfás [Gyá90] proposes a set of simple reduction techniques and derives a lower bound that is logarithmic in the size of the reduced graph. Since then, no progress has been made on the bounds for the general covering case.

Other authors study the relation between covering and partitioning of the edges. It is clear that the latter needs at least as many templates as the former. Caccetta et al.

Table 2.1.: Overview on edge covering contributions for different template classes.

	survey	general bounds	special host graphs	special coverings	other	partition
cliques	[Pul83], [Rob85], [MPR95], [Cav05]	[EGP66], [Lov68], [Alo86], [Gyá90]	[Orl77], [PC80]. [PC81], [PD81], [GP82], [Wal82], [CP83], [Pul84], [CGP85], [EFO88], [MWW89], [Hoo92], [Che+00], [CM01], [CV08], [CFS14]. [JMO16], [JH19], [Cha+21]	[PSW82], [JMO16]	[Kel73], [KSW78], [Cac+85], [DC+86], [EFO88], [Gyá90], [BT06], [Gra+07], [Gra+09], [FH15], [CPP16], [Rod21],	[EGP66], [PSW82], [Cac+85], [EFO88], [Uiy03], [Krá+19], [RUW21]
bicliques	[MPR95]	[HHM77], [Ber78], [Chu80], [Tuz84], [JK09]	[GWS99], [Bez+08]	[FH96], [EP97], [DL07]	[Orl77], [Mül96], [AVJ98], [CF06], [Wat06] [Gün07].	[GP72], [Tve82], [CES83], [EP97], [GWS99], [DL07], [Pin14], [ABH17]
cycles	[Zha97]	[EGP66], [Pyb85], [Bon90], [Fan02]		[Ita+81], [BJJ83], [AT85], [Jae85], [Fan97], [Tho97], [Fan98], [BS01], [HO01]. [IMM05], [Cha09], [Zha16]		
paths		[Chu80], [Pyb96], [Fan02]				[Lov68], [Don80], [Yan98], [Fan05], [ZL06], [BJ17], [Gir+21]

[Cac+85] investigate the difference between clique covering number and clique partitioning number, providing a lower and an upper bound on the worst-case difference on any graph with n vertices. Erdős, Faudree, and Ordman [EFO88], on the other hand, focus on the ratio of the two numbers. They find that this ratio cannot be larger than $\frac{1}{12}n^2$ for any graph on n vertices.

Concerning the complexity of finding the clique covering number, Kou, Stockmeyer, and Wong [KSW78] and, independently, Orlin [Orl77] proved that clique covering is NP-hard. The former also proved that there is no 2-approximation unless P=NP, and later it was shown that no approximation within a factor of n^ε for some $\varepsilon > 0$ is possible [LY94].

More can be said about specific host classes: The problem remains NP-hard if the host graph is planar [CM01], but in this case a polynomial-time approximation scheme is known [BKV12]. The problem is also NP-hard for host graphs of maximum degree 6 [Hoo92], but polynomially solvable for smaller maximum degrees [Pul84; Hoo92]. It can also be solved in polynomial time for chordal graphs [MWW89], line graphs [Orl77], and host graphs with bounded treewidth [BKV12]. Other works consider the clique covering problem on claw-free graphs [JMO16; JH19; Cha+21], on regular graphs [PC80; PC81; CP83], or on split graphs [WW91]. Furthermore, the clique covering problem has been considered for complements of graphs. While Wallis [Wal82] derives bounds for complements of general graphs, other articles focus on complementary graphs of certain graph classes such as paths and cycles [CGP85], cliques [Orl77; PD81; EFO88], matchings [Orl77; EFO88; GP82], or forests [CV08; CFS14]. De Caen et al. [DC+86] provide bounds for the sum and, respectively, the maximum of the clique covering numbers of a graph and its complement. They also present bounds for the respective clique partitioning numbers that are further improved by Rohatgi, Urschel, and Wellens [RUW21].

Instead of restricting the host class, one can also consider special covers. Pullman, Shank, and Wallis [PSW82], for instance, consider the edge partitioning into maximal cliques. Interestingly, not every graph can be partitioned into maximal cliques, as the example of K_4 with one edge removed shows. The paper thus discusses the existence of maximal-clique partitions and studies the minimum number of templates that such a partition must have. With regard to local covering, Javadi, Maleki, and Omoomi [JMO16] examine the local clique covering number. In particular, they derive lower and upper bounds that depend on the maximum degree and the maximum clique number. In addition, they give asymptotic bounds for the local clique covering number if the host graph is claw-free.

Constructive and improving heuristics for the clique covering problem on general host graphs are presented in [Kel73; BT06; Rod21], a comparison of different heuristics is carried out in [Gra+07]. Gramm et al. [Gra+09] consider the decision variant of the clique covering problem with a fixed input parameter k for the number of allowed templates. Extending the reduction techniques of [Gyá90], they derive a problem kernel, i.e., an instance (G', k') that is a yes-instance iff the original instance (G, k) is a yes-instance. They also propose an exact branching algorithm that is applied to the kernel and empirically performs particularly well on sparse host graphs. The question now is: How small can the kernel size get in relation to k? On the negative side, Cygan, Pilipczuk, and Pilipczuk [CPP16] prove that in the worst case, there is no kernel of subexponential size, assuming the exponential time hypothesis (which states that 3-SAT cannot be solved in subexponential time). On the upside, Friedrich and Hercher [FH15] find that the expected kernel size of random intersection graphs can be much smaller than the worst case suggests.

2.2 Edge Covering with Bicliques

Another template class to consider are bicliques, i.e., complete bipartite graphs. It is easy to see that the edges of K_n can be covered with $n-1$ stars. The same holds for the partition case, but here fewer bicliques do not suffice [GP72; Tve82]. For the covering of K_n, however, we can reduce the number of required bicliques to $\lceil \log_2 n \rceil$ [HHM77; Ber78]. Basic bounds for the general case are provided by Bermond [Ber78]. Chung [Chu80] evaluates the asymptotic behavior of the biclique covering number, while Tuza [Tuz84] provides the best known upper bound at $n - \lfloor \log_2 \frac{2n}{3} \rfloor$. A lower bound of $\frac{\nu^2}{m}$ is due to Jukna and Kulikov [JK09] if the matching number ν of the host graph is known.

The difference between biclique covering and biclique partitioning is explored in [Pin14]. A lower bound for the biclique partition number is proved in [GWS99]. Alon, Bohman, and Huang [ABH17] give a probabilistic upper bound for the partition case.

Just as the clique covering problem, the biclique covering problem is NP-hard, even if the host graph is bipartite [Orl77]. It remains NP-hard if the bipartite graph is chordal [Mül96], but is polynomial solvable for bipartite domino-free graphs [AVJ98].

If the host graph is the complement of a path or a cycle, Gregory, Watts, and Shader [GWS99] give exact values for the biclique partition number. Bezrukov et al. [Bez+08] give a tight bound on the biclique covering number if the host graph is a $K_{n,n}$ with a perfect matching removed.

The local biclique covering number of K_n is $\lceil \log n \rceil$, as shown by Fishburn and Hammer [FH96] for $n \leq 16$ and by Dong and Liu [DL07] for general n. Concerning the local biclique partition number, Erdős and Pyber [EP97] prove an upper bound of $\frac{cn}{\log n}$ with c being a constant that is not computed explicitly. If the host graph is planar, then 4 bicliques always suffice [DL07].

Cornaz and Fonlupt [CF06] formulate an integer program to find a minimum biclique cover, while Watts [Wat06] considers the LP relaxation of this problem, i.e., fractional biclique covers.

An application for the biclique cover number is presented by Günlük [Gün07]. It is shown that this number is an upper bound on the min-cut max-flow ratio of a multi-commodity flow problem.

2.3 Edge Covering with Cycles

It is obvious that a cycle covering of the edges exists if and only if the host graph is 2-connected. For arbitrary host graphs, Erdős, Goodman, and Pósa [EGP66] conjectured that any graph can be covered with $n-1$ cycles and edges. While this bound is clearly tight for trees, it was proven for general graphs by Pyber [Pyb85].

Below we summarize the results for 2-connected host graphs and different variants of the cycle cover problem. These variants are also discussed in the survey article by Jackson [Jac93] and in a book on this topic by Zhang [Zha97].

Bondy [Bon90] conjectured that $\lfloor \frac{2n-1}{3} \rfloor$ cycles always suffice for the covering and showed that this bound is best possible. His conjecture was confirmed by Fan [Fan02]. Therefore, cycles are one of the considered template classes for which the exact value of the covering number is known. There are, however, many works that consider related problems.

A famous variant of the present problem is the *cycle double cover conjecture*. As the name suggests, it asks whether every 2-connected graph has a cycle covering where each

edge of the graph belongs to exactly two cycles. While this conjecture is still open, advances towards it are given in the surveys by Jaeger [Jae85], Chan [Cha09], and Zhang [Zha16].

Another well studied and practically relevant version of the cycle covering problem takes into account the length of the covering cycles. More specifically, the goal of this *minimum cost cycle-cover problem (MCCP)* is to find an edge covering with cycles such that the sum of the length of all cycles is minimal. For the unit weight case, Itai et al. [Ita+81] give a constructive upper bound on the total length. This bound is further improved by Bermond, Jackson, and Jaeger [BJJ83], Alon and Tarsi [AT85], Fraisse [Fra85], Fan [Fan97], and Fan [Fan98]. The best known upper bounds for the cumulated cycle length are $m + n - 1$ and $2n - 2$ [Fan98]. Determining the optimal value, however, is NP-hard [Tho97]. The case of metric edge weights is studied by Immorlica, Mahdian, and Mirrokni [IMM05]. They give an approximation result for a natural greedy covering and show that imposing length bounds on the template cycles makes the problem APX-hard. The MCCP with length constraints is also studied by Bläser and Siebert [BS01] and by Hochbaum and Olinick [HO01]. Finally, let us briefly describe the difference between MCCP and the well-known Chinese Postman Problem (CPP). The CPP is concerned with finding a closed walk of minimum length that contains every edge of a given edge-weighted graph. Its practical importance cannot be described better than in [IMM05]: "Besides its obvious application to mail delivery in China, this problem finds application in a variety of routing problems such as robot navigation and city snow plowing planning". Unlike the MCCP, the CPP is known to be polynomial solvable [EJ73]. Every solution of MCCP can easily be transformed into a solution of CPP. The only difference between the two is that a CPP solution can be the union of cycles and edges, whereas edges are forbidden templates for the MCCP. This small change, however, leads to the complexity increase from P to NP-hardness.

2.4 Edge Covering with Paths

The Gallai conjecture states that any graph can be partitioned into $\lceil \frac{n}{2} \rceil$ edge-disjoint paths. Indeed, one can see that this bound is sharp by considering a graph where every vertex has odd degree. Then, n is even and every vertex must be an endpoint of a covering path, which leads to a minimum of $\frac{n}{2}$ covering paths (cf. [Lov68]). Gallai's conjecture is still open, but progress has been made for special cases, see [Lov68; Don80; Yan98; Fan05; ZL06; BJ17; Gir+21] and references therein.

How about the respective covering problem? Analogously to the partitioning case, Chung [Chu80] conjectured that any graph can be covered with at most $\lceil \frac{n}{2} \rceil$ paths. Pyber [Pyb96] was able to prove a weaker bound, but Fan [Fan02] finally confirmed that $\lceil \frac{n}{2} \rceil$ paths indeed suffice. While the covering conjecture was thereby settled, the partitioning case, i.e. the famous Gallai conjecture, remains open.

2.5 Edge Covering with Connected Subgraphs

Finally, we single out the template class of connected subgraphs. Again, the number of templates to use is given. Existing work is focused on the partition case and on the objective that the template graphs should have similar weight. The weight of a template graph is the sum of all edge weights in this graph. For the unit weight case, Jünger, Reinelt, and Pulleyblank [JRP85] work towards an equipartition of the edges into connected subgraphs. Chu et al. [Chu+10] consider a similar problem for trees as host graphs. The general case

with arbitrary graphs and edge weights is handled in [CWC13]. The authors present a linear time algorithm to find a partition into connected components such that the heaviest template has at most twice as much weight as the lightest template.

In Section 3.2.4, we will derive an independent 2-approximation by transforming the problem into a vertex partition problem and exploiting the claw-freeness of line graphs. Finding a partition into connected subgraphs of similar weight is also the subject of the connected vertex partition problem that is discussed in the next section.

2.6 Edge Covering with other Templates

While Gallai conjectured that any graph can be partitioned into $\lceil \frac{n}{2} \rceil$ paths, Chung [Chu78] showed that it is indeed possible with the same number of **trees**. Earlier, Nash-Williams [N64] had already studied this problem, while Vidyasankar [Vid78] considers the respective covering problem on directed graphs.

More general, Harary, Hsu, and Miller [HHM77] derive a bound for the number of **bipartite graphs** required to cover all edges. In particular, they show that $\lceil \log_2 \chi(G) \rceil$ template graphs always suffice, where $\chi(G)$ is the chromatic number. A remarkable consequence of their result is that any planar graph can be covered with two bipartite graphs.

Motivated by an application for testing printed circuit boards, Bussieck [Bus94] studies the problem of covering the edges of a graph by **cuts**. The objective is to minimize the sum of the cardinalities of all cuts. He proves that finding an optimal cut cover is NP-hard and discusses approximation possibilities. Füredi and Kündgen [FK01] provide general bounds as well as sharp bounds for special host graphs. Kündgen and Spangler [KS05] give an improved bound on the size of the cut cover and describe a connection to cycle covers.

Another variant is the covering with **odd graphs**, i.e., graphs where each non-isolated vertex has odd degree. Due to Mátrai [Mát06], three odd subgraphs suffice to cover any simple graph. The same result also holds for loopless graphs [PŠ19].

Knauer and Ueckerdt [KU16] give results for various template classes on different host graphs. As templates they consider **star forests**, **caterpillar forests**, and **interval graphs**. As host graphs they consider, e.g., planar graphs or graphs with bounded tree-width. They overview existing results for these cases and provide new bounds, not only for the classical objective of minimizing the number of necessary templates but also for slightly different objectives.

Thite [Thi06] considers the covering with a given number of **induced subgraphs** and the objective to minimize the number of vertices in the largest template graph. Finding this minimum is NP-hard, but he proves lower bounds and presents approximation algorithms for certain host graph classes.

2.7 Edge Covering as Vertex Covering

In some cases, we can transform an edge covering problem into a vertex covering problem by considering the line graph. Given an undirected graph $G = (V, E)$, the line graph $L(G)$ consists of the vertex set E and has an edge between $e \neq f \in E$ if and only if e and f are incident in G, i.e., they share a common endpoint in G.

If we consider "taking the line graph" as a function L operating on the set of connected graphs, we might be interested if the inverse function L^{-1} exists. In plain words: Can we reconstruct the original graph from the line graph? Strictly speaking, the answer is no.

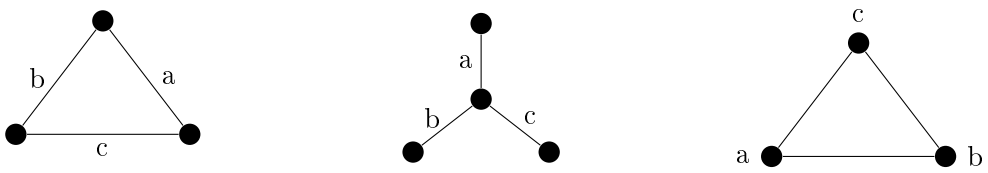


Figure 2.1.: The triangle K_3 and the star $K_{1,3}$ have the same line graph, a K_3.

The function L is not injective as the trivial example of a triangle K_3 and a claw $K_{1,3}$ both mapping to K_3 proves (cf. Figure 2.1). Very surprisingly, however, this is the only exception. Whitney's theorem [Whi32] states that each line graph $H \neq K_3$ has a unique (up to graph isomorphisms) original graph $G = L^{-1}(H)$ with $L(G) = H$. Furthermore, $L^{-1}(H)$ can be reconstructed in linear time (w.r.t. $|V(H)|$) as described in [LTVM15].

Let us point out some simple properties of the function L: First, note that with G being connected, also $L(G)$ is connected. The converse is also true, at least if we ignore any isolated vertices in G. Furthermore, we observe that $L(C_n) = C_n$, $L(P_n) = P_{n-1}$, and $L(K_{1,n}) = K_n$ where C_n and P_n denote the cycle and, respectively, the path with $n \geq 3$ vertices. Since every edge covering of G clearly corresponds to a vertex covering of $L(G)$, and due to Whitney's theorem, we can state the following:

- There is a bijection between edge coverings of G with paths and vertex coverings of $L(G)$ with paths.
- There is a bijection between edge coverings of G with cycles (not containing C_3 as template) and vertex coverings of $L(G)$ with cycles (not containing C_3 as template).
- There is a bijection between edge coverings of G with stars (not containing $K_{1,3}$ as template) and vertex coverings of $L(G)$ with cliques (not containing K_3 as template).
- There is a bijection between edge coverings of G with connected subgraphs (not containing $K_{1,3}$ or K_3 as template) and vertex coverings of $L(G)$ with connected subgraphs (that are line graphs, but not K_3).

In fact, all statements extend to the partitioning case. As this thesis is particularly concerned with connected subgraphs, let us further examine the last statement. The restrictions on the templates are necessary to obtain a bijection. We can, however, also establish a weaker relationship that proves to be more useful. Our goal is to express edge coverings with connected subgraphs as vertex coverings with trees. More specifically, we claim that any solution of one problem can be transformed into an equivalent solution of the other problem, as the following proposition details.

Proposition 2.1 *Let $S_1, \ldots, S_k$ be an edge covering of G with connected subgraphs. Then, there exist trees $T_1, \ldots, T_k$ in $L(G)$ with $V(T_i) = E(S_i)$ for all $i \in [k]$ that constitute a vertex covering of $L(G)$.*

Conversely, let $T_1, \ldots, T_k$ be a vertex covering of $L(G)$ with trees. Then, there exist connected subgraphs $S_1, \ldots, S_k$ in G with $E(S_i) = V(T_i)$ for all $i \in [k]$ that constitute an edge covering of G.

Proof. For the first statement, we note that choosing any spanning tree T_i of $L(S_i)$ for $i \in [k]$ satisfies the requirements. For the second statement, we simply choose $S_i = L^{-1}(T_i)$ (which is well defined since K_3 is not a tree) and recall that with T_i also $L^{-1}(T_i)$ is connected. □

We use this relationship in Chapter 6 when we formulate an edge covering problem on a road network as a vertex covering problem with trees.

3
Balanced Vertex Covering

The first vertex covering problem that we are going to detail concerns the covering with balanced templates. The goal of the balancing is to use templates of similar weight, and there are several important applications for this problem. While the task of political districting clearly requires a partition [BEL03; GW18], other applications such as terrain coverage with multiple robots admit overlapping solutions. The latter includes significant future tasks such as cleaning, harvesting, and security patrolling jobs performed by a number of coordinated robots [Zhe+05; YJC13]. In both applications it is desired that the resulting subregions have a similar weight, i.e., the number of voters or the time necessary for job completion.

There are different popular objectives for balanced covers. Among the most prominent variants are the max-min and min-max objectives. Given a weighted graph and a number of templates, the goal is to maximize the total weight of the minimum part or, respectively, to minimize the weight of the maximum part. The bounded covering problem, on the other hand, aims to minimize the number of templates given a maximum template weight.

In this chapter, we give an overview on different variants of balanced vertex covering, or rather, as we will see, balanced vertex partitioning. Indeed, the literature is almost exclusively concerned with the partitioning case. In Section 3.1, we consider different main variants of the problem and review corresponding results. A majority of the works is dedicated to finding approximation algorithms or inapproximability results. In line with this, we present an approximation algorithm for the most studied variant in Section 3.2. The presented approximation runs in linear time and is valid for both main objectives: min-max and max-min.

3.1 Literature Overview

Without constraints on the template graphs, a balanced partition problem becomes a multi-way number partitioning problem, which is a generalization of the classical NP-hard partition problem, see e.g. [Kor09]. Therefore, the minimum requirement that is generally set on template graphs is connectivity. We divide the literature review for balanced covers into three parts. First, we consider the basic problem with connected templates. Then, additional constraints on the template sizes are imposed. And finally, we conider several other classes of connected templates, namely trees, paths, stars, and cycles.

3.1.1 Connected Vertex Partition

For a vertex weighted graph and an input parameter $k \in \mathbb{N}$, the connected vertex partitioning problem (CVP) asks for a partition of the vertices into k connected components. If k is fixed, the problem is denoted by CVP_k. The CVP is usually considered with max-min

or min-max objective and also known as the *balanced connected partition problem*.

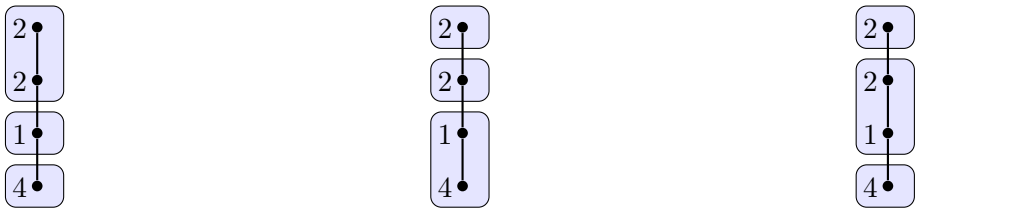
The min-max and max-min version of CVP are closely related. In fact, for CVP_2 the two versions obviously coincide. For larger k though, optimal solutions of the two problems *can* differ. A simple example of this is presented in Figure 3.1. While for this instance one can find a solution that simultaneously optimizes the min-max and max-min objective, some instances do not admit such a solution (cf. [LPS93]).

Concerning the complexity, already CVP_2 is NP-hard [CGM83], even on unit-weighted bipartite graphs as shown by Dyer and Frieze [DF85]. Moreover, it is also hard to approximate. Chataigner, Salgado, and Wakabayashi [CSW07] prove that CVP_2 does not admit a polynomial-time approximation scheme and that there is no α-approximation for CVP with $\alpha < \frac{6}{5}$, unless P = NP. Earlier, Chlebíková [Chl96] had given a $\frac{4}{3}$-approximation for CVP_2, which was later shown to actually be a $\frac{5}{4}$-approximation [Che+20]. Concerning an approximation for the general CVP, the first result was a Δ-approximation, where Δ is the maximum degree of a spanning tree of the host graph [BES19]. We will detail this algorithm in Section 3.2.2. This result is improved by Casel et al. [Cas+21] who derive a 3-approximation for both versions of the CVP.

Focusing on the max-min CVP, more approximation results are known. For CVP_3 and CVP_4 the best known result is a 2-approximation for 3-connected or, respectively, 4-connected graphs [CSW07]. For the CVP_3, this was improved by Chen et al. [Che+20] to a $\frac{5}{3}$-approximation which also applies to general graphs.

The max-min CVP for special host graphs allows for even better results. For instance, CVP_k is polynomially solvable for trees. Note that on a tree, any partition into k connected parts is uniquely determined by choosing $k-1$ cut edges. This observation was used to give polynomial algorithms for the max-min CVP_k [PS81] and for the min-max CVP_k [BPS80]. Earlier, Kundu and Misra [KM77] had considered a closely related problem and provided an algorithm that can be employed to solve the min-max CVP_k in polynomial time. In 1991, Frederickson [Fre91] gave linear-time algorithms for both problems. These are particularly important, since different heuristics transform the original instance onto a tree to efficiently solve the problem there [CWC13; Zho+19].

Another host graph class for which CVP was investigated are grid graphs. While it was shown that CVP_k is NP-hard for arbitrary grid graphs [Bec+98], the max-min CVP_k can be solved in polynomial time for ladders, i.e., grid graphs with two rows and an arbitrary number of columns [Bec+01].



(a) optimal solution for min-max CVP_3 that is not optimal for max-min CVP_3

(b) optimal solution for max-min CVP_3 that is not optimal for min-max CVP_3

(c) optimal solution for min-max CVP_3 and max-min CVP_3

Figure 3.1.: Comparison of solutions for min-max CVP_k and max-min CVP_k for an exemplary instance. The numbers indicate vertex weights.

For the class of series-parallel graphs, CVP_k is also NP-hard. Ito, Zhou, and Nishizeki [IZN06] give a pseudo-polynomial time algorithm for both, the min-max and max-min CVP. They also show that their algorithm can be extended to graphs with bounded tree-width.

Wu [Wu12] gives a fully polynomial time approximation scheme for sufficiently connected (i.e., k-connected for CVP_k) interval graphs.

Finally, there are approaches to solve the max-min CVP with integer linear programs. Matić [Mat14] presents a first formulation and introduces a variable neighborhood search to facilitate the optimization procedure. The formulations of Miyazawa et al. [Miy+20] make use of vertex separators or flows, respectively, to model the connectivity. For the separator version the respective polyhedron is studied for the case of unit-weighted vertices. The same authors extend their work in [Miy+21] introducing valid inequalities with a corresponding separation routine.

The min-max CVP has received less attention. As mentioned above, Becker, Perl, and Schach [BPS80] give a polynomial algorithm for the min-max CVP_k if the host graph is a tree. Returning to general host graphs, the min-max CVP_3 can be approximated within a ratio of $\frac{3}{2}$ [Che+20]. An approximation for the unit-weighted min-max CVP_k for $k \geq 3$ was presented by Chen et al. [Che+21]. They propose a neighborhood search that results in a $\frac{k}{2}$-approximation. Moura, Ota, and Wakabayashi [MOW22] achieve a pseudo-polynomial $\frac{k}{2}$-approximation also for general vertex weights.

A mixed integer linear programming approach using a flow formulation to ensure the connectivity of the subgraphs is due to Zhou et al. [Zho+19]. Since this approach is only feasible for small instances, they also propose a genetic algorithm as a heuristic approach. The main idea is to find a proper spanning tree whose optimal partition will also be optimal for the given graph.

3.1.2 (l, u)-Partition

As an alternative to the min-max or max-min objective, we can impose additional constraints on the size of each template to model the balancing. Given a graph with non-negative vertex weights and numbers $0 \leq l \leq u$, a partition into connected components, each of which has a cumulated weight in the range $[l, u]$, is called an (l, u)-partition. Typical objectives are to maximize or to minimize the number of components of the (l, u)-partition. There is also a feasibility variant that asks if an (l, u)-partition into a given number k of components exists. Applications occur in political districting where each district should have a similar number of residents, but also in paging systems of operation systems and in image processing [Cun19].

Lucertini, Perl, and Simeone [LPS93] consider this problem when the host graph is a path. Exploiting the simple graph structure, they give a preprocessing routine and a greedy algorithm to solve the three mentioned variants of (l, u)-partition in linear time.

The problems are still polynomially solvable on trees [Ito+12]. The authors solve it with dynamic programming while preventing the corresponding table from expanding unreasonably.

For general host graphs the problem is studied by Ito, Zhou, and Nishizeki [IZN06]. They show that all variants are NP-hard, even for series-parallel graphs. For the class of series-parallel graphs, however, a pseudo-polynomial algorithm whose runtime depends on the upper bound u is given.

Cunha [Cun19] tackles the (l, u)-partition of minimum cardinality with integer programming. The problem is solved with a branch-and-cut algorithm that makes use of lifted subtour elimination constraints and other valid inequalities that prevent too small templates.

3.1.3 Special Template Classes

Here, we will discuss several vertex covering problems with different template classes that are related to the *vehicle routing problem* (VRP). The VRP generalizes the traveling salesman problem (TSP) and belongs to the most important and studied problems in combinatorial optimization. It asks for an optimal set of routes for a number of vehicles, starting and ending at one or multiple depots, to visit a given set of customers. There is a number of popular variants for the VRP, e.g., with inclusion of time windows or with pickup and delivery constraints. The capacitated VRP imposes an upper bound on the length of each route. For details on VRP variants and a review of solution techniques see [Lap92; TV02; BRVN16].

In contrast to the CVP and to (l, u)-partitions, we consider edge weighted host graphs. The weights are usually assumed to be metric and in some cases one or several root vertices for the templates are specified. A rooted covering problem requires that every template contains the single root vertex or, in case of multiple given roots, that every template includes a distinct root vertex.

The paper by Farbstein and Levin [FL15] provides contributions to all problems that we consider next. The time complexity of their approximation algorithms, however, may be exponential in k and also depends on an ε that defines the quality of the approximation. By fixing these parameters, the problem can be solved in polynomial time, and hence, it is called fixed parameter tractable (FPT). For brevity, we call the corresponding approximation algorithm an FPT-approximation.

The first covering that we consider uses **trees** as templates. Even et al. [Eve+04] show that the min-max tree covering is NP-hard and derive a 4-approximation which is independently achieved by Arkin, Hassin, and Levin [AHL06]. Khani and Salavatipour [KS14] improve the approximation ratio to 3, and Farbstein and Levin [FL15] give an FPT-approximation with ratio $(2 + \varepsilon - \frac{2}{k+1})$. There still remains a gap, as Xu and Wen [XW10] prove that a 1.5-approximation is the best we can hope for. The problem has also gained notable attention if the host graph is a tree. As seen above, min-max partitioning a tree into k subtrees is polynomially solvable, whereas the respective covering problem is NP-hard already for $k = 2$ [AB96]. Polynomial approximations for this case are possible with a ratio below 2: Averbakh and Berman [AB97] give a $(2 - \frac{2}{k+1})$-approximation, Nagamochi and Okada [NO04] achieve the same result but they reduce the time complexity to $\mathcal{O}(k^2 n)$. The rooted version of this problem can be approximated within a factor of $2 + \varepsilon$ for any $\varepsilon > 0$ [NO03], and for the case $k = 2$ even within a factor of $\frac{4}{3}$ [AB96]. Rooted cases for general host graphs are discussed in [Eve+04], [Nag05], and [FL15]: Parallel to the unrooted case, Even et al. [Eve+04] give a 4-approximation for the rooted version with multiple roots. With an FPT-approximation, the factor can be improved to of $3 + \varepsilon - \frac{2}{k+1}$ [FL15]. Again, no approximation within a factor of 1.5 is possible, unless P=NP [XW10]. For the single-rooted case, Nagamochi [Nag05] provides a $(3 - \frac{2}{k+1})$-approximation, while here, the current inapproximability bound lies at $\frac{10}{9}$ [XW10]. An approximation for the min-max k tree cover with cardinality constraints on each template is due to Das, Jain, and Kumar [DJK20]. The bounded tree cover problem is NP-hard [AHL06] and there are

approximation algorithms with factors of 3 [AHL06] and 2.5 [KS14], respectively. A linear programming formulation for this problem is studied in [YLB20].

A special tree covering is the covering with **paths**. This problem was the subject of the *Whizzkids '96* competition to determine optimal paths for newspaper delivery from a single depot in a specific instance ($n = 120, k = 4$). Applegate et al. [App+02] tackle this particular problem with a branch-and-bound approach. The general problem is studied in depth by Arkin, Hassin, and Levin [AHL06], establishing NP-hardness and providing a 4-approximation, and by Xu and Wen [XW10] who prove that no approximation within a factor of 1.5 is possible, unless P=NP. Farbstein and Levin [FL15] propose an FPT-approximation with ratio $2 + \varepsilon$ for this problem. A linear programming formulation for the path covering problem is again given in [YLB20].

Finally, we have the covering with **cycles** which is closest to the VRP. Most papers explicitly or implicitly assume the host graph to be complete. In this case any tree covering α-approximation can easily be transformed into a cycle covering 2α-approximation. Exploiting the metric edge weights, the transformation mimics the classic TSP approximation with doubling the edges of the tree and using shortcuts to avoid repeated vertices. For instance, this provides a 6-approximation from the tree covering approximation in [KS14]. Improved approximations for min-max cycle covering are due to Jorati [Jor13] and Xu, Liang, and Lin [XLL13] with factor $\frac{16}{3} + \varepsilon$. An FPT-approximation achieves a ratio of $4 + \varepsilon$ [FL15], while the best known inapproximability bound is again 1.5 [YL19]. Yu, Liu, and Bao [YLB19] consider the bounded cycle cover and provide approximation algorithms. In [YLB20] the same authors derive lower bounds with relaxed LP formulations for this problem. Traub and Tröbst [TT21] consider a version of the problem where the number of cycles as well as the maximum weight of the cycles is part of the objective function. Rooted versions of the min-max cycle covering problem date back to Frederickson, Hecht, and Kim [FHK76] who studied this as a min-max k-TSP problem deriving a first approximation. A number of other papers also deal with approximation results for the rooted case [Nag05; XXZ12; Jor13; XLL13; FL15; YL19; LL20; LZ21]. The best known results are a 7-approximation for the single-rooted case [XLL13], and a $(5 + \varepsilon)$-approximation for the multi-rooted version [LL20], as well as an inapproximability bound of 1.5 [YL19].

3.2 An Approximation for Connected Vertex Partition

The literature overview revealed that the connected vertex partition problem is the most studied variant of balanced vertex covering problems. We have seen that a number of approximation results are derived for the min-max and max-min version of the CVP. In this section, we present a Δ-approximation for both versions of the CVP, where Δ is the maximum degree of a spanning tree of the host graph. This means that if W is the objective value of our algorithm and W^* is the optimal objective value, we guarantee that $W \leq \Delta W^*$ for the min-max CVP, and $W \geq \frac{1}{\Delta} W^*$ for the max-min CVP.

When the approach was proposed in [BES19], it provided the first approximation result for the CVP on general graphs. Earlier works did only provide approximations for fixed k or for certain host graph classes. Motivated by the presented approach, the approximation of the general CVP gained further interest, and meanwhile, a 3-approximation has been developed by Casel et al. [Cas+21]. In many cases, the factor of 3 is better than the Δ-approximation discussed here. However, the approach is also much more technical and the presented Δ-approximation is not only easier to implement, but also much faster: If X^*

is the optimal objective value and $w_{\max}$ the maximum node weight, the 3-approximation of the min-max CVP in [Cas+21] runs in $\mathcal{O}(\log(X^*)|V||E|(\log\log X^* \log(|V|w_{\max}) + k^2))$, whereas our algorithm has an $\mathcal{O}(|E|)$ runtime. The difference for the max-min CVP is comparable.

The remainder of this section is structured as follows. First, we propose a general approximation scheme that is subsequently used for our approximation results. In Section 3.2.2, we employ the scheme to derive a Δ-approximation for CVP. Then, we present a refined approach for the max-min CVP that also yields a Δ-approximation (but without a previously imposed condition). Finally, we develop a variation of the approach that is specialized for line graphs, achieving a 2-approximation for the min-max and max-min edge partitioning problem. We start, however, by formally stating the problem at hand.

Problem Formulation: Given a connected and undirected graph $G = (V, E)$ with positive vertex weights $w \in \mathbb{Q}^V_{>0}$ and an integer $k \in \mathbb{N}$, find vertex-disjoint connected subgraphs $S_1, \dots, S_k$ with $V = \bigcup_{i\in[k]} S_i$. In the case of min-max CVP, our goal is to minimize $\max_{i\in[k]} w(S_i)$, while for the max-min CVP, we maximize $\min_{i\in[k]} w(S_i)$.

Let us also introduce some additional notation for the remainder of this section. We denote by $w_{\max}$ the largest weight of all vertices, i.e., $w_{\max} := \max_{v\in V} w_v$. Furthermore, we denote the degree of vertex v in G by $\deg_G(v)$ and the maximum degree of G by $\Delta(G)$.

3.2.1 A General Approximation Scheme for CVP

We propose an approximation scheme that allows us to handle different approximation problems simultaneously. More specifically, we employ the scheme to obtain approximations for the min-max and max-min CVP, and also for the respective versions in edge partitioning.

To this end, we generalize the already mentioned (l, u)-partition. Given an interval $\mathcal{I}$, an $\mathcal{I}$-partition is a partition $\mathcal{S} = \{S_1, \dots, S_m\}$ of the vertices such that $w(S_i) \in \mathcal{I}$ for all $i \in [m]$. Furthermore, if $w(S_i) \in [l, u)$ for all $i \in [m-1]$ and $w(S_m) < u$, we call $\mathcal{S}$ a quasi-$[l, u)$-partition.

We will see that for certain choices of λ, a quasi-$[\lambda, c\lambda)$-partition leads to a c-approximation for the CVP. The following theorem gives the details.

Theorem 3.1 *Let (G, w, k) be an instance of CVP and let $T_1, \dots, T_m$ be connected and form a quasi-$[\lambda, c\lambda)$-partition of G.*

a) If $\lambda := \max\{w_{\max}, \frac{w(G)}{k}\}$, then $T_1, \dots, T_m$ is a c-approximation for the min-max CVP on (G, w, k).

b) If $\lambda := \frac{w(G)}{ck}$, then $T_1, \dots, T_m$ leads to a c-approximation for the max-min CVP on (G, w, k).

Proof.

a) Let $T_1^*, \dots, T_k^*$ be an optimal partition for the min-max CVP on (G, w, k) and let $W^* := \max_{i\in[k]} w(T_i^*)$. Then, $w(G) = \sum_{i\in[k]} w(T_i^*) \leq kW^*$ and, hence, $\lambda \leq W^*$ because $w_{\max} \leq W^*$ is obvious.

Note that $w(T_m) > 0$, and therefore

$$w(G) \;=\; w(T_m) + \sum_{i=1}^{m-1} w(T_i) \;\geq\; w(T_m) + (m-1)\lambda \;>\; \frac{m-1}{k} w(G),$$

which implies that $m - 1 < k$ or, equivalently, that $m \leq k$. As a result $T_1, \ldots, T_m$ is a feasible k-partition and $w(T_i) < c\lambda \leq cW^*$ for any $i \in [m]$, which completes the proof of the first statement.

b) Now let $T_1^*, \ldots, T_k^*$ be an optimal partition for the max-min CVP on (G, w, k) and let $W^* := \min_{i \in [k]} w(T_i^*)$. Then, $w(G) = \sum_{i \in [k]} w(T_i^*) \geq kW^*$ and, consequently, $\lambda \geq \frac{1}{c} W^*$.

Since $w(T_i) < c\lambda$ for every $i \in [m]$, we know that $w(G) = \sum_{i \in [m]} w(T_i) < mc\lambda$ and, hence, $k = \frac{w(G)}{c\lambda} < m$.

We can easily relabel the trees $T_1, \ldots, T_{m-1}$ such that $\bigcup_{j \in \{k, \ldots, m\}} T_j$ is connected. Afterwards, we define

$$S_i := \begin{cases} T_i & , i \in [k-1], \\ \bigcup_{j \in \{k, \ldots, m\}} T_j & , i = k. \end{cases}$$

From $w(T_i) \geq \lambda \; \forall i \in [m-1]$ we obtain $w(S_i) \geq \lambda \geq \frac{1}{c} W^*$ for arbitrary $i \in [k]$ and hence $S_1, \ldots, S_k$ is the sought approximation. □

3.2.2 Δ-Approximation for CVP

The min-max CVP has a natural tendency towards many templates. In the ideal case, every vertex is a single template and the objective value is $w_{\max}$. In some sense, our goal is therefore to minimize the number of templates while keeping the weight of every template below a given value. The max-min CVP, on the other hand, tends towards few templates. Without any restrictions on k, the optimal solution would be to take the full host graph as single template with objective value $w(G)$. From this point of view, our goal is to maximize the number of templates subject to a certain minimum weight of each template.

It is all the more surprising that we apply the same algorithm to achieve an approximation for both versions of the problem. The approximation for the CVP instance (G, w, k) is performed in two steps: First, we find a spanning tree T of G and specify a root node r. Then, we successively remove subtrees of a given minimum weight from the spanning tree, ensuring that the remaining tree is still connected.

While we discuss the choice of the spanning tree later, let us focus on the problem of partitioning a tree into connected parts of similar weight. Lukes [Luk74] was the first to consider partition problems on trees, but did not demand connectivity of the parts. Kundu and Misra [KM77] give an algorithm to partition a tree with node weights into subtrees with an upper weight bound. Since then, various other approaches have been proposed, e.g., in [BPS80; Fre91; ABP93; Ito+12], but none of these were employed to construct approximation algorithms for the corresponding problem on general graphs.

In order to simplify the description of our algorithm, we start by introducing some notation concerning rooted trees. Let T^r be the tree rooted at node r. With $N^+(v)$ we denote the set of child nodes of the node v and $N^-(v)$ is the (unique) parent node of $v \neq r$ in T^r. For $v \in T^r$, the graph T_v^r is the subtree of T^r rooted at v. For instance, in Figure 3.2 we have $T_2 = T_e^a$. Note that both T_v^r and $T^r - T_v^r$ are trees and that their respective node

Algorithm 3.1: `BalancedPartition`(G, w, λ)

Input: graph $G = (V, E)$, $w \in \mathbb{Q}^V_{>0}$, $\lambda \geq w_{\max}$
Output: trees $T_1, \dots, T_m \subseteq T^r$ with $V = \dot{\bigcup}_{i \in [m]} V(T_i)$

1 $T^r \leftarrow$ spanning tree of G rooted at node r
2 $\mathcal{T} \leftarrow \varnothing$; $\quad w^r_v \leftarrow 0 \quad \forall v \in V$
3 $Q \leftarrow$ list of vertices of T^r in reversed BFS order
4 **for** $v \in Q$ **do**
5 **if** $v = r$ **then**
6 **return** $\mathcal{T} \cup T^r$
7 $w^r_v \leftarrow w_v + \sum_{x \in N^+(v)} w^r_x$
8 **if** $w^r_v \geq \lambda$ **then**
9 $\mathcal{T} \leftarrow \mathcal{T} \cup T^r_v$
10 $T^r \leftarrow T^r - T^r_v$

sets are disjoint, i.e., the edge between $v \neq r$ and its parent does neither belong to T^r_v nor to $T^r - T^r_v$.

The tree partitioning routine is described in Algorithm 3.1 and Figure 3.2 illustrates how it works. We successively split off rooted subtrees of minimum weight λ from bottom to top, while manipulating the original tree T^r (cf. line 10). After considering all other nodes, we process the root node and set T_m to be the remaining tree T^r.

With the preceding observations, the correctness of Algorithm 3.1 is clear. Let us now briefly analyze its computational complexity. A spanning tree of G can be found in $\mathcal{O}(|E|)$, the breath-first search then runs in $\mathcal{O}(|V|)$ and afterwards, we process every node in amortized constant time. Thus, our algorithm has a runtime linear in $|E|$.

The following result is the cornerstone for the Δ-approximation.

Lemma 3.2 *Let (G, w, k) be an instance of CVP, let $\lambda \geq w_{\max}$, and let $T_1, \dots, T_m$ be the output of `BalancedPartition`(G, w, λ). Furthermore, assume that the constructed spanning tree is rooted at node r with $\deg_T(r) < \Delta(T) =: \Delta$.*

Then, $T_1, \dots, T_m$ is a quasi-$[\lambda, \Delta\lambda)$-partition.

Proof. For some $i \in [m]$ consider the root v of T_i and observe that $w^r_{x_j} < \lambda$ for all child nodes $x_1, \dots, x_\ell$ of v in T_i. Due to the choice of r we have $\ell \leq \Delta - 1$ and hence

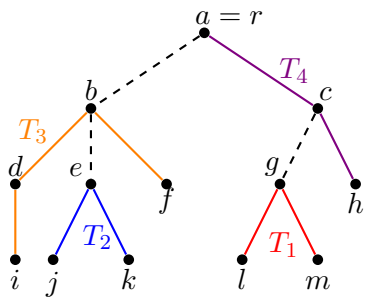


Figure 3.2.: Balanced tree decomposition for an exemplary tree rooted at node a with unit weights and $\lambda = 3$.

$$w(T_i) \;=\; w_v^r \;=\; w_v + \sum_{j=1}^{\ell} w_{x_j}^r \;<\; w_{\max} + \ell\lambda \;\leq\; \Delta\lambda.$$

The fact that $w(T_i) \geq \lambda \;\; \forall i \in [m-1]$ comes from line 8 of the algorithm. □

Now we have everything together to state the main result:

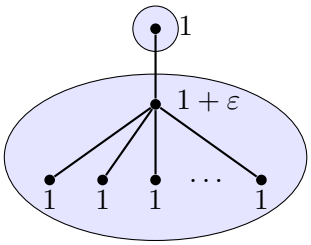
Theorem 3.3 *Let (G, w, k) be an instance of CVP, and let Δ be the maximum degree of a spanning tree of G. Then, a Δ-approximation for the min-max CVP can be computed in $\mathcal{O}(|E|)$. Furthermore, if $\frac{w(G)}{\Delta k} \geq w_{\max}$, a Δ-approximation for the max-min CVP can be computed in $\mathcal{O}(|E|)$.*

Proof. Concerning the min-max CVP, the choice of $\lambda := \max\{w_{\max}, \frac{w(G)}{k}\}$ is a valid input for `BalancedPartition`, and by choosing a spanning tree of maximum degree Δ within the algorithm, Lemma 3.2 together with Theorem 3.1 proves the statement.

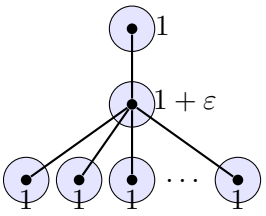
For the max-min CVP, the same theorem suggests to set $\lambda := \frac{w(G)}{\Delta k}$. However, since Lemma 3.2 requires $\lambda \geq w_{\max}$, we can apply the same technique as above but demand that $\frac{w(G)}{\Delta k} \geq w_{\max}$. □

In the next section, we refine the approach for the max-min CVP and guarantee a Δ-approximation also for the case $\frac{w(G)}{\Delta k} < w_{\max}$.

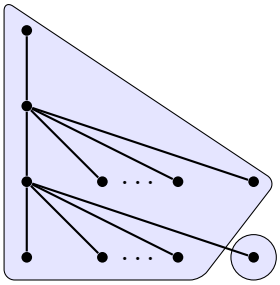
For both versions of the CVP, one can also find examples for which the algorithm attains an approximation ratio of Δ. While the author of this thesis wonders how many people



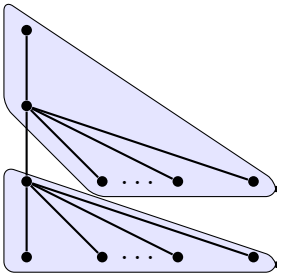
(a) Output of `BalancedPartition` for CVP instance $(G, w, |V|)$.



(b) Optimal solution for min-max CVP instance $(G, w, |V|)$.



(c) Output of `BalancedPartition` for CVP instance $(G, 1, 2)$.



(d) Optimal solution for max-min CVP instance $(G, 1, 2)$.

Figure 3.3.: Exemplary instances of CVP for which `BalancedPartition` attains an approximation ratio of Δ.

actually read this sentence, let us start with the min-max CVP: We consider a star graph where the center has a weight of $1+\varepsilon$ and every other node has unit weight. Consequently, we set $\lambda = w_{\max} = 1+\varepsilon$ and `BalancedPartition` yields the solution depicted in Figure 3.3a with objective value $\Delta + \varepsilon$. In Figure 3.3b, the optimal solution with objective value $1+\varepsilon$ is depicted. Note however, that the example exploits that $m \ll k$ for the output of the algorithm, i.e., we only obtain two trees instead of $|V|$ trees. It remains open if a second stage optimization where we iteratively partition the heaviest tree can further improve the theoretic bound.

An approximation ratio of Δ can also be attained for the max-min CVP, as the example in Figure 3.3c and 3.3d shows. The depicted graph has unit weights and we consider $k = 2$, leading to $\lambda = \frac{2\Delta}{2\Delta} = 1$ and the output of `BalancedPartition` in Figure 3.3c with objective value 1. The optimal solution, shown in Figure 3.3d, on the other hand, has an objective value of Δ. This example can also easily be extended for arbitrary k.

To end this section, we will briefly discuss the choice of the spanning tree. To obtain a best possible approximation result, we are interested in finding a spanning tree whose maximum degree is as small as possible. This problem is known as *minimum degree spanning tree problem*. While the problem is NP-hard, Fürer and Raghavachari [FR92] give a polynomial time algorithm to find a spanning tree of degree at most $\Delta^* + 1$, where Δ^* is the maximum degree of an optimal spanning tree. For special classes of graphs, better results can be achieved. For instance, we will exploit later that every line graph admits a spanning tree of degree at most 3 (cf. Lemma 3.7).

3.2.3 A Refinement for max-min CVP

The Δ-approximation for the max-min CVP from Theorem 3.3 requires $\frac{w(G)}{\Delta k} \geq w_{\max}$. In the following, we will derive an alternative approach to drop this condition.

Let us first examine why the condition was imposed, and why the precondition $\lambda \geq w_{\max}$ in `BalancedPartition` is set. It is necessary in order to apply Lemma 3.2, and in this lemma it is only needed to prove that each template has weight less than $\Delta\lambda$. This, in turn, is used for the max-min variant in Theorem 3.1 to show $m > k$, and that we can hence merge templates to obtain a partition into k parts.

Here, we adjust this approach, and use `BalancedPartition` only for certain parts of the graph. Adhering to the interpretation of maximizing the number of templates subject to a lower weight bound λ on each template, we first identify the *heavy* vertices that can form such a template on their own, i.e., we define

$$H := \{v \in V : w_v \geq \lambda\}.$$

The algorithm `MaxMinPartition` starts by removing these heavy vertices from G, such that we are left with a number of connected components which we categorize based on their cumulated weight. Components with cumulated weight less than λ are aggregated in the set $\mathcal{G}^<$, the other components form the set $\mathcal{G}^\geq$. Note that by design, each component in $\mathcal{G}^<$ is adjacent to a vertex $h \in H$. Therefore, we form templates $S^h = \{h\}$ for every $h \in H$ and add each component in $\mathcal{G}^<$ to the template of an adjacent heavy node $h \in H$. Consequently, we are only left with the components in $\mathcal{G}^\geq$, and these are partitioned using Algorithm 3.1. More specifically, we observe that the maximum vertex weight of any component $G^i \in \mathcal{G}^\geq$ is smaller than λ (since $G^i \cap H = \varnothing$), and that the preconditions for the algorithm are hence satisfied. `BalancedPartition`(G^i, w, λ) now returns a partitioning $S^i_1, \ldots, S^i_{k_i}$ of G^i, and

Algorithm 3.2: `MaxMinPartition`(G, w, λ)

Input: graph $G = (V, E)$, $w \in \mathbb{Q}_{>0}^V$, $\lambda \geq 0$
Output: connected subgraphs $S_1, \ldots, S_m \subseteq G$ with $V = \dot{\bigcup}_{i \in [m]} V(S_i)$, approximation bound Δ

1 $H \leftarrow \{v \in V : w_v \geq \lambda\}$
2 $\mathcal{G}^< \leftarrow$ connected components C of $G - H$ with $w(C) < \lambda$
3 $\mathcal{G}^\geq \leftarrow$ connected components C of $G - H$ with $w(C) \geq \lambda$
4 **for** $h \in H$ **do**
5 $S^h \leftarrow (\{h\}, \varnothing)$ // The graph consisting of node h
6 **for** $C \in \mathcal{G}^<$ **do**
7 $h \leftarrow$ any vertex in H adjacent to C
8 $S^h \leftarrow S^h + C$
9 **for** $G^i \in \mathcal{G}^\geq$ **do**
10 $S_1^i, \ldots, S_m^i \leftarrow$ `BalancedPartition`(G^i, w, λ)
11 $\Delta_i \leftarrow$ maximum degree of the spanning tree used in `BalancedPartition`
12 **if** $w(S_m^i) \geq \lambda$ **then**
13 $\mathcal{S}^i \leftarrow \{S_1^i, \ldots, S_{m-1}^i, S_m^i\}$
14 **else**
15 $\mathcal{S}^i \leftarrow \{S_1^i, \ldots, S_{m-2}^i, S_{m-1}^i + S_m^i\}$
16 **return** $\bigcup_{h \in H} S^h \cup \bigcup_{G^i \in \mathcal{G}^\geq} \mathcal{S}^i$, $\max_{G^i \in \mathcal{G}^\geq} \Delta_i$

iff $w(S_{k_i}^i) < \lambda$, we merge $S_{k_i}^i$ and $S_{k_i-1}^i$ to form a connected subgraph of weight at least λ. The resulting partitioning of G^i is denoted by $\mathcal{S}^i$, and we define $\mathcal{S} = \bigcup_{h \in H} S^h \cup \bigcup_{G^i \in \mathcal{G}^\geq} \mathcal{S}^i$.

Now there are two possibilities: If $|\mathcal{S}| < k$, we did not end up with a sufficient number of templates and, therefore, have to decrease the weight bound λ and iterate. If $|\mathcal{S}| \geq k$, however, we merge connected sets arbitrarily in $\mathcal{S}$ until $|\mathcal{S}| = k$. A binary search for the largest λ in the interval $[0, {}^{w(G)}/_k]$ for which `MaxMinPartition` returns a set $\mathcal{S}$ with at least k elements, leads to a Δ-approximation for max-min CVP.

Lemma 3.4 *The output $\mathcal{S}$ of `MaxMinPartition`(G, w, λ) is a $[\lambda, \infty)$-CVP for G.*

Proof. From the construction it is clear, that each vertex belongs to exactly one template, i.e., we have a vertex partition. Furthermore, the graphs S^h for $h \in H$ are connected (cf. line 7) and $w(S^h) \geq w_h \geq \lambda$. The graphs in $\mathcal{S}^i$ for $G^i \in \mathcal{G}^\geq$ are also connected since they are the output of `BalancedPartition`. If in line 15, the last two graphs are combined, the resulting graph is still connected, due to the design of `BalancedPartition`. This operation also ensures that $w(S^i) \geq \lambda$ for all $S^i \in \mathcal{S}^i$. □

Lemma 3.5 *Let (G, w, k) be an instance of max-min CVP with optimal objective value W^*. If `MaxMinPartition`(G, w, λ) yields output $(\mathcal{S}, \Delta)$ with $|\mathcal{S}| < k$, then $\Delta\lambda > W^*$.*

Proof. For the proof we assume that $\Delta\lambda \leq W^*$, and we will show that with such a λ, the output $\mathcal{S}$ of `MaxMinPartition` has at least k templates.

To this end, we consider the sets H, $\mathcal{G}^<$, and $\mathcal{G}^\geq = \{G^1, \ldots, G^\ell\}$ that are constructed

within `MaxMinPartition`, and an optimal solution $\mathcal{S}^* = \{S_1^*, \ldots, S_k^*\}$ of the instance (G, w, k). For $G^i \in \mathcal{G}^{\geq}$, we define $\mathcal{S}_i^* := \{S \in \mathcal{S}^* : S \subseteq G^i\}$. All other optimal templates form the set $\mathcal{S}_H^*$, i.e., $\mathcal{S}_H^* := \mathcal{S}^* \setminus \bigcup_{i=1}^{\ell} \mathcal{S}_i^*$, and we note that these sets form a partition of $\mathcal{S}^*$.

Observe that for each $S \in \mathcal{S}_H^*$, we have $S \cap H \neq \varnothing$. To see this, recall that H separates all $G^i \in \mathcal{G}^{\geq}$ and all $C \in \mathcal{G}^{<}$ from each other. Therefore, if for some $S \in \mathcal{S}^*$, we know that $S \cap G^i \neq \varnothing$ but $S \not\subseteq G^i$ for some $G^i \in \mathcal{G}^{\geq}$, then $S \cap H \neq \varnothing$ is implied. Analogously, since $w(S) \geq \lambda$, if $S \cap C \neq \varnothing$ for some $C \in \mathcal{G}^{<}$, then $S \cap H \neq \varnothing$ is also necessary. Since each $h \in H$ can only be part of one template, we can derive that $|\mathcal{S}_H^*| \leq |H|$.

Next, we show that $|\mathcal{S}_i^*| \leq |\mathcal{S}^i|$ for $i \in [\ell]$. To this end, we consider a fixed $G^i \in \mathcal{G}^{\geq}$ and the output $S_1^i, \ldots, S_m^i$ of `BalancedPartition`(G^i, w, λ). Now we have

$$|\mathcal{S}_i^*|\, W^* \;\leq\; \sum_{S^* \in \mathcal{S}_i^*} w(S^*) \;\leq\; \sum_{j=1}^{m} w(S_j^i) \;<\; m\,\Delta\lambda. \tag{3.1}$$

The first inequality holds because $W^* = \min_{S \in \mathcal{S}^*} w(S)$. The second inequality holds because $\mathcal{S}_i^*$ is a packing in G^i while $S_1^i, \ldots, S_m^i$ is a partitioning. The third inequality holds because $w(S_j^i) < \Delta\lambda$ for $j \in [m]$, as is guaranteed by `BalancedPartition`. Now since $\Delta\lambda \leq W^*$, we derive $|\mathcal{S}_i^*| < m$ from (3.1), and since $m - 1 \leq |\mathcal{S}^i|$ (recall that the last two templates might be merged), we conclude that $|\mathcal{S}_i^*| \leq |\mathcal{S}^i|$.

Taking everything together, we derive a contradiction to $|\mathcal{S}| < k$:

$$|\mathcal{S}| \;=\; |H| + \sum_{i=1}^{\ell} |\mathcal{S}^i| \;\geq\; |\mathcal{S}_H^*| + \sum_{i=1}^{\ell} |\mathcal{S}_i^*| \;=\; |\mathcal{S}^*| \;=\; k.$$

□

Let us finally examine the time complexity of `MaxMinPartition`. Finding the heavy vertices H and computing the resulting connected components of $G - H$ can be performed in time $\mathcal{O}(|E|)$. The algorithm `BalancedPartition` runs in time $\mathcal{O}(|E(G^i)|)$ for every subgraph $G^i \in \mathcal{G}^{\geq}$. Since the graphs G^i are pairwise disjoint, the total running time of `MaxMinPartition` is $\mathcal{O}(|E|)$.

Theorem 3.6 *Let (G, w, k) be an instance of max-min CVP with optimal objective value W^*, and let Δ be the maximum degree of a spanning tree of G. Then, a Δ-approximation for this instance can be computed in $\mathcal{O}(\log(W^*)|E|)$.*

Proof. First, let us assume integer node weights, which we can always obtain by appropriate multiplication of rational weights. Then, the optimal value W^* is also integer, and for different integer values W we choose $\lambda = \lfloor \frac{W}{\Delta} \rfloor$. If we find that for some λ^*, `MaxMinPartition` yields a partition $\mathcal{S}$ with at least k templates, but for $\lambda^* + 1$ we obtain less than k templates. Then,

$$W^* < \Delta(\lambda^* + \varepsilon) = \Delta\lambda^* + \Delta\varepsilon$$

□

3.2.4 2-Approximation for Edge Partitioning

In this secion, we extend the presented algorithms to achieve a 2-approximation for the respective connected edge partitioning problem (cf. Section 2.5), denoted as CEP. As detailed in Section 2.7, we transform the problem to the line graph, and solve it as a CVP. Knowing that we are concerned with a line graph, however, we can improve on the approximation guarantee.

In 1970, Beinecke [Bei70] proved a characterization of line graphs in terms of forbidden subgraphs. One result is that a line graph cannot contain a claw, i.e., a $K_{1,3}$, as induced subgraph. This has an interesting implication concerning the minimum degree spanning tree of line graphs.

Proposition 3.7 *Every line graph has a spanning tree T with $\Delta(T) \leq 3$. Furthermore, every tree found with depth-first search (DFS) has this property.*

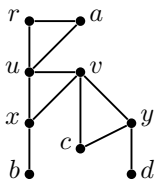
Proof. Let L be a line graph of some simple graph and let T be a tree obtained by depth-first search in L. Assume that T has a vertex v of degree ≥ 4, then v has at least three child nodes a, b, c. Since T is a DFS tree, we know that $ab, ac, bc \notin E(L)$. Thus, L contains an induced claw on the nodes $\{v, a, b, c\}$, contradicting the assumption that L is a line graph. □

With a transformation of the problem to the line graph, this result immediately gives us a 3-approximation for the min-max CEP and max-min CEP. However, we can use the fact that the line graph $L(G)$ of G is claw-free for an even stronger result. Namely, we obtain a 2-approximation for both variants of the CEP. The following lemma lays the foundation.

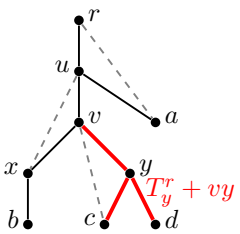
Lemma 3.8 *Let $G = (V, E)$ be a claw-free graph with vertex weights $w \in \mathbb{R}^V_{>0}$ and let $\lambda \in [w_{\max}, w(G)]$. Then there exists a tree $S \subseteq G$ with $w(S) \in [\lambda, 2\lambda)$ such that $G - S$ is connected.*

Proof. The proof is constructive. Consider a DFS spanning tree $T = T^r$ of G which is rooted at some node r. As G is claw-free, $\deg_T(r) < 3$. From Proposition 3.7 we know that $\Delta(T) \leq 3$ and hence, every node in T has at most two child nodes.

If there exists some $v \in T^r$ such that $w(T^r_v) \in [\lambda, 2\lambda)$ we are done since $G - T^r_v$ is connected. In the remaining case there has to exist a node v with child nodes x and y such that $w(T^r_v) \geq 2\lambda$ while $w(T^r_x) < \lambda$ and $w(T^r_y) < \lambda$. In particular, note that $w(T^r_l) \leq \lambda$ for every leaf l in T^r and that v with $w(T^r_v) \geq 2\lambda$ cannot have a single child node x with



(a) line graph L



(b) DFS spanning tree of L with highlighted subtree $T^r_y + vy$

Figure 3.4.: Exemplary transformation from line graph to DFS spanning tree.

Algorithm 3.3: `BalancedLineGraphPartition`(L, w, λ)

Input: line graph $L = (V, E)$, $w \in \mathbb{R}^V_{>0}$, $\lambda \geq w_{\max}$
Output: trees $T_1, \ldots, T_m \subseteq L$ with $V = \dot{\bigcup} V(T_i)$

```
1 T := ∅
2 while w(L) ≥ 2λ do
3     S := tree in L as constructed in Lemma 3.8
4     T := T ∪ S
5     L := L − S
6 S := spanning tree of L
7 return T ∪ S
```

$w(T^r_x) < \lambda$. Consequently, we have

$$\lambda = 2\lambda - \lambda \leq w(T^r_v) - w(T^r_y) = w(T^r_x) + w_v < \lambda + \lambda = 2\lambda$$

to prove $w(T^r_x) + w_v \in [\lambda, 2\lambda)$, and analogously we obtain $w(T^r_y) + w_v \in [\lambda, 2\lambda)$. So if $v = r$, we choose the tree consisting of T^r_y extended with the edge vy, which we denote by $S = T^r_y + vy$, and find that $G - S$ is still connected. If $v \neq r$, then v has a parent node u. Because T^r is a DFS tree we know that $xy \notin E$, and since G is claw-free, we know that $ux \in E$ or $uy \in E$. W.l.o.g. we assume the former and conclude that $S = T^r_y + vy$ is a sought tree in G. □

This result can be used to define an algorithm for tree partitioning of line graphs as listed in Algorithm 3.3. Analogously to `BalancedPartition`, this algorithm yields a quasi-$[\lambda, 2\lambda)$-partition, and due to Theorem 3.1, we immediately obtain a 2-approximation for the min-max connected edge partitioning problem (CEP). The same approximation for the max-min CEP (without condition on $w_{\max}$) is achieved from `MaxMinPartition` by replacing each call to `BalancedPartition` with the same call to `BalancedLineGraphPartition`.

4

Homogeneous Vertex Covering

Grouping similar entities in a complex network is a common and essential task in many applications. Such homogeneous covering problems occur in social studies, political districting, and sales territory design. That is also the reason why the research on this topic is mainly driven by the operations research community. As many applications require disjoint template graphs, the partition variant is predominant in this field. In addition, the unique structure of a partition often makes the problem more amenable to theoretical results. For the problems discussed here, the cardinality of the partitioning may be part of the input or not.

As mentioned above, homogeneity and balancing are not necessarily opposing objectives. If for a homogeneous cover, the template size is bounded by upper or sometimes lower bounds, we call this a capacitated covering. The other case is referred to as uncapacitated covering. Uncapacitated partitioning is closely related to data clustering, where data points are to be grouped in several clusters of similar data points. Formulated as a vertex partitioning problem, however, the host graph is usually complete. This makes methods for general vertex partitioning unattractive for data clustering problems. On the other hand, methods for data clustering are usually inapplicable for general vertex partition problems on arbitrary host graphs or with constrained templates. Therefore, we will not discuss problems or techniques for data clustering in this thesis, but refer to [JMF99; HPK11].

Instead, we begin this chapter by examining different possibilities to measure homogeneity and by presenting three problem classes that are central for homogeneous vertex coverings. We then proceed to overview the most important heuristic (Section 4.2) and exact (Section 4.3) solution approaches. In Section 4.4, we will discuss a method to efficiently incorporate connectivity requirements.

4.1 What is Homogeneity?

The main question for homogeneous covers is: How should we measure homogeneity? We follow the presentation of the survey article by Hansen and Jaumard [HJ97]. Assume that the dissimilarity between any two vertices is given by a symmetric, non-negative matrix (d_{uv}) with zeros on the diagonal. For given vertex weights $w \in \mathbb{R}^V_{\geq 0}$, this could be $d_{uv} = |w_u - w_v|$. For given edge weights, on the other hand, d_{uv} could be the shortest path distance between u and v. Another possibility is that (d_{uv}) is the adjacency matrix with edge weights as entries.

Two ways to achieve homogeneous groups are to maximize the separation between different groups or the homogeneity within the single groups. Given the dissimilarities, we first focus on measures for the separation or homogeneity of a single template T. Concerning separation-based measures, we list the following two.

- split $s(T) := \min_{u \in T,\, v \notin T} d_{uv}$,
- cut $c(T) := \sum_{u \in T} \sum_{v \notin T} d_{uv}$.

For a fixed template, both measures consider vertex pairs where exactly one vertex belongs to the template. While split is concerned with the dissimilarity of the worst-case pair, the cut objective considers the sum over all dissimilarities. Hence, cut is a more robust measure than split. In order to obtain a homogeneous cover, we maximize the value of the worst template or the sum over all templates, respectively.

In addition to separation-based measures we can also consider homogeneity-based measures. For these we consider dissimilarities of vertex pairs with both vertices in a fixed template T. Here, we discuss the following measures.

- diameter $d(T) := \max_{u,v \in T} d_{uv}$,
- radius $r(T) := \min_{u \in T} \max_{v \in T} d_{uv}$,
- star $st(T) := \min_{u \in T} \sum_{v \in T} d_{uv}$,
- clique $cl(T) := \sum_{u,v \in T} d_{uv}$,
- sum-of-squares $ss(T) = \sum_{v \in T} (w_v - \overline{w}_T)^2$ where $\overline{w}_T = \frac{1}{|T|} \sum_{v \in T} w_v$.

Again, we can optimize the value of the worst template or the sum over all templates, but in contrast to separation-based measures, we aim for a minimization. As described in Hansen and Jaumard [HJ97] these measures may also have normalized variants.

Let us take a closer look at the measures (cf. [HJ97]). Note that for the partitioning case, maximizing the sum of cut values is equivalent to minimizing the sum of clique values. This does only hold if we consider the sum of the template values instead of the value of the worst template. Another useful observation is that radius, star, and sum-of-squares make use of a center that can also be used as a representative of a template. While radius considers the maximum distance (or dissimilarity) to the center, the star criterion takes into account all distances. In this sense, the measures are not only comparable to split and cut, but also have a relation to facility location. In a nutshell, given a number of points, a distance matrix, and a number k, the facility location problem is to determine locations for k facilities to minimize the distance to the given points. The two most prominent objectives for this problem are k-center and k-median. The former aims to minimize the maximum distance from a point to its closest center which is equivalent to the worst-case radius objective. On the other hand, the k-median aims to minimize the sum over all distances from a point to its closest center which is equivalent to the sum of stars objective. A special feature of facility location problems is that the host graph is usually complete and, hence, connectivity of templates is not an issue. Finally, note that the sum-of-squares can easily be extended to multidimensional vertex weights by considering the (squared) Euclidean distance between a vertex weight and the mean vertex weight.

We stress that the literature covers even more than the presented measures for homogeneity. However, our selection is concentrated on the most prominent ones in the operations

research community. Depending on the dissimilarity measure, most of the objective functions tend to favor connected templates. This is also desired in various applications and, thus, many models incorporate such connectivity constraints. Before we address different optimization methods, we single out a special homogeneous vertex covering problem and describe two independent fields of application that drive the research on homogeneous covers: community detection and territorial districting.

Graph Partitioning: A famous variant of vertex covering is the partitioning case where the host graph is edge-weighted and the dissimilarities are given by the weighted adjacency matrix. The problem is often considered under the name graph partitioning, and when referring to this particular version, in contrast to a general vertex covering, we will adopt this notion here. The almost universally accepted measure for the quality of a partition is the cut objective, sometimes also in normalized form. In many cases a capacitated clustering is sought, i.e., additional balancing constraints hold the templates to similar sizes. Various books and survey articles (e.g. [Fjä98; BS11; Bul+16]) detail problems, methods and applications for this particular vertex partitioning problem. In the optimization methods overview below, we also discuss results on graph partitioning.

Community Detection: The advances in this field are driven by the analysis of social and biological networks that are typically very large and in most cases unweighted. Problems of community detection are considered in the partitioning but also in the covering version. The name stems from the studying of social or citation graphs where one can observe that two neighbors of the same vertex are more frequently linked by an edge than two arbitrary vertices [GN02]. Accordingly, we can find subgroups of people with relatively dense internal connections. By identifying these communities, we can better understand the structure and the interdependencies in these networks.

The problem is, however, ill-posed and various measures for the community analysis are considered. Let us single out the modularity maximization, defined by Newman and Girvan [NG04]. Informally, the modularity measures the fraction of edges inside of the communities minus this (expected) fraction in a random graph with identical degree distribution. For the specifics, different variants of modularity, and a plethora of other measures we refer the reader to the survey on community measures by Chakraborty et al. [Cha+17]. While the modularity maximization is among the most popular objectives in community detection, it has a notable resolution limit, i.e., it tends to ignore relatively small clusters even if they are well defined communities like cliques [FB07]. Furthermore, as most objectives, modularity maximization is NP-hard [Bra+07].

Due to the size of the host graphs the solving methods in this field are not exact but rely on heuristic approaches. Schaeffer [Sch07] gives an excellent survey over prominent solution techniques, but there are also more recent surveys [FH16; Jav+18].

As pointed out above, the applications of community detection go well beyond the analysis of social networks. Closely related problems arise in the analysis of different biological networks. Here, the clustering helps to uncover the mechanisms behind gene expression [XOX02] or protein-protein interaction [BH03; KPJ04].

Territorial Districting: While community detection is concerned with abstract graphs, territorial districting problems are usually tied to graphs representing geographic maps. The predominant application is political districting but the same problem occurs in the planning of school districts [FG90], police patrol areas [Ami+02; CHQ10; CCY19], or in

sales territory design [HS71; FP88; ZS05; SRC11]. Motivated from these applications, the number of districts is usually given and the goals of the partition are threefold. Resulting districts should be contiguous, compact, and balanced with respect to given vertex weights (e.g., the number of voters, children, or customers), While the contiguity is usually ensured by constraints on the template graphs to be connected, the balance of the templates can be implemented as constraints or as part of the objective function. The compactness of districts is the most vague goal among these. Informally, "a round-shaped district is deemed to be acceptable, while an octopus- or an eel-like one is not" [RSS13]. The compactness can be formalized by using homogeneity measures such as star or radius with appropriate distance measures for every pair of nodes (e.g., bee-line distance or shortest path distance).

Two excellent survey articles on this topic are due to Ricca, Scozzari, and Simeone [RSS13] with an emphasis on optimization methods and by Kalcsics and Ríos-Mercado [KR19] with an emphasis on different variants and applications. A recent book summarizes latest results, models, and applications [Río20].

Let us now turn to different approaches for solving homogeneous covering and partitioning problems. We are focused on works of the operations research community with the presented homogeneity measures. These papers mostly relate to districting applications or graph partitioning while several (heuristic) methods are also used in community design. Table 4.1 categorizes contributions to homogeneous vertex covering problems.

Table 4.1.: Homogeneous vertex covering contributions: Entries with dashed underlines do not impose the connectivity of the templates as a constraint.

	heuristic	exact
split	[Han+03]	[HJM90], [MSN97]
cut	[KL70], [DH72], [Fie75], [SM00], [Bic11], [Kim+11], [LK13], [SS13]	[GW89], [BCR97], [Sen01], [Arm+08a], [LÖ10], [Sør17], [Hoj+21], [PW21], [VB22]
diameter	[Ami+02], [RS11]	[GN70]
radius	[RS08], [RF09], [ERD14]	[Nyg88], [RF09], [SRC11], [Lar+16], [CGP19], [BSS23]
star	[FP88], [BLP05], [Seg+07]	[MJN98], [Seg+07], [Apo+08], [SRC11], [BSS23]
clique	[ABR92], [MSN97], [HFV99], [DAR12]	[JMN93], [Fer+96], [Fer+98]
sum-of-squares	[MS95]	[Hes+65]
survey	[Mur85], [Gor96], [HJ97], [SKK00], [HM03], [RSS13], [Bul+16]	

4.2 Heuristic Solution Approaches

Especially for large instances, many common solution methods rely on hierarchical clustering that can be either agglomerative (*bottom-up*) or divisive (*top-down*). The former start with singleton templates that are successively merged, for instance in a greedy fashion. On the other hand, divisive methods start with the host graph as single template and repeatedly divide the current template graphs, for instance by computing minimum cuts or by spectral methods exploiting eigenspaces of the graph's Laplacian [DH72; Fie75; SM00].

Similar to agglomerative approaches, graph coarsening is a method where multiple vertices are contracted to a single vertex. This process can either be repeated until a sought number of vertices remains, or other optimization methods can be applied to the coarsened graph. Bichot [Bic11] gives an overview on this topic and discusses different coarsening and uncoarsening techniques. This approach is also particularly well suited for parallelization [Pel11; LK13].

Another heuristic approach is an adaption of the k-means algorithm. Iteratively, template centers are chosen (*location*) and every vertex is then assigned to its closest center (*allocation*). Again, a new template center can be determined for every template and the process iterates until a fixed point, i.e., a local optimum, is found. This process was first used by Hess et al. [Hes+65]. Later, Segura-Ramiro et al. [Seg+07] present an adaption for the case that the templates should also be balanced.

Graph partitioning techniques from computational geometry are reviewed by Schloegel, Karypis, and Kumar [SKK00]. Geometric approaches are based on coordinates of the vertices instead of the underlying graph structure. If such coordinates are not given, they can be set based on the connectivity of the graph [Hal70]. Again, many solution methods are based on recursive bisection (see e.g. [HR95; GMT98]). Another approach, lying in between geometric and location-allocation methods, uses Voronoi diagrams to partition the space. After fixing the Voronoi centers, the resulting Voronoi cells give a partition of the space and, therefore, of a graph embedded therein. Partitioning using Voronoi regions is applied for community detection [Der+14] and also for political districting [RSS08].

If the host graph is a tree, many homogeneous covering problems can be solved in polynomial time. While Maravalle and Simeone [MS95] prove this for the sum-of-squares measure, Hansen et al. [Han+03] consider the split version. Both exploit this fact for another heuristic approach where the problem is solved on a spanning tree of the host graph.

Apart from these constructive approaches, several local improvement heuristics can be employed. We start with approaches specialized for graph bipartition. Note that the uncapacitated version of this problem can efficiently be solved using a max-flow min-cut algorithm [FF56]. Including balancing constraints, the first and most famous improvement heuristic is due to Kernighan and Lin [KL70]. The method iteratively swaps two vertex sets of same cardinality in different parts of the current bipartition. The sets are chosen so as to improve the objective value as much as possible. Many other heuristics for graph bipartitioning either refine or build on this algorithm (e.g. [FM82; KK98]). A more modern approach by Sanders and Schulz [SS13] makes local improvements by detecting negative cycles.

Regarding general vertex partition problems, greedy local search techniques are used in [HFV99], [RF09], and [DAR12] for example. Hansen and Mladenović [HM03] propose a variable neighborhood search that uses multiple neighborhood definitions. If a local

optimum in a neighborhood is found, one can simply change the neighborhood and continue the search. Other improvement approaches include tabu search [ABR92; BEL03; DAR12], simulated annealing [ABR92; Ami+02; DAR12], or genetic algorithms [BLP05; Kim+11]. A comparison of different local search methods for districting problems is done by Ricca and Simeone [RS08]. In particular, they compare the greedy local search, tabu search, simulated annealing, and another heuristic called *old bachelor acceptance* that is due to Hu, Kahng, and Tsao [HKT95]. Their computational results based on political districting instances for Italy show that the latter produces the best results in the majority of the cases.

4.3 Exact Solution Approaches

In general, exact methods in this field rely on integer programming formulations. We start with the few exact approaches that follow a different route: Hansen, Jaumard, and Musitu [HJM90] present an exact approach for the split case without connectivity constraints in which a sequence of bin packing problems is solved. Sensen [Sen01] solves the graph partitioning problem with a branch-and-bound algorithm where the bounds are based on multi-commodity flow problems.

With respect to IP formulations, we can group the approaches into two categories: Formulations where we have one variable per possible template, and those where the optimal templates are composed by variables related to the vertices of the host graph.

4.3.1 Column Generation

A naive approach for the vertex covering problem uses one variable per feasible template. Let $\mathcal{T}$ denote the set of feasible templates, and assume that the inhomogeneity of template $T \in \mathcal{T}$ is expressed by the penalty term p_T. Then, we can use the following classical covering IP formulation:

$$\min_{x} \quad \sum_{T \in \mathcal{T}} p_T \, x_T \tag{4.1a}$$

$$\text{s.t.} \quad \sum_{T \ni v} x_T \;\geq\; 1 \qquad \forall v \in V \tag{4.1b}$$

$$x_T \;\in\; \{0,1\} \quad \forall T \in \mathcal{T} \tag{4.1c}$$

We have a binary variable x_T for every feasible template T, indicating if T is used in the covering, and minimize the cumulated penalty (4.1a) while covering all vertices (4.1b). The partitioning case is obviously modeled by replacing the inequality in (4.1b) with an equality. If we are not optimizing with regard to the cumulated (in)homogeneity, but aim to minimize the maximum penalty $p_{\max}$ of all covering templates, this can be modeled with the additional constraint $p_{\max} \geq p_T \, x_T$.

While this formulation is simple and self-evident, it quickly becomes impracticable to enumerate all feasible templates or determine all associated penalties. Early attempts indeed enumerate a set $\mathcal{T}$ of promising templates to run the above model with. Garfinkel and Nemhauser [GN70] solve a political districting problem with up to 2495 possible templates, Nygreen [Nyg88] enumerates up to 564 possible templates for a similar problem.

In many applications, however, we can expect millions of feasible templates. In such situations, a column generation approach, where new templates are generated only if necessary, is an excellent tool. We will go into the details of this approach in Section 6.2.2, and continue here with the literature overview.

Johnson, Mehrotra, and Nemhauser [JMN93] are the first to employ column generation in a branch-and-price framework. Mehrotra and Trick [MT98] copy this approach but develop a new method to tackle the pricing problem. A column generation approach for connected templates is done by Mehrotra, Johnson, and Nemhauser [MJN98]. They solve the homogeneous partitioning problem with star objective in a case study concerning electoral districts in South Carolina. To reduce the complexity of the problem, the set of possible template graphs is further restricted. Namely, it is required that with any vertex in a template also all vertices on a shortest path to the template center have to belong to this template. This constraint dramatically reduces the complexity of the subproblem [SC98]. Clautiaux, Guillot, and Pesneau [CGP19] consider a similar problem motivated by an application in waste collection. They also use column generation and solve the NP-hard pricing problem with Lagrangian relaxation and a dynamic program. However, identical to [MJN98], they demand that each template is a subtree of the shortest path tree to its center. In Section 6.2.2, we will present an approach to allow arbitrary connected subgraphs within the pricing problem.

4.3.2 Compact Models

Apart from these approaches to dynamically generate possible templates, there are compact models that merge vertices to optimal templates. The models are almost exclusively designed for graph partitioning, i.e., the templates are not required to be connected, and we minimize the cumulated weight of all cut edges. In the following, we review three different IP formulations to model the graph partitioning problem. We also examine how connectivity constraints can be incorporated into the formulations. For this purpose, there are basically two possible approaches: Enforcing connectivity by some kind of flow variables, or dynamically separating non-connected solutions within a branch-and-cut framework.

Recall that for partitioning problems, the min-cut and max-clique objectives are equivalent and, indeed, both variants are used in these formulations. For the following exposition, we assume that $V = \{1, \dots, n\}$ and denote the edge weights of the host graph by (w_{uv}).

Triangular Model: The triangular formulation (TRI) was designed for the clique partitioning problem, i.e., to partition the vertices of a complete graph with arbitrary edge weights $w \in \mathbb{R}^E$ into any number of cliques while minimizing the cut weight. The formulation of Grötschel and Wakabayashi [GW89] uses binary variables x_{uv} for any pair of vertices with $u < v$ to indicate whether u and v belong to the same template:

(TRI)

$$\max_x \sum_{uv \in E} w_{uv}\, x_{uv} \tag{4.2a}$$

$$\text{s.t.} \quad x_{uv} + x_{uw} - x_{vw} \le 1 \qquad \forall u, v, w \in V : u < v < w \tag{4.2b}$$

$$x_{uv} - x_{uw} + x_{vw} \le 1 \qquad \forall u, v, w \in V : u < v < w \tag{4.2c}$$

$$-x_{uv} + x_{uw} + x_{vw} \le 1 \qquad \forall u, v, w \in V : u < v < w \tag{4.2d}$$

$$x_{uv} \in \{0, 1\} \qquad \forall u, v \in V : u < v \tag{4.2e}$$

These *triangle inequalities* model the transitivity of the equivalence relation of vertices belonging to the same template. The same formulation can be used for the graph partitioning problem, i.e., to partition the vertices of an arbitrary graph into arbitrary sets while minimizing the cut: It simply suffices to set $w_{uv} = 0$ for all $uv \in \binom{V}{2} \setminus E$. The equivalent interpretation is to add all non-existing edges to the host graph and setting their weight to 0.

This approach, however, has a drawback: Although the original host graph might be sparse, the formulation needs $\mathcal{O}(n^3)$ constraints. By showing that it is sufficient to consider only triples u, v, w such that at least two of these nodes are adjacent, Nguyen et al. [Ngu+17] are able to reduce this to $\mathcal{O}(mn)$.

Grötschel and Wakabayashi [GW90] carry out a polyhedral study and find various facets of the corresponding polytope. The resulting cutting plane algorithm [GW89] solves all considered instances in the root node, i.e., no branching is necessary. Oosten, Rutten, and Spieksma [ORS01] continue the polyhedral study and provide more facets for this polytope.

The TRI formulation has another drawback, as it is not possible to incorporate a bound k on the size of the partitioning (without introducing more variables, leading to formulations similar to the ones discussed next). On the positive side, it is possible to model the capacitated partitioning problem by adding the constraint

$$\sum_{uv \in E} w_u \, x_{uv} \;\leq\; W_U - w_v \qquad \forall v \in V, \tag{4.3}$$

or a similar constraint for a lower weight bound on each template (cf. [LÖ10; PW21]).

Faigle, Schrader, and Suletzki [FSS87] appear to have been the first to carry out a polyhedral study of the capacitated clustering problem, considering unit weights on the vertices and an upper cardinality bound on the templates. Labbé and Özsoy [LÖ10] examine the polytope if the cardinality of the templates is bounded from below and above.

Vertex-to-Cluster Formulation: The vertex-to-cluster formulation (VTC) assigns each vertex to one of the k possible clusters. If the number of templates is not restricted, we have to set $k = n$. We consider binary variables x_v^i for $v \in V$, $i \in [k]$ to indicate if vertex v is assigned to template i. We need additional binary variables z_{uv} for $uv \in E$ to indicate whether u and v belong to different templates, and should hence be accounted for in the objective function. The approach was first stated by Johnson, Mehrotra, and Nemhauser [JMN93], but we will mainly follow the presentation of Validi and Buchanan [VB22].

$$\max_{x,z} \sum_{uv \in E} w_{uv}\, z_{uv} \tag{4.4a}$$

$$\text{s.t.} \quad \sum_{i \in [k]} x_v^i = 1 \qquad \forall v \in V \tag{4.4b}$$

$$z_{uv} \geq x_u^i - x_v^i \qquad \forall u, v \in E,\ \forall i \in [k] \tag{4.4c}$$

$$x_u^i + x_v^i + z_{uv} \leq 2 \qquad \forall u, v \in E,\ \forall i \in [k] \tag{4.4d}$$

$$x_v^i \in \{0, 1\} \qquad \forall v \in V,\ \forall i \in [k] \tag{4.4e}$$

$$z_{uv} \in \{0, 1\} \qquad \forall u, v \in E \tag{4.4f}$$

(VTC)

Constraints (4.4b) ensure that each solution is indeed a partition, while constraints (4.4c) force that $z_{uv} = 1$ whenever uv is a cut edge. The constraints (4.4d) are only necessary if there are negative edge weights, and in this case, they enforce the reverse of (4.4c). Using the x variables, this model can easily be extended for the capacitated case.

A major issue of this formulation is symmetry, meaning that any feasible partitioning is represented by $k!$ equivalent solutions of VTC that come from permuting the template labels. Such symmetries cause trouble in the branch-and-bound procedure and, hence, different approaches to break symmetries have been proposed (see [Mar10; PR19] for an overview).

The symmetries occurring here are very similar to the ones arising in the classical vertex coloring formulation. No wonder that approaches to break those, e.g., from [MDZ00], transfer to our formulation. One of the standard tricks is to enforce that the template with label i contains no smaller nodes than i (recall that $V = [n]$), which can be done by the inequalities

$$x_v^i \leq x_i^i \text{ for } v > i \quad \text{and} \quad x_v^i = 0 \text{ for } v < i. \tag{4.5}$$

While this eliminates the symmetry only to some degree, Kaibel and Pfetsch [KP08] propose a set of linear inequalities to remove all symmetry. Faenza and Kaibel [FK09] use an extended formulation to drastically reduce the number of necessary inequalities. Validi and Buchanan [VB22] confirm a positive impact of this formulation for a districting problem. They also introduce a number of variable fixing techniques for the capacitated version.

Kaibel, Peinhardt, and Pfetsch [KPP11] handle the symmetries of VTC in a different way. Instead of adding symmetry breaking inequalities, they propose a number of fixing rules to apply in the nodes of the branch-and-bound tree. A generalization of this concept, called orbital branching and fixing, is presented in [Ost+11] (not oriented towards graph partitioning).

With modern MIP solvers constantly evolving, simple symmetry breaking inequalities become less important for the modeler, as many symmetry issues are handled within the solver during the presolve or via orbital branching and fixing [Ach+20].

Problem specific polyhedral insights, on the other hand, can considerably strengthen the formulations. For the capacitated graph covering problem with unit weights, Sørensen finds various classes of facet-defining inequalities [Sør04; Sør07; Sør17]. Ferreira et al. [Fer+96] study the capacitated version on a vertex-weighted host graph. They find different facets and valid inequalities for this polytope and employ these in a branch-and-cut framework [Fer+98].

Concerning connected templates, Validi, Buchanan, and Lykhovyd [VBL22] consider four connectivity formulations, two based on flows and two based on cuts. They find that the cut approaches perform better and argue that the compactness objective of the templates often suffices to induce connectivity. Hojny et al. [Hoj+21] compare a single-commodity flow and a node separator approach and find the flow formulation to be a competitive way to model connectivity. The flow formulations require the designation of a root of each template, leading to formulations that are detailed next.

Vertex-to-Root Formulation: The idea of the vertex-to-root formulation (VTR) is that one node of each template graph serves as representative (or root) of this template. Instead of assigning the vertices to cluster labels as in the VTC formulation, we assign the vertices to cluster representatives. Hess et al. [Hes+65] were the first to formulate this model in the context of political districting. Here, we have binary variables x_v^r for $v, r \in V$ that indicate if vertex v is assigned to the template rooted at $r \in V$. The z variables are identical to VTC.

$$\min_{x,z} \sum_{uv \in E} w_{uv}\, z_{uv} \tag{4.6a}$$

$$\text{s.t.} \quad \sum_{r \in V} x_v^r = 1 \qquad \forall v \in V \tag{4.6b}$$

(VTR)

$$z_{uv} \geq x_u^r - x_v^r \qquad \forall u, v \in E,\ \forall r \in V \tag{4.6c}$$

$$x_v^r \in \{0, 1\} \qquad \forall v \in V,\ \forall r \in V \tag{4.6d}$$

$$z_{uv} \in \{0, 1\} \qquad \forall u, v \in E \tag{4.6e}$$

The model is very similar to VTC. In fact, we simply replaced every 'i' with 'r' and every '$[k]$' with 'V', i.e., instead of assigning vertex v to k possible templates, we now assign v to n possible representatives, increasing the number of variables and worsening the symmetry of the model. So what is the advantage of this model? The variables x_r^r that characterize the roots of the templates can be very helpful to integrate connectivity constraints. Again, the x variables make it easy to formulate capacity constraints on each template or a maximum covering size.

The symmetry of the VTR formulation is even worse compared to VTC. However, we cannot only add the adapted inequalities (4.5), enforcing that the representative of each template is its smallest vertex, but also improve them. As Validi and Buchanan [VB22] point out, the concept of "smallest" can be extended to any ordering on the vertices. More specifically, they construct an ordering that exploits the lower and upper capacity bounds on connected templates which leads to significant reductions of the problem size: In their experiments on large districting instances with hundreds of nodes, over 95% of the x variables could be fixed.

Special Cases: A special case of capacitated clustering is the equipartition or equicut problem. The goal is a partition of the vertices into two sets that differ by at most 1 in cardinality. The associated polytope is studied by Conforti, Rao, and Sassano [CRS90] and Souza and Laurent [SL95]; a branch-and-cut algorithm and a computational study is presented by Brunetta, Conforti, and Rinaldi [BCR97]. In order to limit the size of the branch-and-bound tree, Lisser and Rendl [LR03] and Anjos et al. [Anj+13] develop improved bounds for this problem.

The capacitated bipartition without requiring the parts to be of nearly equal cardinality is studied by Armbruster et al. [Arm+08b] from a polyhedral standpoint with an accompanying computational study in [Arm+08a].

Polytopes for partitioning a clique into a bounded number of templates are also considered by several authors [DGL92; CR95; AKP16; AK20]. If the host graph is a tree and the template centers are fixed, Apollonio et al. [Apo+08] prove that the polytope of connected centered partitions is integral and, hence, optimal partitions can be found efficiently. While [Apo+08] is focused on the star objective, Lari et al. [Lar+16] discuss the radius objective. A polyhedral study for arbitrary host graphs is due to Chopra and Rao [CR93]. In addition, Chopra [Cho94] finds that cycle inequalities suffice to define the polytope if the host graph is series-parallel. Finally, we note that for a fixed number of templates, the uncapacitated graph partitioning problem can be solved in polynomial time [GH94].

4.4 Connectivity with Few Roots

We have seen that many covering (or partitioning) problems require connectivity of the templates. If the connectivity is enforced with a flow, we need root nodes that serve as sources (or sinks) for the flow. The VTR formulation allows every vertex as potential root, but if we are given a lower weight bound W_L on the templates – as is usual in districting problems – we might be able to drastically reduce the number of roots that is needed. Our goal is to remove as few vertices as possible from G such that any connected component in the resulting graph has cumulated weight less than W_L. Consequently, any connected subgraph with weight at least W_L has to contain at least one of the removed vertices, and we can choose these as roots. In the literature, such a vertex set is known as balanced separator, but its definition is not consistent. Here, we use the following:

Definition 4.1 *Let $G = (V, E)$ be a graph with node weights $w \in \mathbb{R}^V_{\geq 0}$, $X \subseteq V$, and $\alpha \in [0, 1]$. We call X an α-balanced separator if any connected component C in $G - X$ satisfies $w(C) \leq \alpha\, w(G)$.*

In this section, we consider a more general problem that we can also use for finding a minimum number of roots for connectivity.

Minimum Balanced Separator

Instance: A graph $G = (V, E)$ with node weights $w \in \mathbb{R}^V_{\geq 0}$ and $\alpha \in [0, 1]$.

Problem: Find an α-balanced separator in G of minimum cardinality.

Unfortunately, this problem is NP-hard. By setting $w \equiv 1$ and $\alpha = \frac{1}{|V|}$, we basically search for a minimum vertex cover, which is one of the classical NP-hard problems. If we set $\alpha = \frac{W_L - \varepsilon}{w(G)}$ for ε sufficiently small, Minimum Balanced Separator models our problem of minimizing the number of necessary roots.

Balanced separators have a variety of other applications, and the book by Rosenberg and Heath [RH01] discusses many of them. Most notably, they can be used in divide-and-conquer algorithms reducing a given problem to a number of smaller instances [LT80], which, in turn, finds application for instance in VLSI layout for computer chip design [BL84].

The problem was originally introduced by Lipton and Tarjan [LT80] who also gave the first approximation result. For planar graphs, this has been improved in [Dji00; Ale+07], and for graphs with maximum degree 3 in [BJ92]. Preprocessing techniques to reduce the problem size are studied in [Cas+21]. Other works focus on the separation into two vertex sets, either from an exact [BS05; SB05; SC11] or approximative [FHL08; Hol+10; BW17] view.

Here, we are going to discuss two exact approaches for Minimum Balanced Separator. We start by presenting an adaption of a mixed-integer programming (MIP) formulation from [Eli18]. The formulation is very large and has a weak LP relaxation. Therefore, we propose a different model and solve the Minimum Balanced Separator problem by dynamically adding violated constraints to an alternative formulation. Our computational comparison of the two approaches shows that the latter performs significantly better.

4.4.1 Compact Flow Formulation

Elijazyfer [Eli18] proposes a MIP formulation to find α-balanced separators in edge weighted graphs. In the following, we present the adapted version to the node weighted case. The connectivity is based on a flow formulation which requires directed graphs. To turn G into a directed graph, we define A as the bidirected arc set of the edges in E. The idea is to search for an arc partition such that each component induces a connected subgraph and its cumulative weight (without separator nodes) does not exceed $\alpha\, w(G)$. The set of vertices occurring in more than one component constitutes an α-balanced separator. Therefore, the objective is to minimize the cardinality of this vertex set, and the MIP formulation is as follows.

$$\min_{x,s,c,y,q} \quad \sum_{v\in V} x_v \tag{4.7a}$$

$$\text{s.t.} \quad \sum_{r\in V} y_a^r = 1 \qquad \forall a \in A, \tag{4.7b}$$

$$y_{(u,v)}^r = y_{(v,u)}^r \qquad \forall r \in V\ \forall (u,v) \in A, \tag{4.7c}$$

$$2\, y_{(u,v)}^r \le s_u^r + s_v^r \qquad \forall r \in V\ \forall (u,v) \in A, \tag{4.7d}$$

$$q_a^r \le y_a^r\,(|V|-1) \qquad \forall r \in V\ \forall a \in A, \tag{4.7e}$$

$$\sum_{v\in\delta^+(v)} q_a^r - \sum_{v\in\delta^-(v)} q_a^r \ge s_v^r \qquad \forall r \in V\ \forall v \in V\setminus\{r\}, \tag{4.7f}$$

$$1 + x_v\,(|V|-1) \ge \sum_{r\in V} s_v^r \qquad \forall v \in V, \tag{4.7g}$$

$$\sum_{v\in V} (s_v^r - c_v^r)\, w_v \le \alpha\, w(G) \qquad \forall r \in V. \tag{4.7h}$$

$$c_v^r \le s_v^r \qquad \forall r \in V\ \forall v \in V, \tag{4.7i}$$

$$c_v^r \le x_v \qquad \forall r \in V\ \forall v \in V \tag{4.7j}$$

$$x_v \in \{0,1\} \qquad \forall v \in V, \tag{4.7k}$$

$$y_a^r \in \{0,1\},\ q_a^r \in \mathbb{N}_0 \qquad \forall r \in V\ \forall a \in A, \tag{4.7l}$$

$$s_v^r,\ c_v^r \in \{0,1\} \qquad \forall r \in V\ \forall v \in V. \tag{4.7m}$$

The binary variables x define the balanced separator and the variables y define the arc partition. We have the flow variables q ensuring connectivity of the components, and variables s and c to determine the weight of a component.

Constraints (4.7b) partition the arcs and (4.7c) force both directions of an arc to be in the same component. In (4.7d), it is ensured that both endpoints of an arc are in the same component as the arc. Lines (4.7e) and (4.7f) specify a flow to ensure connectivity, and we will discuss this method in more detail in the next chapter. If a vertex is in more than one subgraph, it is deemed part of the separator, which is forced by (4.7g). The upper weight bound of each component is set in (4.7h), while the variable c_v^r indicates if vertex v in the component rooted at r is excluded in the cumulative weight. This is only possible if $s_v^r = 1$ (4.7i) and if v belongs to the separator (4.7j).

We can see that the model contains a large number of variables and constraints, and uses many big M constraints. This and the strong symmetry within the model lead to a weak LP relaxation. Indeed, the dual bound of every single of the 75 instances in our computational study remains zero, even after two hours. Symmetry breaking constraints similar to the ones concerning the vertex-to-root formulation in the previous section, would certainly benefit this model. Also note that for the connectivity, we consider a flow emerging from every vertex, while our main motivation is to avoid this necessity. If we have to solve multiple problems on the same graph (as in the pricing problem of a column generation approach), however, it can be useful to invest the time and determine an optimal set of roots.

4.4.2 Dynamic Cut Formulation

We propose a different exact approach that circumvents the issues of the compact flow model. Our goal is to find a set of vertices that intersects every connected subgraph of cumulative weight greater than $\alpha\, w(G)$. This is essentially a Hitting Set problem where we choose $\mathcal{U} = V$ and $\mathcal{S} = \{V' \subseteq V : G[V']$ is connected and $w(V') > \alpha\, w(G)\}$.

Hitting Set

Instance: A set $\mathcal{U}$ and $\mathcal{S} \subseteq 2^{\mathcal{U}}$.

Problem: Find $X \subseteq \mathcal{U}$ such that $X \cap S \neq \varnothing\ \forall S \in \mathcal{S}$ and $|X|$ is minimum.

Our IP formulation for Minimum Balanced Separator thus mimics the classical formulation for Hitting Set:

$$\min_{x} \quad \sum_{v \in V} x_v \tag{4.8a}$$

$$\text{s.t.} \quad \sum_{v \in S} x_v \ \geq\ 1 \qquad \forall S \in \mathcal{S}, \tag{4.8b}$$

$$x_v \ \in\ \{0,1\} \qquad \forall v \in V. \tag{4.8c}$$

The binary variable x_v indicates if v is part of the balanced separator, and the objective is to minimize its cardinality. Constraints (4.8b) ensure that each subgraph in $\mathcal{S}$ contains at least one separating vertex. As the number of these constraints is potentially huge, we consider those only for a subset $\mathcal{S}' \subseteq \mathcal{S}$ and dynamically add violated constraints within a branch-and-cut approach.

We may start with $\mathcal{S}' = \varnothing$ and call the separation routine only for integer solutions. For such a solution x, we consider the set $X = \{v \in V : x_v = 1\}$. The separation problem is to simply compute the cumulative weight of each connected component in $G - X$, which can be done with a breadth-first search (BFS). If its weight is larger than $\alpha\, w(G)$, we add the vertex set of this subgraph to $\mathcal{S}'$ and iterate.

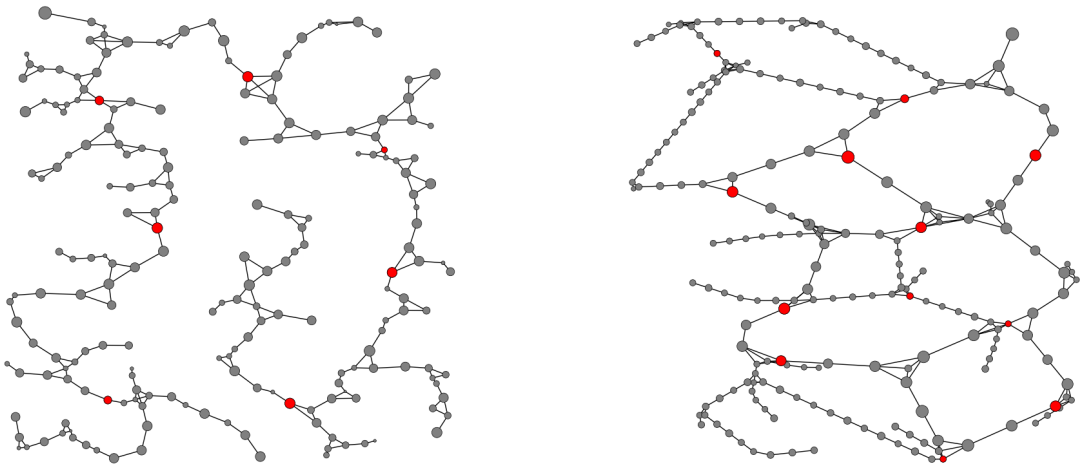
For the implementation, we improve the formulation by adding violating subgraphs of minimal weight to $\mathcal{S}'$. Instead of adding a full component C of $G-X$ with $w(C) > \alpha\, w(G)$, we start a BFS at any vertex in C. As soon as the current subgraph violates the upper bound, we add the respective vertex set to $\mathcal{S}'$ and restart the BFS at an uncovered vertex of C.

4.4.3 Computational Comparison

For a practical comparison of the two approaches, we tested both formulations on test instances that resemble transit networks of different complexity. We consider the same 25 line graphs of trees and 50 line graphs of voronoi networks as for our study in Section 6.4.1. Here, we go not into detail on the construction of these networks but state that the graphs have around 200 vertices and that typical instances together with an optimal solution are depicted in Figure 4.1. Inspired by the application from Chapter 6, we set the parameter α such that $\alpha\, w(G) = 100 - \varepsilon$, where $\varepsilon = 10^{-4}$. This leads to values of α ranging from 0.12 to 0.21.

We ran the experiments on machines equipped with Intel Xeon E3-1234 CPUs with 3.7GHz and 32GB RAM. The code is written in Python 3.6 and to solve MIPs Gurobi 9.1 is used. The time limit is set to two hours. We call formulation (4.7) the *compact formulation* and denote the branch-and-cut approach for (4.8) by B&C.

First, we note that the compact formulation is indeed very weak and suffers from symmetry issues. The dual bound is still at 0 after the time limit of two hours for all 75 instances. In contrast, B&C is able to solve each of the `tree_lg` instances in a fraction of a second to optimality. The performance of B&C on the `voronoi_lg` instances varies.



(a) `tree_lg` (b) `voronoi_lg`

Figure 4.1.: Exemplary instances from our computational study with optimal balanced separator highlighted in red.

While 15 instances are solved within 10 seconds, 31 within 100 seconds, and 45 out of 50 within 1000 seconds, there are two instances that hit the time limit. We observe that for the harder instances, the issues lie on the primal and on the dual side, i.e., Gurobi has problems to find an optimal solution but also to provide a bound to prove optimality.

Concerning the quality of the solutions, the compact formulation is also much worse. The objective value of the incumbent solution after two hours is on average 30% higher than the optimal value for `tree_lg`. With regard to the `voronoi_lg` instances, this gap increases to over 40%. In fact, the compact formulation yields an optimal solution only for one of the 75 instances but is not able to prove optimality as the dual bound remains 0. Once again, this illustrates the superiority of the proposed B&C approach.

5

Finding a Single Optimal Subgraph

This thesis revolves around covering a graph with a number of connected subgraphs. Why, then, is this chapter concerned with finding a single subgraph? In the previous chapter, we have seen that a branch-and-price approach is one way to solve covering problems. An important subproblem in this approach is to identify single promising subgraphs to include into the formulation. This motivates the detailed study of finding single optimal (connected) subgraphs. Indeed, the final chapter of this thesis is devoted to solving a specific districting problem with column generation. Here, we are focused on the resulting pricing problem which can be formulated as an independent problem and is fascinating in its own right.

The most important and well-studied representative for this problem type is the Maximum Weight Connected Subgraph Problem, which is to find a node set of maximum cumulated weight that induces a connected subgraph in a given node-weighted graph.

Maximum Weight Connected Subgraph (MWCS)

Instance: $G = (V, E)$, $p \in \mathbb{R}^V$.

Problem: Find $V' \subseteq V$ such that $G[V']$ is connected and $p(V')$ is maximal.

In the rooted version of MWCS, a given set $R \subseteq V$ has to be a subset of the selected nodes V'. In the capacitated variant, we are given additional weights $w \in \mathbb{R}^V_{\geq 0}$ and numbers $0 \leq W_L \leq W_U$, and demand that $w(V') \in [W_L, W_U]$.

A plethora of applications has driven the research on the MWCS and its variants. This includes applications in oil-drilling [HP94], communication network design [LD98; KLT15], systems biology [Ide+02; Dit+08; Yam+09; Bac+12], environmental conservation [Con+07; DG10], video activity detection [CG12], and forest planning [Car+13].

In this chapter, we discuss the critical task for the MWCS to enforce connectivity within MIPs. In particular, we focus on the case with a single given root node, and state formulations using single-commodity flows, multi-commodity flows, arc separators, and node separators. In Section 5.2, we propose a coarse-to-fine paradigm that splits the connectivity issue into a macro and a micro level. The micro level consists of induced paths with vertices of degree 2 and is modeled independently from the macro level, the coarse graph. In a polyhedral comparison, we show that the most important connectivity formulation on this disaggregated network is tighter than on the original graph.

After this more general overview, we shift our focus to a fairly specific variant of MWCS. We consider a single-rooted and capacitated MWCS with an additional balancing constraint: The nodes of the host graph are colored blue and red, and the chosen subgraph

is required to be balanced regarding its cumulated blue and red weight. We formalize this problem in Section 5.3, review the related literature and present an IP formulation. In Section 5.4, we present powerful reduction techniques for a preprocessing that can drastically reduce the problem size. Further improvements in the form of LP strengthening cuts are studied in Section 5.5. We propose different families of constraints that speed-up the optimization if added to the initial model. An extensive computational study in Section 5.6 compares the connectivity formulations and assesses the impact of our proposed improvements. Finally, we investigate the problem of finding such an optimal subgraph on a tree. In Section 5.7, we show that the simple structure of the graph allows for a simpler IP formulation for connectivity and a dynamic programming approach to solve the problem.

5.1 Enforcing Connectivity in MIPs

In this section, we discuss various ways to model connectivity within a (M)IP. A decent number of papers are dedicated to this problem and the related study of the connected subgraph polytope. These are mostly motivated by the Steiner tree problem and its variants concerning Steiner arborescences, prize-collecting Steiner tree, and maximum weight connected subgraphs. Similar connectivity formulations, apart from the vast literature concerning traveling salesman problems, originate for instance from the generalized minimum spanning tree problem [MLT95; Pop09; Pop20], a connected network design problem [MR05], or the minimum arborescence problem [Duh+08]. Very recently, connectivity in MIPs has also been studied with regard to balanced connected partitions [Miy+21], the connected max-k-cut problem [Hoj+21], and, close to our setting, with regard to political districting [VBL22; VB22]. We have already reviewed these models in Section 4.3.

The first connectivity models for Steiner problems by Aneja [Ane80] and Wong [Won84] date back to the 1980s and propose a "row generation scheme" using minimum cuts, and, respectively, a multi-commodity flow. Since then, other formulations as well as new separation and preprocessing techniques have been developed. A review of different formulations and comparisons of the respective LP relaxations can be found in [GM93; KPH93; MW95; PD01; RFK22]. In addition, computational comparisons of distinct models for variants of the Steiner tree problem are carried out in [DG10; ÁMLM13a; ÁMLM13b; Fis+17]. An excellent survey article by Ljubić [Lju20] covers all relevant topics concerning Steiner trees, including variants, MIP formulations, preprocessing techniques, and applications.

Complementing the variety of IP formulations, the problem has also been studied from a polyhedral perspective. The connected subgraph polytope is the convex hull of all node incidence vectors inducing a connected subgraph. A full description is known when the graph is a tree [KLS91], a cycle [Goe94b], series-parallel [Goe94b], or complete bipartite [Lüt18]. Other important facets and valid inequalities for the mentioned and related polytopes are presented in [CR94; KM98; ÁMLM13a; ÁMLM13b; WBB17]. The edge-induced connected subgraph polytope was studied in [Goe94a; Goe94b; KZ14; BKN15].

Having reviewed the literature for connected subgraphs, we shift our focus to the single-rooted case. For a graph $G = (V, E)$ and a node $r \in V$, an r-tree is a tree of G containing r. Consequently, we study the r-tree polytope, i.e., the convex hull of

$$\mathcal{Y}_r := \{\chi(V') \ : \ V' \subseteq V \text{ is the vertex set of an } r\text{-tree}\}.$$

Note that $conv(\mathcal{Y}_r)$ is, in fact, a facet of the connected subgraph polytope. Goemans [Goe94b] studies the related r-tree polytope $conv(\mathcal{Y}_r^+)$ in the dimension $\{0,1\}^{|E|+|V|}$, i.e., he considers a concatenation of edge and node incidence vectors of r-trees. Clearly, $conv(\mathcal{Y}_r)$ is a projection of $conv(\mathcal{Y}_r^+)$. Since Goemans [Goe94b] gives a full description of $conv(\mathcal{Y}_r^+)$ if G is a cycle or series-parallel, the optimization over $conv(\mathcal{Y}_r)$ is also polynomial in these cases. We note that in [Goe94b], the empty graph is also considered an r-tree to simplify the polyhedral study. Since this edge case is not practically relevant, and for ease of exposition, we exclude the empty graph from $\mathcal{Y}_r$.

The presence of a single root simplifies the (M)IP formulations. Hence, we present four important formulations that describe $\mathcal{Y}_r$. For some of the formulations we consider the bidirected version $D = (V, A)$ of the undirected graph $G = (V, E)$. Throughout the presentation, we use binary y variables to indicate the nodes of the selected subgraph, x variables for describing a flow on A, and binary z variables to model decisions concerning individual arcs.

5.1.1 Single-Commodity Flow (SCF)

A single-commodity flow to ensure connectivity was proposed by Gavish and Graves [GG82] in the context of the traveling salesman problem. Maculan [Mac87] transforms the formulation to the Steiner tree problem. Similar SCF models are used in [LD96; Shi05; Con+07; DG10; Hoj+21; Miy+21; VBL22] and [VB22]. The third and fourth of these papers consider a rooted and budgeted (only with upper bounds) version. Our formulation is adapted for the single-root case. This spares us from introducing an artificial super source.

$$y_r = 1 \tag{5.1a}$$

$$\sum_{a \in \delta^-(v)} x_a - \sum_{a \in \delta^+(v)} x_a = y_v \qquad \forall v \in V \setminus \{r\} \tag{5.1b}$$

$$x_{uv} \leq M_{uv}\, y_v \qquad \forall (u,v) \in A \tag{5.1c}$$

$$x_a \geq 0 \qquad \forall a \in A \tag{5.1d}$$

$$y_v \in \{0,1\} \qquad \forall v \in V \tag{5.1e}$$

The idea is to construct a flow that emerges at the root node and where each node of the chosen subgraph consumes one unit of flow while all other nodes satisfy flow conservation. This is ensured by equalities (5.1b). Inequalities (5.1c) are necessary to activate every node that is used by the flow. The use of a big M parameter is bad for the LP relaxation, but necessary. It should be chosen as small as possible, and $M = |V| - 1$ for each arc (u, v) is an upper bound. Following Validi and Buchanan [VB22], the upper weight bound allows us to improve it to $M = \max_{D \subseteq V \setminus \{r\}} \{|D| : w(D) \leq W_U - w_r\}$, which we can compute with a greedy algorithm. We can further strengthen the formulation by considering arc specific big M values. Consider the arc $(u, v) \in A$ and let ℓ_{ru} be the unit-weight shortest path length from r to u. Since each chosen node consumes one flow unit, the flow value on arc (u, v) is bounded by $M_{uv} = M - \ell_{ru}$. Note that these values can be determined with a single breadth-first search.

5.1.2 Multi-Commodity Flow (MCF)

The first multi-commodity flow formulation for connectivity is due to Beasley [Bea84] and, independently, to Wong [Won84]. Maculan [Mac87] shows that the LP relaxation of MCF is stronger than the SCF relaxation.

$$y_r = 1 \tag{5.2a}$$

$$\sum_{a \in \delta^-(w)} z_a^v - \sum_{a \in \delta^+(w)} z_a^v = 0 \qquad \forall v, w \in V \setminus \{r\}, w \neq v \tag{5.2b}$$

$$\sum_{a \in \delta^-(v)} z_a^v = y_v \qquad \forall v \in V \setminus \{r\} \tag{5.2c}$$

$$\sum_{a \in \delta^+(v)} z_a^v = 0 \qquad \forall v \in V \setminus \{r\} \tag{5.2d}$$

$$z_a^w \leq y_v \qquad \forall v, w \in V \setminus \{r\},\ a \in \delta^-(v) \tag{5.2e}$$

$$z_a^v \geq 0 \qquad \forall a \in A,\ \forall v \in V \setminus \{r\} \tag{5.2f}$$

$$y_v \in \{0, 1\} \qquad \forall v \in V \tag{5.2g}$$

In the MCF formulation, we consider an r-v flow z^v for every node $v \neq r$. Flow conservation is stated in (5.2b), and the flow value is determined by (5.2c) and (5.2d): It is set to 1, if node v is chosen, and otherwise set to 0. Finally, (5.2e) guarantees that a node is chosen if any flow uses it.

The advantage of the MCF formulation is that the arc flow is binary and, hence, we have no big M. This benefits the LP relaxation of the MCF. On the other hand, the number of variables and constraints is much larger than in the SCF formulation.

5.1.3 Rooted Arc Separators (RAS)

A second way to enforce connectivity is with separators. First, we consider a formulation using edge cuts. As our problem is stated on an undirected graph, we can stick with the natural undirected formulation, first proposed by Aneja [Ane80], or consider the corresponding formulation in the bidirected graph. Chopra and Rao [CR94] show that the LP relaxation of the bidirected formulation is tighter than for the undirected case. Goemans and Myung [GM93] further investigate the relation between the two and give an extended undirected relaxation that is equivalent to the bidirected arc cut relaxation. However, we present the arc formulation, which goes back to a Steiner arborescence formulation by Wong [Won84] and is predominantly used in the literature. Similar models are applied in [KM98; Lju+06; Duh+08; DG10; ÁMLM13b; Fis+17; RK19].

$$y_r = 1 \tag{5.3a}$$

$$\sum_{a \in \delta^-(v)} z_a = y_v \qquad \forall v \in V \setminus \{r\} \tag{5.3b}$$

$$\sum_{a \in \delta^-(S)} z_a \geq y_v \qquad \forall v \in S,\ \forall S \subseteq V \setminus \{r\} \tag{5.3c}$$

$$z_a \in \{0, 1\} \qquad \forall a \in A \tag{5.3d}$$

$$y_v \in \{0, 1\} \qquad \forall v \in V \tag{5.3e}$$

If the z variables span an arborescence rooted at r, then the nodes of the arborescence form a connected subgraph in G. Therefore, we have (5.3b) to activate the nodes of the arborescence, and to ensure an in-degree of 1. In addition, we have (5.3c) to ensure that there is a directed r-v-path if v is chosen.

The set $\delta^-(S)$ in (5.3c) is an arc separator for r and v, i.e., there is no r-v-path in $D-S$. Since there is an exponential number of these constraints, the LP relaxation is solved with a cutting plane approach. The crucial ingredient is the separation routine where violated constraints are identified.

Separation: For the separation, we consider a node v and compute a maximum flow from r to v in D with arc capacities z. If the flow value is smaller than y_v, the inequality of any corresponding minimum cut is violated, and we add an appropriate constraint. Hence, the separation can be solved efficiently with at most $n-1$ maximum flow computations.

While this describes the basic procedure, there are a number of tricks to improve the separation. First of all, we have to decide when to cut off fractional solutions of the LP relaxation. While Ljubić et al. [Lju+06] decide to check for a violated inequality for each v with $y_v > 0$, Fischetti et al. [Fis+17] choose a threshold of $y_v \geq 0.5$ for a fractional separation, and Dilkina and Gomes [DG10] separate only for $y_v > 1-\varepsilon$.

As our experiments also suggest to separate for $y_v > 1-\varepsilon$, we present another improvement for this case that showed to have a significant effect: Instead of considering every vertex separately, we compute the connected components of $G - \{v \in V : y_v \leq 1-\varepsilon\}$ and compute a maximum flow only for a single representative of each non-root component. Since the resulting cut is also a minimum cut for each node of this component, this can dramatically reduce the separation time. In particular, we spare the separation for every node of the root component. Also, this benefits situations where a node subset induces a connected subgraph, but the z variables do not form an arborescence, as the example in Figure 5.1 shows: While in this case, the solution is not feasible for (5.3), we can still skip the separation, as our main goal is to find a vertex set that induces a connected subgraph.

Further improvements like back-cuts, nested cuts, and the use of minimum cardinality separators were proposed by Koch and Martin [KM98] and empirically confirmed in [Lju+06] and [Fis+17].

The idea of back-cuts, that was already described by Chopra, Gorres, and Rao [CGR92], is to find a maximum flow (minimum cut) not only in the described network but also in the reversed network where every arc is flipped. As the resulting cuts tend to be different, more cuts are generated and the number of cutting plane iterations empirically decreases. Ljubić et al. [Lju+06] note that both cuts (forward and backward) can be found with one

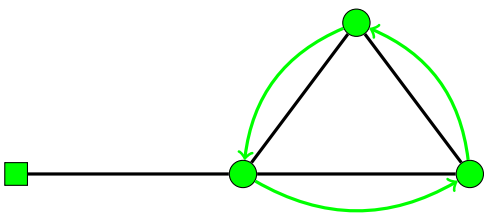


Figure 5.1.: (Infeasible) solution for (5.3) that represents a feasible template: Nodes and arcs in green correspond to variables with value 1 while all other values are 0, the root is depicted as a square.

maximum flow calculation using a specific implementation from [CG95].

Nested cuts are another way to generate more cuts in a single cutting plane iteration. The idea is to set all capacities of already found minimum cut arcs to 1, and to iterate until the flow value is 1. This ensures that arc sets of found violated cuts are disjoint.

The third approach for enhancing the separation is to use minimum cuts of minimum cardinality. Therefore, a small $\varepsilon > 0$ is added to every capacity before computing a maximum flow. While this might lead to increased running times of flow calculations, the use of minimum cardinality cuts can have a great impact on the overall performance. Indeed, the results are split. While [KM98] and [Fis+17] confirm a very positive effect, [Lju+06] report extended running times.

LP Strengthening: Apart from an efficient separation routine, the performance of this approach heavily depends on adding adequate cuts to the initial model to strengthen the LP relaxation. Many of the cuts discussed in the literature are focused on the Steiner tree problem, exploiting the fact that only terminal nodes have to be connected [KM98] or dealing with asymmetry that we do not have [Lju+06]. Here, we will outline two types of cuts from the literature that translate to our setting; see, e.g., [Lju+06; ÁMLM13b]. In Section 5.5, we will present new families of inequalities that help to strengthen the initial model.

Most importantly, we should extend the initial model with so-called 2-cycle inequalities

$$z_{uv} + z_{vu} \leq y_v \qquad \forall (u,v) \in A, \tag{5.4}$$

allowing at most one of each bidirected arc pair to be part of the arborescence. At the same time, these constraints ensure that both end nodes of an arc are active if the arc is chosen. This concept can be generalized to cycles of length k in G. Let C be such a cycle and let $A[C]$ denote all $2k$ bidirected arcs corresponding to C. Then, for each node $u \in C$,

$$\sum_{a \in A[C]} z_a \leq \sum_{v \in C \setminus \{u\}} y_v \tag{5.5}$$

is a valid inequality. There are, indeed, cases where these cuts are useful. For instance, in [ÁMLM13b], all 4-cycle inequalities are added initially if G is a grid graph.

Ljubić et al. [Lju+06] note that for any separator S and for $v \in S$ we can add the equations (5.3b) for nodes in S and subtract the inequality (5.3c) to obtain the valid inequality

$$\sum_{a \in A \cap S^2} z_a \leq \sum_{u \in S \setminus \{v\}} y_u.$$

These *generalized subtour elimination constraints* are already studied by Goemans [Goe94b], and found to be facet-defining under certain circumstances.

5.1.4 Rooted Node Separators (RNS)

The rooted node separators formulation is similar to the previously described RAS, but can be formulated on the original undirected graph and uses only node variables. Such a formulation was first used by Fügenschuh and Fügenschuh [FF08] and later applied in [Bac+12; ÁMLM13b; Car+13; ÁMS17; Fis+17].

Let us denote by $\mathcal{N}(r, v)$ the set of r-v-node separators, i.e., all sets $S \subseteq V$ such that there is no r-v-path in $G - S$.

$$y_r = 1 \tag{5.6a}$$

$$\sum_{w \in S} y_w \geq y_v \qquad \forall v \in V \setminus \{r\},\ \forall S \in \mathcal{N}(r, v) \tag{5.6b}$$

$$y_v \in \{0, 1\} \qquad \forall v \in V \tag{5.6c}$$

The formulation is straight forward. Inequalities (5.6b) ensure that if node v is chosen, then any r-v-node separator must also contain a chosen node.

For the separation routine we describe two possibilities. The first one is in accordance with the RAS separation and described, for instance, in [FF08] and [ÁMLM13b]. A directed auxiliary graph D is created by splitting each node v into an arc (v_{in}, v_{out}) with capacity y_v. Every edge $vw \in E$ is replaced by the arcs (v_{out}, w_{in}) and (w_{out}, v_{in}), each with capacity 1. Now, all procedures from the RAS separation carry over, when comparing the maximum r_{out}-v_{in}-flow value to y_v.

Since the described separation procedure is rather time consuming, Fischetti et al. [Fis+17] propose a different variant that ignores violations of (5.6b) for fractional solutions. Instead, their separation routine is only called when the branching produced an integer solution. A great advantage of this approach is that the authors of [Fis+17] show that minimal separators for integer solutions can be found very efficiently. Since their algorithm is intended for the prize-collecting Steiner tree problem, we give an adaption for the single-rooted connected subgraph problem in Algorithm 5.1. Each connected component and its neighborhood can be computed with an adapted breadth-first search. Lines (8)–(10) handle the back-cuts. The resulting separation sets are pairs of node sets (C, S), and for every $v \in C$, we know that S is an r-v-separator and, therefore, we add the corresponding constraint (5.6b) to the model and iterate.

Algorithm 5.1: `NodeSeparation`

Input: $G = (V, E), r$, integer solution y
Output: separation sets $\mathcal{S}$

1 $\mathcal{S} \leftarrow \varnothing$;
2 $V' \leftarrow \{v \in V : y_v = 1\}$;
3 $C'_r \leftarrow$ connected component of r in $G' := G[V']$;
4 $N(C'_r) \leftarrow \{v \in V \setminus C'_r : \exists u \in C'_r : uv \in E\}$;
5 **foreach** *component* C' *of* G' *with* $r \notin C'$ **do**
6 $\quad C \leftarrow$ component of $G - C'_r$ that contains C';
7 $\quad \mathcal{S} \leftarrow \mathcal{S} \cup (C', N(C'_r) \cap V(C))$;
8 $\quad N(C') \leftarrow \{v \in V \setminus C' : \exists u \in C' : uv \in E\}$;
9 $\quad C \leftarrow$ component of $G - C'$ that contains r;
10 $\quad \mathcal{S} \leftarrow \mathcal{S} \cup (C', N(C') \cap V(C))$;
11 **return** $\mathcal{S}$;

5.2 A Coarse-to-Fine Paradigm

In the uncapacitated MWCS, one can often remove edges or nodes or contract edges to simplify the graph structure [RKM19]. With the capacity constraints and the balancing condition, however, these reductions are not possible anymore.

In this section, we discuss how to improve connectivity formulations for graphs with a large fraction of degree 2 vertices. These graphs usually contain long induced paths where every vertex has degree 2. We exploit the fact that with any chosen vertex of such a path (not containing the root), at least one of the endpoints of the path has to be chosen as well. Following this idea, the coarse network is formed by contracting the specified paths. Now, we model the connectivity on the coarse network and on each of the paths separately. We call this disaggregation the *coarse-to-fine* (C2F) approach.

5.2.1 Construction of Coarse and Fine Graphs

Given is a graph $G = (V, E)$ together with a root $r \in V$ and the vertices of the coarse network $V_c \subseteq V$ with $r \in V_c$. Let $V_{>2} \subseteq V$ be the vertices with degree greater than 2, and we demand $V_{>2} \subseteq V_c$. Note that with this condition, each connected component of the induced subgraph $G[V \setminus V_c]$ is a path. The arcs of the coarse network $D_c = (V_c, A_c)$ are constructed as follows: Whenever there exists a path between two coarse vertices $u, v \in V_c$, only containing vertices from $V \setminus V_c$ (except for u and v), the path is replaced by two coarse arcs (u, v) and (v, u), and the fine nodes on this path are denoted by $P_f(u, v)$. Observe that this construction also replaces an edge between two coarse vertices with two arcs, and hence, the coarse graph can be a multigraph. We will see later that it is sufficient to restrict the connectivity formulation to a certain *core graph* that has no vertices of degree 1, except for the root (cf. Section 5.4). Our implementation, which allows for multiarcs and loops, therefore uses $V_c = V_{>2} \cup \{r\}$. For ease of exposition, however, we assume here that V_c contains all leaves in G and that V_c is chosen in a way such that A_c is a simple set: When $(u, v) \in A_c$ is a multiarc then at least one arc originates from a path that contains vertices from $V \setminus V_c$. By lifting one of these vertices to V_c the resulting coarse network does not contain the corresponding arc anymore.

Recall that the bidirectional graph of G is denoted by $D = (V, A)$. The fine network is then given by $D_f = (V, A_f)$, where $A_f = A \setminus A_c$.

5.2.2 Connectivity in the Fine Graph

We present two possibilities to model the connectivity within each fine path: a flow and a predecessor formulation.

Fine Flow Model:

$$\sum_{a \in \delta^-_{D_f}(i)} x_a - \sum_{a \in \delta^+_{D_f}(i)} x_a = y_i \qquad \forall i \in V \setminus V_c \tag{5.7a}$$

$$x_{uj} \leq M_{uj}\, y_u \qquad \forall (u, j) \in A_f : u \in V_c \tag{5.7b}$$

$$x_{ij} \leq M_{ij}\, y_j \qquad \forall (i, j) \in A_f : j \notin V_c \tag{5.7c}$$

$$y_v \in \{0, 1\} \qquad \forall v \in V \tag{5.7d}$$

$$x_a \geq 0 \qquad \forall a \in A_f \tag{5.7e}$$

Analogously to the SCF formulation, each active vertex consumes one unit of flow, but here, we have one flow per fine path and it can emerge from either of the two coarse end nodes. As for the SCF, (5.7c) activate the head of an arc with flow and (5.7b) activate the coarse nodes from which the flows emerge.

Concerning the big M values, let us consider $(i,j) \in A_f$ and $(u,v) \in A_c$ such that $j \in P_f(u,v)$. By m_{jv} we denote the number of arcs from j to v in $G[P_f(u,v) \cup \{v\}]$ and we assume that $m_{iv} = m_{jv} + 1$, i.e., (i,j) lies on the directed path from u to v that includes j. Then, $M_{ij} = m_{jv}$ is a feasible choice because the fine flow over (i,j) has to carry flow for at most m_{jv} vertices.

Fine Predecessor Model: A different way to model the fine connectivity exploits the simple structure of the fine paths. To this end, we introduce binary arc variables z_a for $a \in A_f$. Now note that every arc $(i,j) \in A_f$ with $i \notin V_c$ has a unique preceding fine arc $prec(i,j) := (h,i) \in A_f$ with $h \neq j$. We can now model the fine flow as follows:

$$\sum_{a \in \delta^-_{D_f}(j)} z_a = y_j \qquad \forall j \in V \setminus V_c \tag{5.8a}$$

$$z_{ij} \leq z_{prec(i,j)} \qquad \forall (i,j) \in A_f : i \notin V_c \tag{5.8b}$$

$$z_{ij} + z_{ji} \leq y_i \qquad \forall (i,j) \in A_f \tag{5.8c}$$

$$z_a \in \{0,1\} \qquad \forall a \in A_f \tag{5.8d}$$

Fine node j is active if and only if one of the two incoming arcs is active. Each active arc then activates the preceding arc, until a coarse vertex is reached. Finally, we ensure that the tail of every active arc is active, but strengthen the constraint to 2-cycle inequalities (5.8c). In particular, this activates the coarse vertex where the active fine path starts.

5.2.3 Connectivity in the Coarse Graph

We can model the connectivity in the coarse graph with any of the presented formulations from Section 5.1 with the advantage that the coarse graph is usually much smaller. In particular, this drastically reduces the big M values for the SCF formulation in sparse graphs which leads to tighter LP relaxations compared to the SCF in the original graph.

C2F-Coupling: Finally, we have to couple the coarse and the fine connectivity which, in this case, means that an active coarse arc has to activate all nodes of the respective fine path. If we have binary arc variables $\hat{z}$ for the coarse arcs, we do this with

$$\hat{z}_{uv} + \hat{z}_{vu} \leq y_i \qquad \forall (u,v) \in A_c \; \forall i \in P_f(u,v). \tag{5.9}$$

Otherwise, if we have a coarse single-commodity flow $\hat{x}$, we use

$$\hat{x}_a \leq M_a \, y_i \qquad \forall a \in A_c \; \forall i \in P_f(a), \tag{5.10}$$

where M_a comes from the SCF formulation on D_c. When we have no arc variables such as in the RNS formulation, we cannot apply the C2F principle. While possible, we also refrain from using it in combination with the MCF formulation.

Proposition 5.1 *With the fine predecessor model and $\hat{z}$ variables on the coarse arcs, the inequalities*

$$\hat{z}_{uv} \leq z_{iv} \qquad \forall (u,v) \in A_c \; \forall i \in P_f(u,v) \cap N(v). \tag{5.11}$$

imply the inequalities (5.9).

Proof. Let $(u,v) \in A_c$, $k \in P_f(u,v)$, and let j_u and j_v be the neighbors of k that are closer to u and, respectively, v. Furthermore, let $i \in P_f(u,v) \cap N(v)$ be the neighbor of v in $P_f(u,v)$. Then,

$$\hat{z}_{uv} \leq z_{iv} \overset{(5.8b)}{\leq} z_{prec(i,v)} \overset{(5.8b)}{\leq} \cdots \overset{(5.8b)}{\leq} z_{j_u k}$$

and, consequently,

$$\hat{z}_{uv} + \hat{z}_{vu} \leq z_{j_u k} + z_{j_v k} \overset{(5.8a)}{=} y_v.$$

□

5.2.4 Polyhedral Study

Let us explore if the C2F model provides any benefit from a polyhedral perspective. Using the fine predecessor model, the RAS and C2F-RAS formulations are very similar. We consider

$$\begin{aligned}
\mathcal{C}^{\text{RAS}} := \{y \in [0,1]^V \mid\ & \exists \hat{z} \in [0,1]^{A_c} \; \exists z \in [0,1]^{A_f} \text{ such that} \\
& (y, \hat{z}) \text{ satisfies (5.3a) – (5.3c), (5.4) on } D_c, \\
& (y, z) \text{ satisfies (5.8a) – (5.8c), and } (\hat{z}, z) \text{ satisfies (5.11)}\}, \\
\mathcal{F}^{\text{RAS}} := \{y \in [0,1]^V \mid\ & \exists z \in [0,1]^A \ : (y,z) \text{ satisfies (5.3a) – (5.3c), (5.4) on } D\}.
\end{aligned}$$

The coarse-to-fine model is characterized by the set of coarse vertices; to avoid ambiguities we denote the polytope corresponding to the LP relaxation by $\mathcal{C}^{\text{RAS}}_{V_c}$.

Proposition 5.2 *Let $G = (V,E)$ be a graph together with a root $r \in V$, and let $V_c \subseteq V$ with $r \in V_c$ and $V_{>2} \subseteq V_c$ be a set of coarse vertices, then*

$$\mathcal{C}^{RAS}_{V_c} \subseteq \mathcal{F}^{RAS}.$$

Proof. Let $y \in \mathcal{C}_{V_c}$ be a feasible solution and let $\hat{z} \in [0,1]^{A_c}$ and $z \in [0,1]^{A_f}$ be corresponding arc values that satisfy the conditions from the definition of $\mathcal{C}^{\text{RAS}}$. If $V_c = V$, we have $D_c = D$ and $A_f = \varnothing$ and, therefore, $(y, \hat{z})$ satisfies (5.3a) – (5.3c), (5.4) on D so that $y \in \mathcal{F}$.

Now we show that if $V_c \subsetneq V$, we can always lift a vertex $i \in V \setminus V_C$ to the coarse network such that $y \in \mathcal{C}_{V_c \cup \{i\}}$. To this end, let $(u,v) \in A_c$ be any coarse arc with $P_f(u,v) \neq \varnothing$ and let $i \in P_f(u,v)$ with $N(i) = \{v, j\}$. We construct the coarse network D'_c for $V'_c = V_c \cup \{i\}$ as described above, i.e., we replace the coarse arcs (u,v) and (v,u) with the arcs $(u,i), (i,u), (v,i)$, and (i,v). Conversely, we lose the fine arcs (v,i) and (i,v), and if $u = j$, also the fine arcs (u,i) and (i,u) in the new fine arc set A'_f. We will proceed to construct vectors $\hat{z}' \in [0,1]^{A'_c}$ and $z' \in [0,1]^{A'_f}$ to certify that $y \in \mathcal{C}_{V_c \cup \{i\}}$.

For the fine network we can simply use $z'_a = z_a$ for $a \in A'_f$, and since we only consider

a subset of the previous constraints, we know that (y, z') satisfies (5.8a) – (5.8c). For the new coarse arcs, we set $\hat{z}'_{ui} = z_{ji}$, $\hat{z}'_{iu} = \hat{z}_{vu}$, $\hat{z}'_{vi} = z_{vi}$, and $\hat{z}'_{iv} = \hat{z}_{uv}$, and for all other arcs $a \in A'_c \cap A_c$, we set $\hat{z}'_a = \hat{z}_a$. Since the root constraint (5.3a) is trivially satisfied, let us check the indegree constraint (5.3b): For the node u it is satisfied because we replaced the arc (v, u) with the arc (i, u) and $\hat{z}'_{iu} = \hat{z}_{vu}$. The same reasoning works for node v and, trivially, (5.3b) holds for all other nodes in $V'_c \setminus \{i\}$. For node i, the equality also holds because

$$\sum_{a \in \delta^-(i)} \hat{z}'_a \;=\; \hat{z}'_{vi} + \hat{z}'_{ui} \;=\; z_{vi} + z_{ji} \overset{(5.8a)}{=} y_i.$$

Concerning the separator constraints (5.3c), observe that each arc separator in D_c that used (u, v) now contains either (u, i) or (i, v). Since $\hat{z}'_{iv} = \hat{z}_{uv}$ and

$$\hat{z}'_{ui} = z_{ji} \overset{(5.8b)}{\geq} z_{iv} \overset{(5.11)}{\geq} \hat{z}_{uv}, \tag{5.12}$$

all separator constraints are still satisfied in D'_c.

The C2F-coupling constraints (5.11) are trivially satisfied by our setting of $\hat{z}'$, so that we are only left with the 2-cycle constraints (5.4) in D'_c. Let us first consider the 2-cycle with nodes u and i; the 2-cycle between v and i then follows immediately. Substituting for $\hat{z}'$ yields $\hat{z}'_{ui} + \hat{z}'_{iu} = z_{ji} + \hat{z}_{vu}$, and we need to show that the latter is less or equal to y_i and y_u, respectively. Analogously to (5.12), we obtain $\hat{z}_{vu} \leq z_{ij}$ and, therefore, we have

$$z_{ji} + \hat{z}_{vu} \;\leq\; z_{ji} + z_{ij} \overset{(5.8c)}{\leq} y_i.$$

For the other case, we assume that k is the neighbor of u in $P_f(u, v)$ and obtain

$$z_{ji} + \hat{z}_{vu} \overset{(5.8b)}{\leq} z_{uk} + \hat{z}_{vu} \overset{(5.11)}{\leq} z_{uk} + z_{ku} \overset{(5.8c)}{\leq} y_u.$$

We conclude that $y \in \mathcal{C}_{V_c \cup \{i\}}$, and we iterate this process until $V_c = V$. □

The preceding result established that the C2F-RAS formulation is at least as tight as the RAS formulation. When we solve the RAS with branch-and-cut, however, not all arc separator constraints are present. In this case, the C2F-RAS formulation can produce better results as the example in Figure 5.2. shows: An activation of the arc (v_2, v_3) implies with the fine predecessor model the activation of (v_1, v_2) and (r, v_1). But also without the predecessor constraints, the activation of v_3 triggers the activation of the coarse arc (r, v_3), and the coupling constraints then enforce $y_{v_1} > 0$.

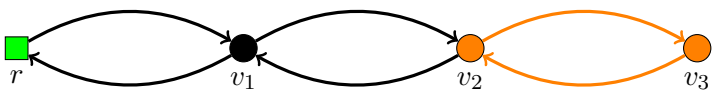


Figure 5.2.: (Infeasible) solution which is feasible for the initial RAS model but not for the initial C2F-RAS with $V_c = \{r, v_3\}$. The y and z variables with positive value are displayed in color: orange represents the value $\frac{1}{2}$ and green the value 1.

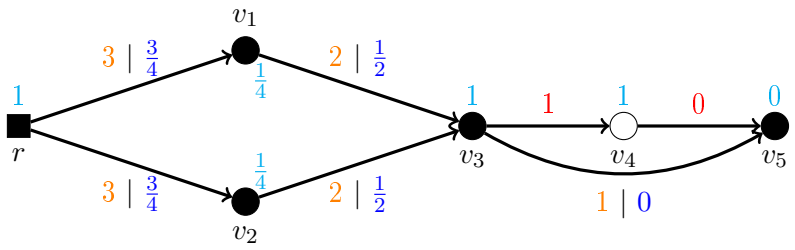


Figure 5.3.: Solution for C2F-SCF with $V_c = \{r, v_1, v_2, v_3, v_5\}$. The y values are colored in cyan, big M values in orange, $\hat{x}$ values in blue, and z values in red.

What about the respective polytopes for the SCF formulation? We would like to prove a similar statement for

$$\begin{aligned}
\mathcal{C}^{\mathrm{SCF}} := \{y \in [0,1]^V \mid\ & \exists \hat{x} \in \mathbb{R}^{A_c}_{\geq 0}\ \exists z \in [0,1]^{A_f} \text{ such that} \\
& (\hat{x}, y) \text{ satisfies (5.1a) – (5.1c) on } D_c, \\
& (y, z) \text{ satisfies (5.8a) – (5.8c), and } (\hat{x}, z) \text{ satisfies (5.10)}\}, \\
\mathcal{F}^{\mathrm{SCF}} := \{y \in [0,1]^V \mid\ & \exists x \in \mathbb{R}^{A}_{\geq 0}\ : (x, y) \text{ satisfies (5.1a) – (5.1c) on } D\}.
\end{aligned}$$

However, this is not possible as the following example shows. Let us consider the graph depicted in Figure 5.3 and assume unit-weights and $W_U = 4$. One can check that the big M values are set as specified and that the depicted coarse flow and the fine path values are indeed feasible, i.e., $y \in \mathcal{C}^{\mathrm{SCF}}$. In particular, note that the coupling constraints (5.1c) for (r, v_1) and (r, v_2) are tight.

If we lift the vertex v_4 to the coarse network, however, the big M values on (r, v_1) and (r, v_2) do not change, i.e., the flow values on these arcs cannot increase. On the other hand, the coarse flow now has to carry the flow for v_4 and, thus, an additional flow value of 1 has to leave the root node. As this is not possible, we conclude that there is no $x \in \mathbb{R}^{A}_{\geq 0}$ such that (x, y) satisfies (5.1a) – (5.1c) for the depicted vector y and, hence, $y \notin \mathcal{F}^{\mathrm{SCF}}$.

Our counterexample shows that the big M values are too tight for the theoretic result that we aimed for. One can adjust the setting and use weaker big M values to obtain a model for which the transformation of a solution is possible. Indeed, we followed this path, and describe a different C2F-SCF formulation in [BSS23]. There, we prove that for this formulation, the C2F-SCF formulation is tighter than the respective SCF formulation. We refrain from repeating the formulations and the corresponding polyhedral study here, and refer the interested reader to [BSS23].

5.3 Rooted MWCS with Balancing and Capacity Constraints

After the detailed discussion of connectivity in IPs, it is time to formally introduce the main problem of this chapter. It is essentially a combination of two known optimization problems: The MWCS in the single-rooted and capacitated version, and the Balanced Connected Subgraph Problem (BCS). For the BCS, we are given a graph $G = (V_b \dot{\cup} V_r, E)$ with nodes colored either blue or red, and seek a maximum-cardinality subgraph that contains an equal number of blue and red nodes.

Now we are ready for the main problem:

Balanced, Rooted, and Capacitated MWCS (BRCMWCS)

Instance: A graph $G = (V, E)$ with $V = V_b \dot\cup V_r$, node weights $w \in \mathbb{R}^V_{\geq 0}$ and $p \in \mathbb{R}^V$, as well as numbers $0 \leq W_L \leq W_U$, $\Delta \geq 0$, and a root node $r \in V$.

Problem: Find $V' \subseteq V$ such that $G[V']$ is connected with $w(V') \in [W_L, W_U]$, $r \in V'$, and $|w(V' \cap V_b) - w(V' \cap V_r)| \leq \Delta$ while maximizing $p(V')$.

In other words, the BRCMWCS asks for a maximum-profit connected subgraph that is built from a root r, respects a lower and upper weight bound, and contains a "similar mass" of red and blue nodes. Figure 5.4 depicts an exemplary instance of this problem.

Complexity: The BRCMWCS is NP-hard and the reduction can come from multiple angles. For instance, we know that the rooted MWCS is NP-hard due to [ÁMLM13b]. By setting $W_L = 0$, $W_U = w(V)$, and $\Delta = w(V)$, we basically drop the capacity and balancing constraint and the rooted MWCS reduces to the respective BRCMWCS with arbitrary node colorings.

On the other hand, since the Balanced Connected Subgraph Problem is NP-hard (see [Bho+19a]), a reduction from this direction is also possible. With unit weights and unit profits, $W_L = 0$, $W_U = |V|$, and $\Delta = 0$, we can solve the respective BRCMWCS once for every node representing the root, and thereby solve the BCS.

Finally, the capacity constraints alone make the problem NP-hard. A reduction from the number partition problem is possible by considering a star graph where the leaf weights are the numbers of the partition instance and the center, which we use as the root, has weight 0. By setting $W_L = W_U = \frac{w(V)}{2}$ and $\Delta = w(V)$ for arbitrary node colorings, we see that the resulting BRCMWCS (with arbitrary profits) has a feasible solution if and only if the partition problem has a feasible solution.

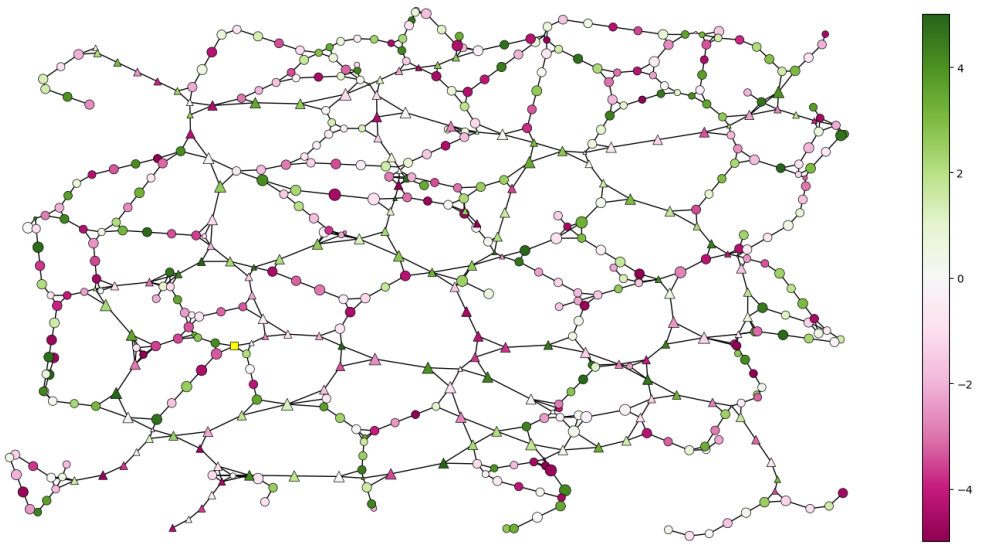


Figure 5.4.: Exemplary BRCMWCS instance. The root node is depicted as a yellow square, circles represent nodes in V_b and triangles nodes in V_r, node sizes correspond to weights and colors to node profits.

5.3.1 Related Work

The Balanced Connected Subgraph Problem was introduced in [Bho+19a] and shown to be NP-hard, even on planar, bipartite, and chordal graph, or when a single root node is specified. When the graph is a tree, however, Bhore et al. [Bho+19a] give a labeling algorithm to solve the BCS in time $\mathcal{O}(|V|^4)$. Kobayashi et al. [Kob+19] improve the runtime to $\mathcal{O}(|V|^2)$ by running a dynamic program on a transformed rooted binary tree with possibly additional uncolored nodes. The authors also briefly study a weighted version of BCS and give complexity results for special graph classes. Further complexity and inapproximability results as well as polynomial-time algorithms for the BCS on other special graph classes are provided in [Bho+19a; Bho+19b; Dar+19; Mar+21].

Shifting to relevant variants of MWCS, the capacitated version with a lower bound has been considered in [HP94; LD96; LD98]. However, the nodes bear unit weights and $W_L = W_U = k$, i.e., the goal is a maximum weight connected subgraph with exactly k nodes. While Hochbaum and Pathria [HP94] propose a dynamic program that finds the optimal solution on trees and a $\frac{1}{k}$-approximation on general graphs, the authors of [LD98] reduce the problem to the single-rooted case which is heuristically solved in [LD96].

The rooted and capacitated MWCS has been studied in [DG10; ÁMLM13b], but only with an upper weight bound, i.e., $W_L = 0$. Dilkina and Gomes [DG10] compare three connectivity models: A single-commodity flow, a multi-commodity flow, and a Steiner arborescence (SA) formulation that is similar to our RAS. On 100 synthetic 10×10 grid instances with 3 roots, the computational comparison shows that the SCF LP relaxation is fastest but provides the worst integrality gap. The SA LP relaxation is also relatively fast and gives the best gap. The MCF LP relaxation is the slowest of the three models and the gap is between the SCF and SA relaxations. With respect to optimal integer solutions, the results in [DG10] indicate that SCF is best if the upper weight bound W_U is so large that it can almost be ignored. In the other case, however, SA performs better on the considered instances.

Álvarez-Miranda, Ljubić, and Mutzel [ÁMLM13b] refrain from including the SCF formulation into their computational comparison, and only evaluate the SA formulation against a node separator (NS) formulation. They find that the performance of the two formulations is complementary, and depends on the instance. It seems as the NS formulation with fewer variables performs better on dense graphs, whereas the SA formulation seems better suited for sparser graphs.

Computational studies show that preprocessing methods and primal heuristics generally have a huge impact when considering the general MWCS and its relatives. Reduction tests can prove that certain nodes or edges must belong to every solution, or that they cannot be part of any solution. This helps to drastically reduce the problem size. A number of techniques for different Steiner problems are proposed in [CGR92; KM98; PD01; CCL06; Lju+06; GVHS08; EKK14; Lei+18; RK19; RKM19]. Unfortunately, these algorithms are highly problem specific, and most of the approaches do not translate to the BRCMWCS, since they conflict with our capacity constraints $W_L \leq w(V') \leq W_U$ or make use of Steiner terminals that we do not have. Concerning the Balanced Connected Subgraph Problem, we are not aware of any preprocessing approaches at all.

5.3.2 IP Formulation for BRCMWCS

We use binary y variables to indicate the chosen vertices, and we model the rooted connectivity with the abstract constraint $y \in \mathcal{Y}_r$. The IP formulation of the BRCMWCS is straightforward and can be stated as follows.

$$\max_{y} \quad \sum_{v \in V} p_v \, y_v \tag{5.13a}$$

$$\text{s.t.} \quad W_L \leq \sum_{v \in V} w_v \, y_v \leq W_U \tag{5.13b}$$

$$\sum_{v \in V_r} w_v \, y_v - \sum_{v \in V_b} w_v \, y_v \leq \Delta \tag{5.13c}$$

$$\sum_{v \in V_b} w_v \, y_v - \sum_{v \in V_r} w_v \, y_v \leq \Delta \tag{5.13d}$$

$$y \in \mathcal{Y}_r \tag{5.13e}$$

The objective (5.13a) is to maximize the profit of the chosen subgraph. The capacity constraints are specified in (5.13b), and the balancing constraints in (5.13c) and (5.13d).

5.4 Reduction Techniques for BRCMWCS

When we consider an instance of BRCMWCS, we might be able to eliminate certain parts of the network, or even prove infeasibility in a preprocessing phase. Computational studies show that preprocessing methods for the MWCS generally have a huge impact on the solution time [EKK14; RK19; RKM19]. While the general methods for the MWCS do not carry over to the BRCMWCS, we propose a number of effective approaches that significantly reduce the problem size and computation times for our problem.

5.4.1 Basic Reduction Techniques

We start with some basic reduction techniques, but stress that the word "basic" relates to the mathematical depth of the approaches and not to their impact. In fact, for many instances, these reductions account for a large portion of the effect that is achieved by the preprocessing.

For a straightforward presentation, we transform the node weights (w_v) in G to arc weights in the bidirected version $D = (V, A)$. Therefore, for $(u, v) \in A$ we define

$$\begin{aligned}
\tilde{w}_{(u,v)} &:= w_v, \\
\tilde{w}^b_{(u,v)} &:= \begin{cases} w_v & , \text{ if } v \in V_b, \\ 0 & , \text{ else}, \end{cases} \\
\tilde{w}^r_{(u,v)} &:= \begin{cases} w_v & , \text{ if } v \in V_r, \\ 0 & , \text{ else}. \end{cases}
\end{aligned} \tag{5.14}$$

W_U-Radius: A simple and yet effective method is to exclude all vertices that are not within distance W_U to r w.r.t. the vertex weights w. As it is impossible to reach these within the weight bound of W_U, they cannot be part of any feasible solution. We can determine these nodes with a single call of Dijkstra's algorithm on D with arc weights $\tilde{w}$ starting at r.

Color Radii: A natural extension of this idea is to consider the node weights in combination with the node colors. If we combine the capacity constraints with the balancing condition, we can derive capacity bounds for each color class.

Lemma 5.3 *Given a BRCMWCS instance $(G, w, c, W_L, W_U, \Delta, r)$, let*

$$W_L^b := \frac{W_L - \Delta}{2}, \qquad W_U^b := \min\left(\frac{W_U + \Delta}{2},\, w(V_r) + \Delta,\, w(V_b)\right),$$
$$W_L^r := \frac{W_L - \Delta}{2}, \qquad W_U^r := \min\left(\frac{W_U + \Delta}{2},\, w(V_b) + \Delta,\, w(V_r)\right).$$

Then, any feasible solution $T = (V_T, E_T)$ satisfies

$$w(V_T \cap V_b) \in [W_L^b,\, W_U^b] \qquad \text{and} \qquad w(V_T \cap V_r) \in [W_L^r,\, W_U^r].$$

Proof. The bounds are fairly easy to deduce. Exemplarily, we prove the lower weight bound for the set of chosen "blue" vertices, i.e., $w(V_T \cap V_b) \geq \frac{W_L - \Delta}{2}$. For brevity, we write $w^b = w(V_T \cap V_b)$ and use w^r accordingly. If $w^r < w^b$, we can prove this bound for w^r and it would follow for w^b. Thus, we assume that $w^r \geq w^b$ and, then, we have $w^b + w^r = w(V_T)$ and $w^r - w^b \leq \Delta$. Consequently,

$$w^b \;\geq\; w^r - \Delta \;=\; w(V_T) - w^b - \Delta \;\geq\; W_L - w^b - \Delta,$$

and the statement follows immediately. □

Ignoring the lower bounds for the moment, we can define the color radius for each color class as the set of nodes that can be reached from the root on a path that satisfies the upper weight bound of the according color. Analogously to the W_U-radius, we can determine this set with a single shortest path tree computation in D with arc weights $\tilde{w}^b$ and $\tilde{w}^r$, respectively. Every node that is outside of either color radius cannot be part of any feasible solution and can be removed.

Restricting Connectivity Formulations to the Core Graph: A clever idea to reduce the model size is to concentrate the connectivity issue to a *core graph* by handling "outer" parts of the network separately. More specifically, if $v \neq r$ is a leaf in G with unique neighbor u, we can add the inequality $y_v \leq y_u$ to our model and remove v from the graph that is used for the connectivity formulation. By successively removing nodes of degree 1 (except for the root node) we obtain the core graph.

For very sparse, tree-like instances, which we also consider here, the restriction to the core graph proves to have an enormous impact. Note that if the graph is a tree, then the core graph consists of only one node, the root. In this case, the added inequalities indeed suffice to model the connectivity condition as we will see in Section 5.7.

Algorithm 5.2: `ComputeBicolorLabels`

Input: $G = (V_b \dot\cup V_r,\ E),\ w,\ r,\ \ell_{\max}$
Output: set of labels per node in G

Consider arc weights ω in the bidirected version of G with $\omega_{(u,v)} \leftarrow \left(\tilde{w}^b_{(u,v)},\ \tilde{w}^r_{(u,v)}\right)$;
$\ell_r \leftarrow \begin{cases}(w_v, 0) & \text{, if } r \in V_b,\\ (0, w_v) & \text{, else}\end{cases}$; $labels[v] \leftarrow \begin{cases}\{\ell_r\} & \text{, if } v = r,\\ \varnothing & \text{, else}\end{cases}$ for $v \in V$;
$L \leftarrow \{(r, \ell_r)\}$;
repeat
  $L' \leftarrow \varnothing$;
  for $(u, \ell_u) \in L$ **do**
    for *each neighbor* v *of* u *in* G **do**
      $\ell' \leftarrow \ell_u + \omega_{(u,v)}$;
      if ℓ' *respects* $\ell_{\max}$ *and is not dominated by any label at* v **then**
        Add ℓ' to $labels[v]$;
        Add (v, ℓ') to L';
        Remove dominated labels in $labels[v]$;
  $L \leftarrow L'$;
until $L = \varnothing$;
return *labels*

5.4.2 The Bicolor Radius

Instead of computing the single color radii, we can also consider the *bicolor radius*, i.e., the set of nodes that can be reached from the root on a path that satisfies the upper weight bound of both color classes.

Unfortunately, the bicolor radius cannot be computed via a shortest path tree. In fact, it is essentially a constrained shortest path problem to determine if a specified node is inside the bicolor radius. We solve the problem with a Bellman-Ford-like labeling algorithm that determines Pareto-optimal shortest path costs. The approach is detailed in Algorithm 5.2 and uses two-dimensional arc weights and node labels (for the blue and red cumulated weight, respectively). The upper weight bounds, given as a pair $\ell_{\max} = (W^b_U, W^r_U)$, are part of the input. We use the usual notion of domination, i.e., label ℓ_1 dominates label ℓ_2 if each weight in ℓ_1 is less than or equal to the corresponding weight in ℓ_2 and if at least one of the inequalities is strict. The algorithm is closely related to constrained shortest path labeling approaches and runs in pseudo-polynomial time. A node v is in the bicolor radius if and only if $labels[v] \neq \varnothing$ where *labels* is the output of Algorithm 5.2.

The effect of the bicolor radius preprocessing can be substantial, and goes far beyond the single color radii. For the instance depicted in Figure 5.5, the single color radii cannot exclude a single node. The bicolor radius, on the other hand, is able to eliminate all gray nodes, essentially eliminating half of the graph.

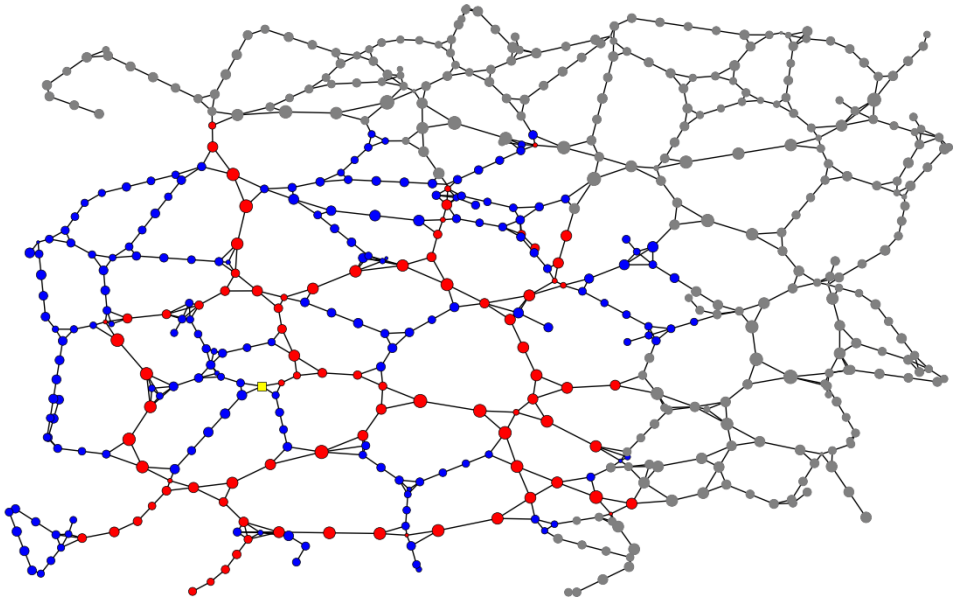


Figure 5.5.: Effect of the bicolor radius preprocessing. The root node is depicted as a yellow square, the gray nodes are outside of the bicolor radius and can be removed.

5.4.3 Preprocessing

The bicolor radius is an effective method to remove nodes from the graph that cannot be part of any feasible template. Complementary to this approach, we can also fix nodes that have to be part of any feasible template.

In order to fix a node v, we consider the connected component C of r in $G - v$. If C violates the lower weight bound of any color class, this component cannot contain a feasible template and, thus, we can fix node v to be part of any solution. The routine described in Algorithm 5.3 strictly follows this principle. In our implementation, however, we precompute the articulation points of G and consider the nodes in a depth-first search (DFS) order. Therefore, if v is an articulation point we can efficiently deduce the color weights of the root component from the parent node. If, in addition, the root component of $G - v$ is feasible, we do not have to consider any descendants of v in the DFS tree.

Since the weight bounds and the network may change when we remove nodes, and the set of fixed nodes can impact the removal, it makes sense to repeat the procedures until a stable

Algorithm 5.3: `FixNodes`

Input: $G = (V_b \dot\cup V_r,\ E), w,\ r,\ W_L^b,\ W_L^r$
Output: set V^f of fixed nodes

```
1 V^f ← ∅;
2 for v ∈ V(G) \ r do
3     C ← connected component of r in G − v;
4     if w(V^b ∩ V(C)) < W_L^b or w(V^r ∩ V(C)) < W_L^r then
5         V^f ← V^f ∪ {v};
6 return V^f;
```

Algorithm 5.4: `Preprocessing`

Input: $G = (V_b \dot\cup V_r,\ E),\ w,\ r,\ W_L^b,\ W_L^r,\ W_U^b,\ W_U^r$
Output: relevant subgraph $G' \subseteq G$,
set V^f of fixed nodes

1 $V^f \leftarrow \{r\}$
2 **repeat**
3 Define node weights w' with $w'_v \leftarrow \begin{cases} w_v & , \text{ if } v \notin V^f, \\ 0 & , \text{ else} \end{cases}$ for $v \in V$
4 $\ell'_{\max} \leftarrow \big(W_U^b - w(V^f \cap V_b),\ W_U^r - w(V^f \cap V_r)\big)$
5 $labels \leftarrow$ `ComputeBicolorLabels`$(G, w', r, \ell'_{\max})$
6 **repeat**
7 Remove every node v with $labels[v] = \varnothing$ from G
8 Recompute $W_L^b, W_L^r, W_U^b, W_U^r, \ell'_{\max}$
9 Remove all labels in *labels* that do not dominate $\ell'_{\max}$
10 **until** *no nodes were removed*;
11 **if** $w(V_b) < W_L^b$ *or* $w(V_r) < W_L^r$ *or* $V^f \not\subseteq V(G)$ **then**
12 **return** $(\varnothing, \varnothing)$
13 $V^f \leftarrow$ `FixNodes`(G, w, r, W_L^b, W_L^r)
14 **until** *no additional nodes were fixed*;
15 **return** (G, V^f)

state is reached. To this end, we propose Algorithm 5.4 that presents our preprocessing routine. The input for the weight bounds stems from Lemma 5.3.

Note that we ignore the weight of all fixed nodes (cf. line 3) for the computation of the bicolor labels. In return, we subtract the weight of fixed nodes from the upper weight bounds (cf. line 4). This can lead to a tighter bicolor radius and remove even more nodes.

The update of the weight bounds in line 8 allows us to repeatedly remove nodes without the more expensive recomputing of all bicolor labels.

We can also quickly identify infeasibility, as it is done in line 11: Whenever the remaining graph does not meet a lower weight bound or when we removed a fixed node, we can deduce that there is no balanced connected subgraph that meets the capacity constraints.

Finally, in order to improve our formulation, we can add the valid inequalities from Lemma 5.3 to our respective MIP, i.e.,

$$W_L^b \le \sum_{v \in V^b} w_v\, y_v \le W_U^b, \qquad W_L^r \le \sum_{v \in V^r} w_v\, y_v \le W_U^r. \tag{5.15}$$

Let us consider the functioning of Algorithm 5.4 on a specific example depicted in Figure 5.6. The table contains key values at different stages of the algorithm, namely, after the execution of the line that is specified in the respective column. We report the total bicolor weight $\tilde w(V^f)$ of the fixed nodes, the cumulated bicolor weight $\omega(V)$ of remaining unfixed nodes, as well as the current lower and upper weight bound reduced by the fixed weight, Ω_L and Ω_U.

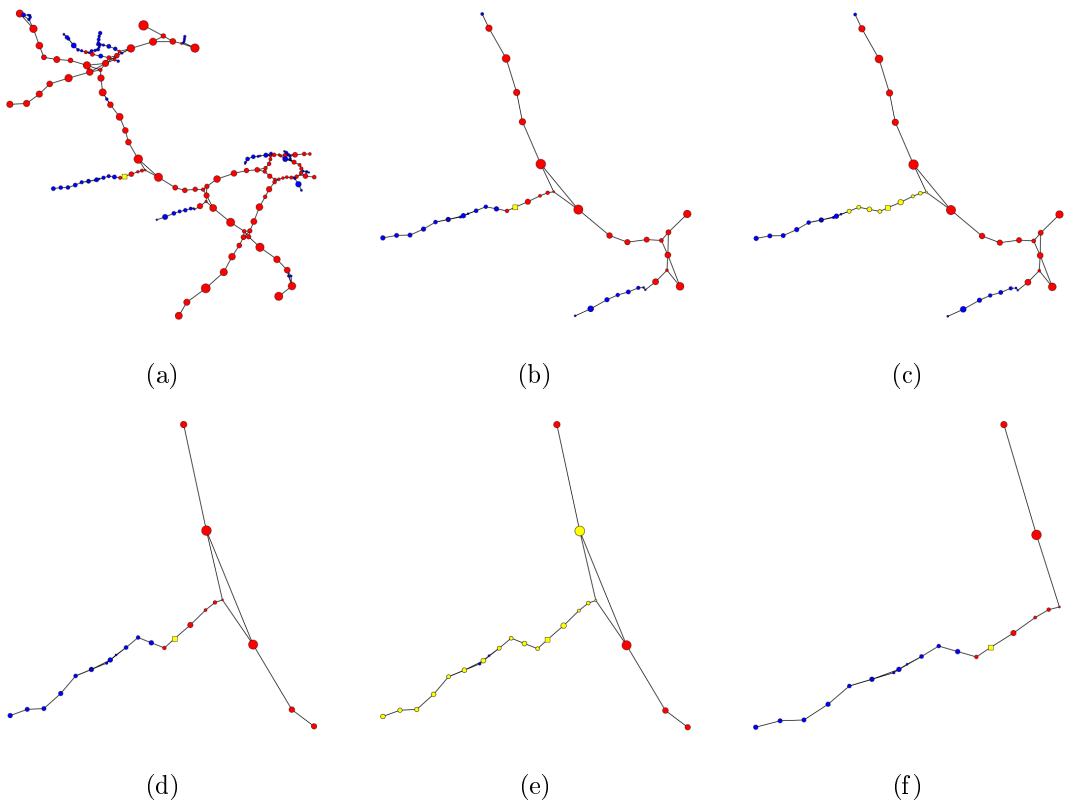


image	line	$\tilde{w}(V^f)$	$\omega(V)$	Ω_L	Ω_U
(a)	1	(0.0, 0.0)	(228.7, 876.5)	(47.4, 47.4)	(100.0, 100.0)
(b)	10	(0.0, 0.0)	(78.6, 178.6)	(47.4, 47.4)	(78.6, 81.2)
(c)	13	(12.4, 16.4)	(66.2, 162.2)	(35.0, 31.0)	(66.2, 64.8)
(d)	10	(12.4, 16.4)	(37.7, 65.4)	(35.0, 31.0)	(37.7, 36.6)
(e)	13	(45.0, 37.8)	(5.1, 44.0)	(2.4, 9.6)	(5.1, 14.9)
(f)	10	(45.0, 37.8)	(5.1, 9.8)	(2.4, 9.6)	(5.1, 0.0)
–	13	(45.0, 47.6)	(5.1, 0.0)	(2.4, 0.0)	(5.1, 0.0)

Figure 5.6.: Key steps of the preprocessing in Algorithm 5.4 for an exemplary instance, portrayed as images and within a table. The root node and fixed nodes are colored yellow. The columns of the table are specified in the text.

We see that the first discarding step results in a massive reduction of the network size. Then, it is possible to fix a certain subset of nodes while this, in turn, lowers the upper weight bound Ω_U (and sets the weight of fixed vertices to 0). With the new bounds, it is possible to discard even more nodes, and on the resulting graph, we find new nodes that have to be part of any solution. After this fixing and a next discarding step, we perform a final node fixing round. As one can see in the last line of the table, the cumulative weight of all remaining unfixed red nodes is 0.0, i.e., all red nodes of the remaining graph have to be included in any feasible solution. Starting with a graph on 161 vertices, the preprocessing was able to discard 140 of these nodes and fix 18 of the remaining 21 vertices. While this

is indeed an extreme example, we see that such an effect is not unusual for the pricing instances considered in the next chapter.

5.5 LP Strengthening

With removing dispensable nodes and fixing necessary ones, we often reduce the problem size significantly. However, we can gain even more valuable insight before starting the optimization. In the following, we will explore a number of possibilities to incorporate additional information into the initial MIP formulation to strengthen its LP relaxation.

The overall effect of our strengthening methods is immense and Figure 5.7 shows their impact. Depicted are the optimal solutions to the LP relaxations of SCF and the initial RAS model, i.e., (5.3) without (5.3c) but with 2-cycle inequalities (5.4), once without the strengthening cuts proposed in the following and once with their inclusion. All nodes with y value 0 are colored gray. We see that the LP relaxation of the basic SCF is very weak and that for the RAS, the solution consists of several connected components that are far apart. Both solutions look much better when the strengthening cuts are added. In fact, the vertices with positive y values in the RAS solution now induce a connected graph. We can also evaluate the improvement with respect to the objective value. While the optimal objective value of the IP is 11.6, the basic LP relaxations achieve 24.4 (SCF) and 23.0 (RAS). The strengthened versions, on the other hand, have optimal objective values of 14.6 (SCF) and 14.7 (RAS). So let us discover which cuts cause this great improvement.

5.5.1 Implied Nodes

The first option that we consider is the fixing of nodes, given that a certain other node is included in a solution. More specifically, our goal is to identify vertices $u \neq v$ such that for every feasible subgraph T we have

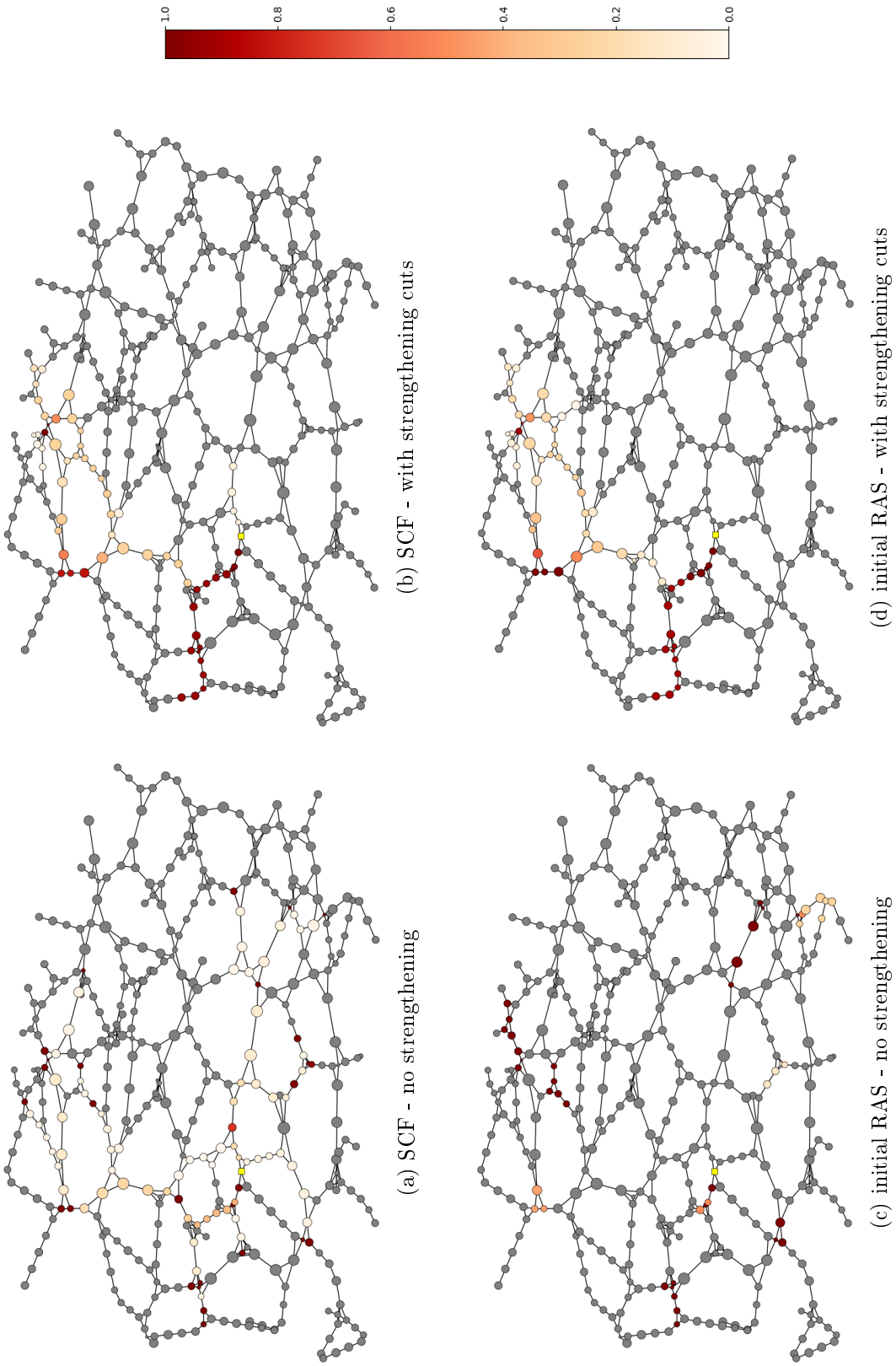
$$u \in T \implies v \in T.$$

These implications can be integrated into the MIP model with the simple constraint $y_u \leq y_v$. We have already seen these constraints when building the core graph, and indeed, a leaf (unless it is the root) and its neighbor are a special case of implied nodes.

Articulation Points: A simple method to find such implications is to determine the articulation points of G, i.e., the set of all vertices v such that $G - v$ is not connected anymore. It is well known that the articulation points of G can be found with a modified DFS in time $\mathcal{O}(|E|)$ [HT73]. If v is an articulation point and u is not in the root component of $G - v$, then the inclusion of v is clearly necessary for the inclusion of u in any feasible template, and the above inequality is valid. Lüthen [Lüt18] and Hojny et al. [Hoj+21] also follow this direction but find that there are too many cuts to add. We concentrate on a subset of these that proves to be very useful: the neighbors of articulation points. In particular, let $\{v\}$ be an r-u-separator with $uv \in E$, then we call the valid inequality $y_u \leq y_v$ an *Articulation Point Neighborhood Cut*.

General Case: Again, the color classes and capacity constraints allow for more implications. To this end, we consider the contraposition of the statement, i.e., $v \notin T \implies u \notin T$. Our goal is to run Algorithm 5.2 on $G-v$ to find vertices u that cannot be reached anymore.

We can do this more efficiently, if we store each label ℓ together with the Pareto-optimal



(a) SCF - no strengthening

(b) SCF - with strengthening cuts

(c) initial RAS - no strengthening

(d) initial RAS - with strengthening cuts

Figure 5.7.: Solutions to LP relaxations of the SCF and initial RAS model with and without strengthening constraints.

path P_ℓ that led to it in the initial run of Algorithm 5.2. For $x \in V \setminus \{r\}$ let

$$P(x) := \bigcup_{(\ell,P_\ell)\in \mathit{labels}[x]} V(P_\ell)$$

be the set of nodes that are on any considered Pareto-optimal r-x-path. Now let us consider a specific vertex v and remove it from G. The set $A_v := \{w \in V : v \in P(w)\}$ consists of all vertices for which we have to delete at least one label. If after the removal of all labels (ℓ, P_ℓ) with $v \in P_\ell$, every vertex still has at least one label, there are no implications to be drawn from $v \notin T$. In the other case, we start our algorithm `ComputeBicolorLabels` on $G - v$ in a slightly adjusted variant: Apparently, all labels of nodes in $V \setminus A_v$ do not change, and the nodes $S := (V \setminus A_v) \cap N(A_v)$ are the nodes that do not change any label, but are adjacent to vertices that are affected. Therefore, we define the start labels as $L := \{(w, \ell) : w \in S, \ell \in \mathit{labels}[w]\}$ and use this set in line 3 of Algorithm 5.2. With this adjustment, we ensure that the search for new labels is steered towards the relevant part of the graph. Every node u that has no label after the `ComputeBicolorLabels` run on $G - v$ can only be part of any feasible solution if v is also part of the template.

In order to reduce the number of implications, we have two strategies. First, we consider only coarse nodes as necessary nodes, i.e., we consider $G-v$ only for v with $\deg(v) \geq 3$. And second, we filter implied nodes by removing transitivity redundancies, i.e., with $u \implies v$ and $v \implies w$, we can drop the implication $u \implies w$.

Activation Pairs: For the sake of completeness, we also briefly cover the implication $u \notin T \implies v \in T$. It means that at least one of the two nodes u and v has to be part of any solution, and the corresponding cover constraint that can strengthen our MIP model is $y_u + y_v \geq 1$. We can find such a node pair by running Algorithm 5.3 on $G - v$ for each node $v \neq r$. The performance can be improved if we consider the nodes v in a DFS order as we did for the node fixing and prune early. Our computational tests revealed, however, that these inequalities do not help in reducing the computation time.

5.5.2 Conflicts

Besides implied nodes ($u \in T \implies v \in T$) and activation pairs ($u \notin T \implies v \in T$), there is a third implication which we study here:

$$u \in T \implies v \notin T$$

for every feasible template T. It states that the two nodes u and v cannot be both part of a feasible solution. In other words, they form a conflict pair and the inequality to potentially tighten the LP relaxation of our formulation is $y_u + y_v \leq 1$.

The idea of analyzing conflicting pairs (or even larger sets), and deriving stronger constraints has been proven to be very useful in different set packing problems, such as knapsack or matching problems. The proposed idea also translates to the rooted and capacitated MWCS, i.e., without the colors and balancing condition.

Finding Conflict Pairs: In order to identify conflict pairs, we can once more make use of the upper capacity bounds from Lemma 5.3. If for two nodes u and v and for every tree $T = (V_T, E_T)$ containing r, u, and v, we know that $w(V_T) > W_U$ or $w(V_T \cap V_b) > W_U^b$ or $w(V_T \cap V_r) > W_U^r$, then (u, v) is a conflict pair.

Algorithm 5.5: `SteinerTreeConflictPairs`

Input: $D = (V,\ A),\ r,\ \tilde{w} \in \mathbb{R}^A_{\geq 0},\ w_{\max}$
Output: set C of conflict pairs

```
1  C ← ∅;
2  len ← all pairs shortest path lengths in D w.r.t. w̃;
3  for all node pairs u, v ∈ V \ {r} do
4      for c ∈ V do
5          weight ← len[(r, c)] + len[(c, u)] + len[(c, v)];
6          if weight ≤ w_max then
7              // there is a feasible Steiner tree
8              Go to line 3 and check the next node pair;
9      Add (u, v) to C;
10 return C;
```

Deciding if there exists a tree connecting a given set of terminal vertices at some cost, is a Steiner tree problem. Fortunately, an elegant combinatorial approach is known for the Steiner tree problem with three terminals. It is based on the insight that any Steiner tree with three terminals is a union of paths from a common center node (that is possibly a terminal itself) to the terminals.

Building on this, Algorithm 5.5 describes our procedure to identify conflict pairs. As input, we use the bidirected version D of G, the root node r, and one of the following three weight combinations:

- $\tilde{w}, \quad w_{\max} = W_U - w_r,$
- $\tilde{w}^b, \quad w_{\max} = W^b_U - w_r \cdot \chi(r \in V_b),$
- $\tilde{w}^r, \quad w_{\max} = W^r_U - w_r \cdot \chi(r \in V_r),$

with arc weights as defined in (5.14) and χ denoting the characteristic function. The union of the three respective return values of Algorithm 5.5 constitutes our set of conflict pairs.

The proposed algorithm runs in time $\mathcal{O}(|V|^3)$, but there is some room for improvement, especially for very sparse graphs. First of all, it is not necessary to consider every node in V as a potential center for the Steiner tree with terminals r, u, v. Since all arc weights are non-negative, it suffices to consider the coarse nodes V_c as well as r, u, and v as center. The coarse graph can also be used for the shortest path computations. Suppose that we computed all distances $d(u, v)$ for all pairs (u, v) of coarse nodes, and let $i \in P_f(u_1, u_2)$ and $j \in P_f(v_1, v_2)$ be fine nodes on different coarse arcs, i.e., $(u_1, u_2) \neq (v_1, v_2) \neq (u_2, u_1)$. Since computing distances on a path takes linear time, we can efficiently compute the distance $d(i, j)$ from i to j as

$$\begin{aligned} d(i,j) = \min\{&d(i,u_1) + d(u_1,v_1) + d(v_1,j),\\ &d(i,u_1) + d(u_1,v_2) + d(v_2,j),\\ &d(i,u_2) + d(u_2,v_1) + d(v_1,j),\\ &d(i,u_2) + d(u_2,v_2) + d(v_2,j)\}. \end{aligned}$$

The computation of the distance from a coarse to a fine node is analogous. Altogether, we need time $\mathcal{O}(|E|)$ to compute all distances on paths and $\mathcal{O}(|V_c|^3)$ for the all pairs shortest path distances in the coarse graph D_c. In the worst case, we have to check all $\mathcal{O}(|V_c|)$ potential centers for all node pairs and can hence reduce the time to $\mathcal{O}(|V|^2\,|V_c|)$.

In practice, there are even more tricks to speed up the procedure. We might be able to exclude a set of potential centers for a certain node pair right away. Then, we can sort the remaining centers heuristically to obtain a feasible Steiner tree as early as possible. We can also make use of "implied conflicts": If u_1 and v_1 form a conflict pair and u_2 implies u_1 while v_2 implies v_1, then (u_2, v_2) is also a conflict pair. Finally, if the endpoints of two coarse arcs are crosswise in conflict then, obviously, every fine node of one arc is in conflict with every fine node of the other arc. We could extend the list of practical tricks but already the mentioned ones are not necessary for Algorithm 5.5 on our instances. There is, however, another variant in which these improvements play a vital role.

Analogously to the color radii that we improved to the bicolor radius, we can also extend the Steiner trees from above to bicolor Steiner trees. This means that for two nodes u and v, we only accept a Steiner tree with terminals r, u, and v that simultaneously meets the upper bounds of both color classes as "certificate of non-conflict". The good news is that this tree is still the union of three paths from a common center c. However, instead of adding three (one-dimensional) lengths, we have to check all combinations of two-dimensional Pareto-optimal labels to prove infeasibility. Even worse, we have to compute these Pareto-optimal labels from a potential center to all other nodes in advance. We implemented this approach, and even though one would expect the resulting (additional) conflicts to be particularly valuable, it had no positive effect on the solution time. Therefore, we do not describe this approach in more detail, and instead, we adhere to the conflicts generated by the simpler approach in Algorithm 5.5.

For most of our instances, this routine is already able to identify a large number of conflict pairs. In fact, in most cases there are so many conflict pairs that it is bad to include all of them and we need a reduction strategy. We will discuss two approaches: Deriving large conflict sets and identifying essential conflicts.

The Conflict Graph: The individual conflict pair inequalities $y_u + y_v \leq 1$ can be too weak to effectively strengthen the LP relaxation. We construct the *conflict graph* on the vertex set V by introducing an edge for every conflict pair. Analogously to the set packing problem (see [BW00] for details), we can derive stronger inequalities from the conflict graph: Given an odd cycle C of length $2k+1$ in the conflict graph, the inequality $\sum_{v \in C} y_v \leq k$ is known to be stronger than the ordinary conflict pair inequalities. These odd-cycle cuts, however, usually do not help the optimization process. The situation is different when considering a clique C in the conflict graph. The clique cuts $\sum_{v \in C} y_v \leq 1$ are known to be often beneficial for the LP relaxation.

We experimented with including clique cuts to our formulation. Since the number of cliques is even much larger than the number of conflict pairs, we determined an edge covering with cliques, and only added the respective clique cuts. These additional inequalities, however, showed to have a negative effect on the solution time in our tests. A typical conflict clique together with an optimal solution for the BRCMWCS instance is depicted in Figure 5.8. One can see that the conflicting nodes are quite far apart and located at the boundary of the graph. Depending on the node profits of the instance, such clique cuts are rarely violated by the optimal LP relaxations. Hence, we shift our focus to identifying more meaningful conflict pairs.

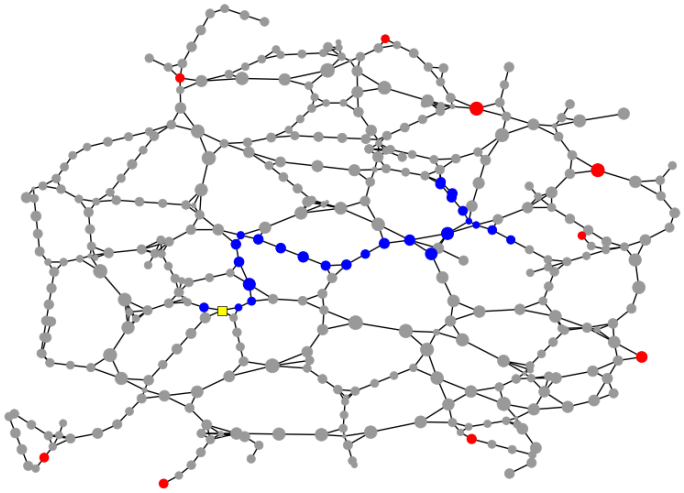


Figure 5.8.: The nodes of an exemplary conflict clique are colored red, an optimal solution is colored blue.

Essential Conflicts: Our experiments showed that the addition of all conflict cuts results in significantly longer solution times. The same holds true if only violated cuts are added dynamically to the program. Hence, it is necessary to identify essential conflicts that actually facilitate the solution process. Such conflicts presumably involve nodes that are closer to the root node or have a high profit.

In order to specify these nodes, we define a scoring function that assigns a value from the interval $[-1, 1]$ to every vertex. A positive profit as well as a small ratio between the shortest r-v-path length and W_U increase the score of node v. Essential conflicts are then defined as all conflict pairs between nodes with a positive score. Figure 5.9 shows the node scores and all resulting essential conflicts for an exemplary instance. Observe that the conflict sets are now much more central. We evaluate the impact of adding these essential conflicts to the respective MIP formulation in Section 5.6.5.

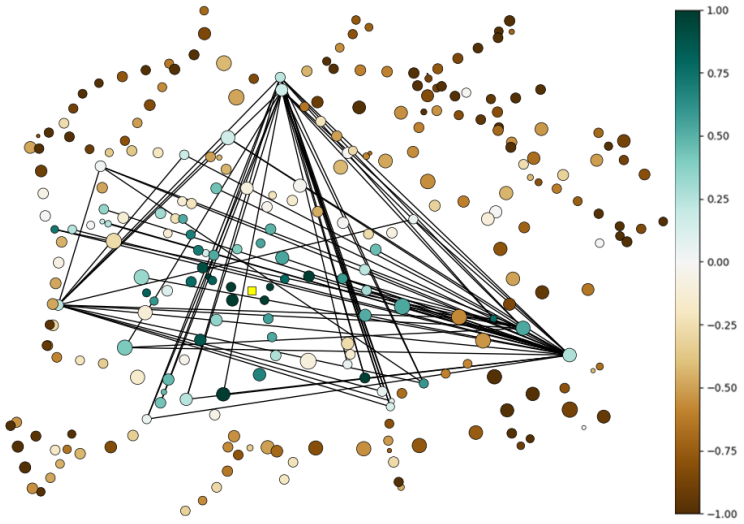


Figure 5.9.: Essential conflicts and node scores of an exemplary instance.

5.5.3 Binary Arc Variables

The arc separator formulation uses binary arc variables z that should describe an arborescence for the sought template. The seeming advantage of the node separator formulation is that it only uses node variables y. The single-commodity flow, on the other hand, has non-negative flow variables x for the arcs. While it might sound counterintuitive, we could add binary arc variables also to the SCF and RNS model. But why should we do this?

While this has obviously no effect concerning any integer solutions, it can help the LP relaxation. Indeed, we found that the inclusion of the z variables in combination with indegree constraints and 2-cycle cuts is highly beneficial for both models. One great advantage of the arc variables is their ability to break symmetries, as the following example shows.

Transferring indegree constraints to node variables, we get constraints of the form $\sum_{u \in N(v)} y_u \geq y_v$ for every $v \neq r$. Now, we consider the induced path (v_1, v_2, v_3, v_4) that does not contain r and note that $y_{v_1} = y_{v_4} = 0$ and $y_{v_2} = y_{v_3} = 1$ satisfy these constraints, since v_2 is an active neighbor of v_3 and vice versa. If, on the other hand, we consider indegree constraints in combination with 2-cycle cuts, this solution is not feasible anymore. From $y_{v_2} = y_{v_3} = 1$ and $z_{v_2v_3} + z_{v_3v_2} \leq 1$, we know that $z_{v_1v_2} + z_{v_4v_3} \geq 1$ which results in $y_{v_1} + y_{v_4} \geq 1$. The symmetry that v_2 and v_3 can act as active neighbor for each other is broken.

If we add binary arc variables z to the SCF formulation, we can also use them to activate the nodes that the flow uses. More specifically, we replace constraints (5.1c), i.e., $x_{uv} \leq M_{uv}\, y_v$ for all $(u, v) \in A$, with the following constraints:

$$x_{uv} \;\leq\; M_{uv}\, z_{uv} \qquad \forall (u, v) \in A.$$

Together with the 2-cycle cuts $z_{uv} + z_{vu} \leq y_v$ for $(u, v) \in A$, we obtain a stronger version of the original constraints.

The inclusion of z variables into SCF and RNS also allows us to incorporate the other strengthening cuts that we will present in the following. These concepts also have natural "node versions", i.e., cuts that only involve y variables, and we also experimented with these. However, our tests show that the arc versions perform better. Our explanation is that the node versions suffer from symmetry issues similar to the example above, and the arc versions have stronger implications.

For the remainder of this chapter, we will denote by SCF+z (respectively RNS+z) the formulation (5.1) (respectively (5.6)) with additional variables $z \in \{0, 1\}^A$ and additional constraints (5.3b) and (5.4).

5.5.4 Root Ring Cuts

Once again, we exploit the capacity constraint to derive implications for our MIP model.

Lemma 5.4 *Let S be a node separator in G and let C_r be the connected component of r in $G - S$. If $w(C_r[V_b]) < W_L^b$ or $w(C_r[V_r]) < W_L^r$, then*

$$\sum_{v \in S} y_v \;\geq\; 1$$

is a valid inequality for SCF, MCF, RAS, and RNS.

Proof. Obviously, C_r is not a feasible template as it violates a lower weight bound from Lemma 5.3. Therefore, more nodes have to be included and since the template has to be connected, at least one node in S is necessary. □

In order to present a simple method to identify disjoint separators that meet the stated criteria, we introduce a new notation. Let $N^i(v)$ be the set of vertices with shortest unit-weight distance i to vertex v. In particular, we have $N^0(v) = \{v\}$ and $N^1(v) = N(v)$, and we note that all of these node sets are pairwise disjoint. With an adjusted BFS we can now find the sets $N^1(r), \ldots, N^k(r)$ such that

$$w\left(\bigcup_{i=0}^{k-1} N^i(r) \cap V_b\right) < W_L^b \qquad \text{or} \qquad w\left(\bigcup_{i=0}^{k-1} N^i(r) \cap V_r\right) < W_L^r$$

and

$$w\left(\bigcup_{i=0}^{k} N^i(r) \cap V_b\right) \geq W_L^b \qquad \text{and} \qquad w\left(\bigcup_{i=0}^{k} N^i(r) \cap V_r\right) \geq W_L^r$$

Simply enough, the determined node sets $N^1(r), \ldots N^k(r)$ are separators that meet the condition in the lemma above. As each of these sets can be seen as a ring around the root node, we call the respective inequalities *root ring cuts*.

Instead of using node separators, we can do the described procedure also with arc separators. Therefore, we define

$$\delta^{+i}(r) := \left\{(u,v) \in A : u \in N^{i-1}(r), v \in N^i(r)\right\}.$$

Again, we have $\delta^{+1}(r) = \delta^+(r)$, and observe that $\delta^{+i}(r)$ and $\delta^{+j}(r)$ are disjoint for $i \neq j$. Analogously to the node separator case, we find the maximum number k such that the root component in $G - \delta^{+k}(r)$ is not feasible with respect to the lower color weight bounds. Finally, for $i \in \{1, \ldots, k\}$, we can add the valid inequalities

$$\sum_{a \in \delta^{+i}(r)} z_a \;\geq\; 1 \tag{5.16}$$

to SCF+z, RAS, and RNS+z. Our experiments show that the addition of arc root ring cuts is superior to adding the cuts with node variables. Figure 5.10 shows the root rings in different colors for an exemplary instance.

5.5.5 Extended Indegree Cuts

Recall that constraints (5.3b) from the RAS formulation are

$$\sum_{a \in \delta^-(v)} z_a \;=\; y_v \qquad \forall v \in V \setminus \{r\}.$$

As they state that each node of the arborescence spanned by z has exactly one arc going into it, they are called *indegree constraints*. Our idea to extend these constraints is just as simple as it proves to be effective. And yet, we are not aware of similar constraints in the literature with an equivalent effect.

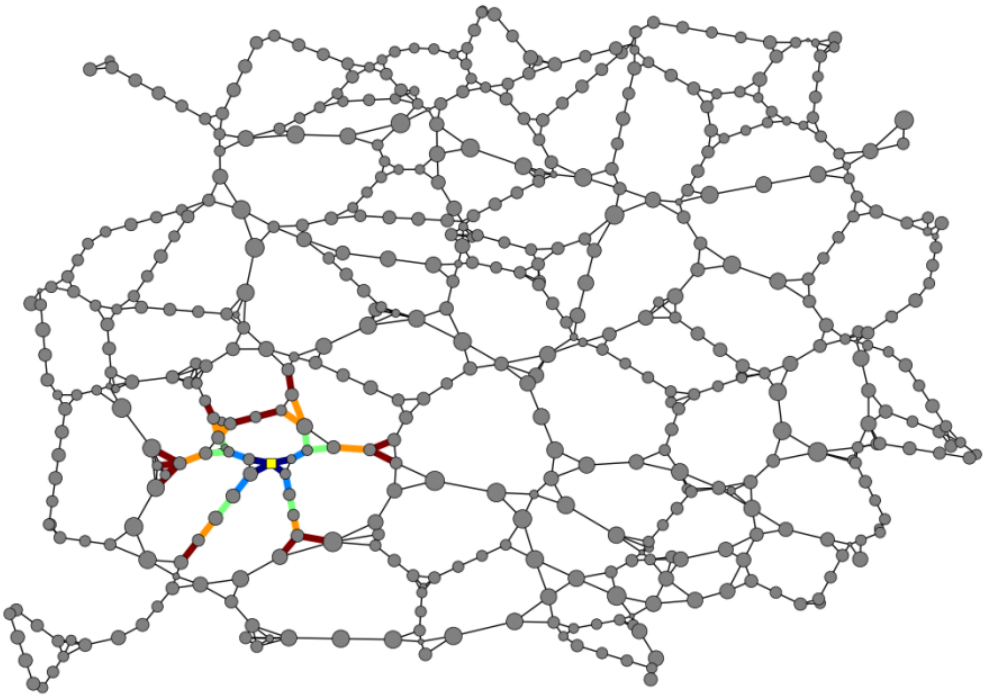


Figure 5.10.: Root rings: At least one arc of every color class has to be chosen to obtain a connected subgraph that meets the lower color weight bounds from Lemma 5.3.

For the presentation, we introduce a new notation that is in line with our definition for δ^{+i} above. Namely, for $x \in V$ and $i \in \mathbb{N}$ we write

$$\delta^{-i}(x) := \left\{(u,v) \in A : u \in N^i(x), v \in N^{i-1}(x)\right\}.$$

Proposition 5.5 *Let $v \in V$ and let $k \in \mathbb{N}$ such that $r \notin N^i(v)$ for $i \in \{0,\dots,k-1\}$. Then,*

$$\sum_{a \in \delta^{-k}(v)} z_a \;\geq\; y_v \tag{5.17}$$

is a valid inequality for SCF+z, RAS, and RNS+z.

Proof. Since $r \notin N^i(v)$ for $i \in [k]$, we know that $N^k(v)$ is an r-v-node separator and, hence, $\delta^{-k}(v)$ is an r-v-arc separator. Now, if $y_v = 0$, the inequality is trivially true, and in the other case, the cover inequality for the separator is valid. □

We call the cuts (5.17) *extended indegree cuts* (EIC), and denote all such cuts for $i \leq k$ with EIC-k. Since for vertices of degree 2, the 2-cycle cuts (5.4) have the same effect, it is sufficient to consider the EIC for vertices of degree ≥ 3. Note that the EIC are essentially certain arc separator constraints (5.3c). To see this, we set $S := \bigcup_{i=0}^{k-1} N^i(v)$ and note that $\delta^{-i}(v) = \delta^-(S)$. But what is the benefit of adding these cuts to the initial model?

The EIC essentially help to avoid small cycles and, in fact, they are more effectively than the k-cycle cuts (5.5). For example, consider the solution depicted in Figure 5.11. Note that this solution is feasible for the initial RAS model, i.e., (5.3) without (5.3c), and

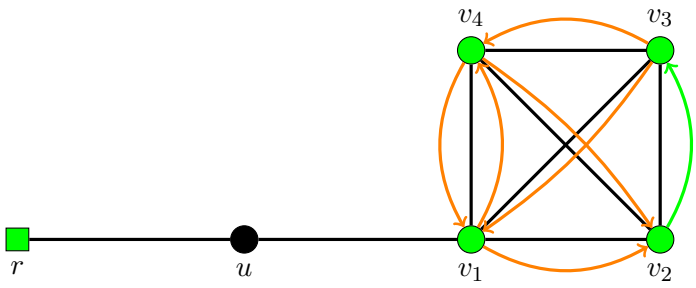


Figure 5.11.: (Infeasible) solution which is feasible for the initial RAS model. The y and z variables with positive value are displayed in color: orange represents the value $\frac{1}{2}$ and green the value 1.

it remains feasible with additional 2-cycle and 3-cycle inequalities. The solution is not even infeasible with 4-cycle inequalities, as one can verify. However, already adding the constraints for EIC-2 breaks the cycle since $\delta^{-2}(v_1) = \{(r, u)\}$ and $\delta^{-2}(v_2) = \{(u, v_1)\}$.

Alternatively, one could lift one of the 4-cycle inequalities to include all arcs of the induced subgraph, i.e.,

$$\sum_{a \in A(G[C])} z_a \leq \sum_{v \in C \setminus \{u\}} y_v$$

for the cycle $C = (v_1, v_2, v_3, v_4)$. While this cuts off the displayed LP solution, these constraints are more complex. Also, we would need lifted k-cycle inequalities to "escape" a clique of size k whereas EIC-2 still suffice in this case.

The strength of the extended indegree cuts is also confirmed in our computational tests. We find that in practice, the EIC-2 constraints perform much better than the 3-cycle cuts. While one cannot prove that EIC-2 dominate 3-cycle cuts (cf. Figure 5.12), we suspect that the practical dominance stems from the different behavior in cliques. Another advantage of the extended indegree cuts is that they are much simpler to construct compared to k-cycle cuts. In fact, an adapted BFS up to depth k suffices for EIC-k.

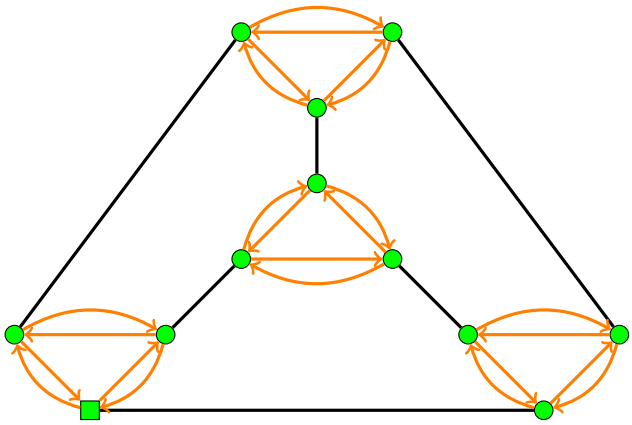


Figure 5.12.: (Infeasible) solution where EIC-2 constraints are satisfied but 3-cycle cuts would prevent this solution. The y and z variables with positive value are displayed in color: orange represents the value $\frac{1}{2}$ and green the value 1.

5.5.6 Fine Path Cuts

We have seen that the EIC are a subset of the arc separator constraints (5.3c) that are beneficial for the initial model. Here, we present a method to find other valuable arc separators to add the corresponding cuts initially.

To this end, we consider the coarse graph $D_c = (V_c, A_c)$ and for $(u, v) \in A_c$ we define the arc weight $w_{uv} := w_u + w_v + w(P_f(u, v))$. For a given node v, our goal is to find an r-v-arc separator in D_c of maximum weight. The motive is that the implication of the separator constraint from a large value of y_v is stronger because it affects a vertex set of larger weight. In order to find such a separator, we set arc capacities $c_{uv} := \frac{1}{w_{uv}^2}$ in D_c and compute a minimum r-v-cut X_c. We transform this cut to an r-v-arc separator X in D by replacing each arc $(u, v) \in X_c \setminus A$ with the last arc of the respective fine path in D. Then,

$$\sum_{a \in X} z_a \;\geq\; y_v \tag{5.18}$$

is a valid inequality for SCF+z, RAS, and RNS+z. . With the arc preceding constraints (5.8b) in combination with indegree constraints, the value of z_a for each $a \in X$ is now propagated across the fine path. In particular, we have $y_i \geq z_a$ for each node i of the fine path, boosting the implication from (5.18).

Instead of computing these cuts for every node, we do this only for a relevant subset. These important nodes can be solely derived from the network topology or can be determined dynamically, but in our case, we simply add *fine path cuts* for the top ten nodes with the highest profit.

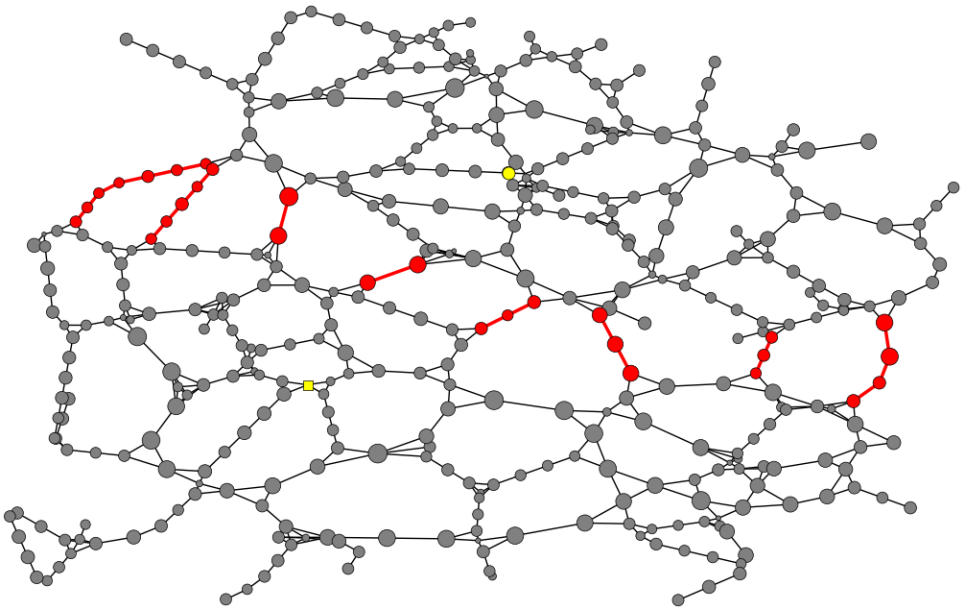


Figure 5.13.: Fine Path Cuts: All nodes and arcs of at least one colored fine path have to be active in order to reach the yellow node v.

5.6 Computational Study

The goal of our computational study is to evaluate the different models and improvements that we reviewed and introduced. In particular, we compare the different connectivity formulations, and assess the impact of the coarse-to-fine model, the inclusion of conflicts, and the other strengthening techniques that we presented.

We start with an introduction of our test instance set. In Section 5.6.2, we detail the test setting and describe the experiments.

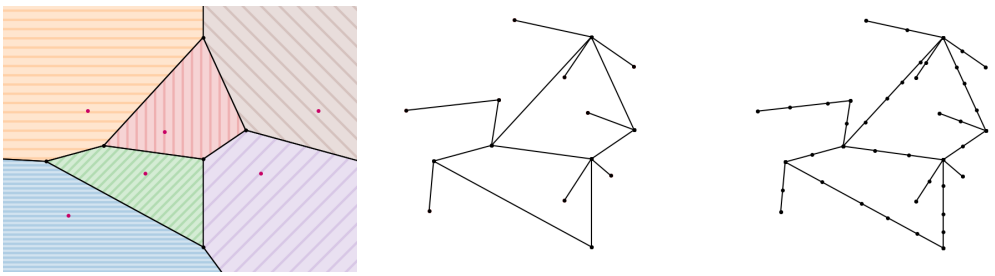
5.6.1 Test Instances

We are not aware of any test instances from the literature that fit our setting of the BRCMWCS. Consequently, we generated a new set of instances. Our goal is to mimic transit networks, and we found that Voronoi diagrams are an excellent starting point.

The basic Voronoi graph is spanned by the ridges of the Voronoi cells (cf. Figure 5.14a) and we use Euclidean edge weights. Now, we add additional random leaves (cf. Figure 5.14b), split the edges to obtain a specified number of nodes (cf. Figure 5.14c), and stretch each edge by a random factor. The resulting networks resemble typical subgraphs of road networks. In Section 6.4.1, we will also mimic the traffic flow on such instances and analyze the resulting traffic networks in more detail.

For this chapter, it is sufficient to note that we consider four groups of instances that are depicted in Figure 5.15. First, we consider actual pricing instances from the next chapter that are defined on the line graph of Voronoi graphs (details on this are given in Section 6.4.1). These are contained in the set `voronoi` which consists of 45 instances: We have five different networks (and node colorings) and for each, we consider nine different profits (three are actual reduced costs from the pricing problem, three are distributed uniformly at random in the interval $[-1, 1]$, and three are based on the normal distribution to thin out extreme profits). Then, we consider three other instance classes for which we only use random colors and random profits: `large_voronoi` contains instances that are similar to the first class but are much larger, `sparse_voronoi` with instances having a very large fraction of vertices with degree 2, and `grid` which consists of 25×25 grids with the central node being the root.

Some important attributes for the instance classes are summarized in Table 5.1. Smaller values for $f_L[\%]$ and $f_U[\%]$ indicate a bigger potential for the preprocessing routine. As we did not want to blow up the instance size just to reduce it in the preprocessing, only few or often no vertices are outside of the W_U radius of the root node.

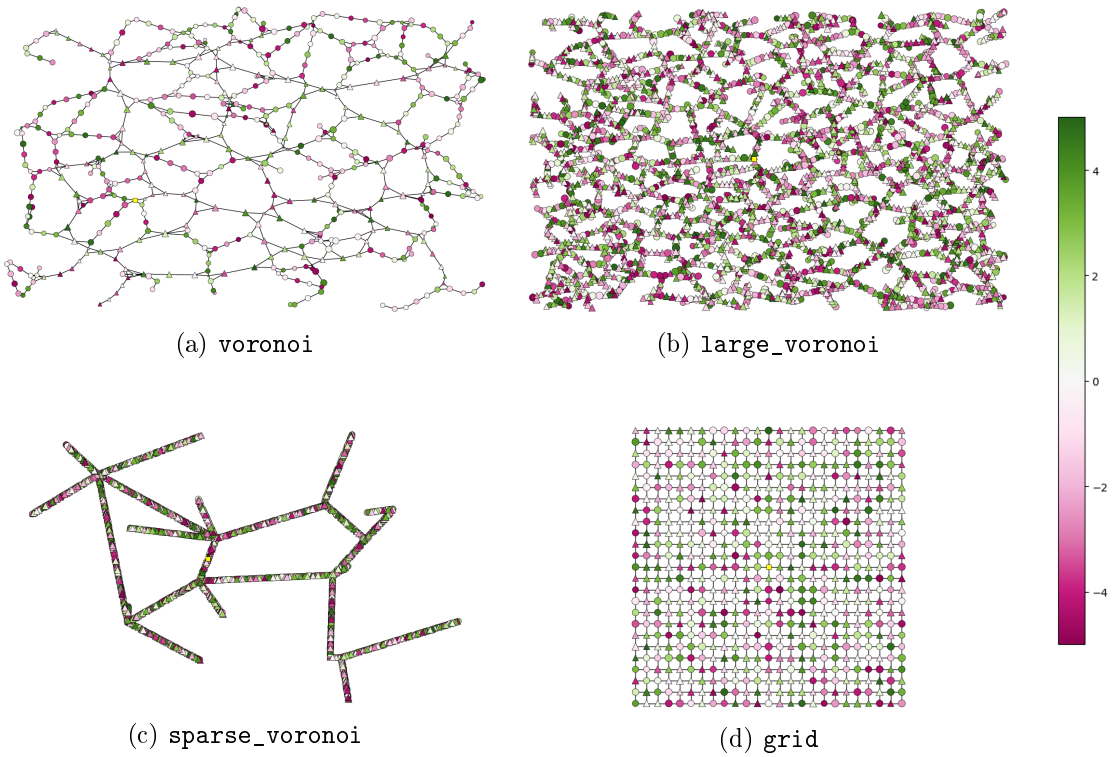


(a) (b) (c)

Figure 5.14.: Construction steps of Voronoi instances.

Table 5.1.: Selected properties of considered instance sets: # denotes the number of instances in the respective group, $f_L[\%]$ the average fraction of nodes within the lower weight radius around the root, and $f_U[\%]$ the fraction w.r.t. the respective upper bound.

Instance set	#	$\lvert V\rvert$	$\lvert E\rvert$	$f_L[\%]$	$f_U[\%]$
`voronoi`	45	565.4	864.0	45.9%	94.7%
`large_voronoi`	10	3994.1	5491.5	74.1%	100.0%
`sparse_voronoi`	10	2500.6	2517.4	73.6%	99.9%
`grid`	10	625.0	1200.0	100.0%	100.0%



(a) `voronoi` (b) `large_voronoi`

(c) `sparse_voronoi` (d) `grid`

Figure 5.15.: Exemplary BRCMWCS instances. The root node is depicted as a yellow square, circles represent nodes in V_b and triangles nodes in V_r, node sizes correspond to weights and colors to node profits.

5.6.2 Experiments and Test Setting

We can group our experiments into four categories:

- a comparison of the connectivity models,
- the assessment of the impact of our coarse-to-fine model,
- an evaluation of adding conflict cuts,
- and the analysis of the impact of selected single parameters.

Together with the connectivity models, we evaluate the effect of adding binary arc variables to SCF and RNS as proposed in Section 5.5.3, and we also see the enormous effect of combining all improving methods. Taking the example of single-commodity flow, we use the following notation: SCF denotes the basic formulation as it is introduced in Section 5.1. SCF+z also includes binary arc variables z in combination with indegree and 2-cycle constraints, and SCF+C2F is the basic flow formulation on the coarse graph with the predecessor formulation for the fine paths. Finally, SCF* denotes the single-commodity flow formulation with the best combination of our improving methods that we found in extensive tests that go far beyond the presented settings. This best combination is detailed in Section 5.6.6 but it basically means the use of all proposed methods except for the coarse-to-fine model and conflict cuts. For the moment, it suffices to view the * version as "the optimal" setting.

We ran our experiments on machines equipped with Intel Xeon E3-1245 CPUs with 3.70GHz and 32GB RAM. Our code is written in Python 3.8 and we use Gurobi 9.5 with default settings as LP and MIP solver. We introduce a time limit of one hour, and instances that terminate due to memory errors are set to the time limit. All times reported in the following are in seconds.

Details on RAS: In the basic variant, the initial model of RAS is only strengthened with 2-cycle inequalities. During the separation, we ignore nodes with $y_v \leq 0.999$ and compute a maximum flow only for each non-root component. With the flow computation in Python code, however, the separation routine for the RAS is the bottleneck of this approach. By using the maximum flow function from scipy, which is written in C, we obtain a massive speed-up and the separation is no longer the bottleneck. For our instances, nested cuts, minimum cuts of minimum cardinality, and generalized subtour elimination constraints did not provide any advantages, but the inclusion of back-cuts showed to have a slight positive effect overall. The influence of back-cuts is also discussed in Section 5.6.6.

Separation for RNS: We use the adapted approach of Fischetti et al. [Fis+17] and call the separation only for integer solutions. In consequence, we observe that almost no time is spent in the separation routine.

5.6.3 Comparison of Connectivity Models

Our first results concern the choice of the connectivity model, as shown in Table 5.2. We provide instance-wise solution times in Tables A.1 and A.2 in the appendix.

When comparing the basic variants, the arc separator formulation is clearly the best. For the `voronoi` instances, the SCF formulation at least finds solutions in reasonable time, but for most of the other instances and for almost all instances with RNS, the time limit

Table 5.2.: Average computation times (in seconds) for different connectivity formulations in the basic variant, with z variables, and as optimal variants.

Instance set	SCF	MCF	RAS	RNS	SCF+z	RNS+z
`voronoi`	321.0	3124.3	8.8	3495.5	84.4	3163.3
`large_voronoi`	3600.0	3600.0	556.2	3600.0	2403.4	3600.0
`sparse_voronoi`	3240.1	3015.3	6.0	3282.2	95.9	2565.7
`grid`	3045.1	3600.0	21.9	3600.0	1228.9	3600.0

Instance set	SCF*	MCF*	RAS*	RNS*
`voronoi`	10.1	1957.6	3.6	186.3
`large_voronoi`	201.2	3600.0	69.0	3246.6
`sparse_voronoi`	3.9	2997.6	2.1	1.7
`grid`	389.5	3600.0	24.1	3600.0

of one hour is reached. The timeout for MCF, on the other hand, is mostly due to memory issues.

The impact of adding z variables with indegree and 2-cycle constraints to SCF is remarkable, especially for sparse instances, providing a speed-up factor of over 33. Concerning the RNS formulation, this addition on its own is only helpful for a handful of instances. More specifically, for 8 of the 75 instances this leads to a substantial improvement, while for 2 instances, we observe a negative effect.

Adding the other improving cuts, however, leads to a massive reduction of computation times across the board. Here, we can already witness the gigantic impact of our proposed improvements. The arc separator formulation is still the clear favorite by far. And while the MCF and RNS formulation still struggle with solving the `large_voronoi` and `grid` instances within the time limit, there is a clear improvement for the `voronoi` instances and, in case of the RNS, especially for the sparse instances. For the latter class, RNS* is now even the fastest method for 9 out of 10 instances, although the solution times are generally much lower than for any other instance set.

To sum up, the clear winner of this comparison is the RAS formulation and the clear loser the MCF. The RNS* seems particularly suited for sparse graphs but is relatively useless for the other instance sets. Finally, the SCF* formulation is the only practical alternative to RAS* but on our test instances, it is clearly dominated by RAS* and there is no argument for using SCF* instead[1].

5.6.4 The Impact of Coarse-to-Fine

Next, we evaluate the coarse-to-fine approach and, as stated above, we only consider C2F in combination with SCF and RAS. The average optimization times for our instance groups are given in Table 5.3, instance-wise results are given in Table A.3 and Table A.4 in the appendix.

Combined with the basic flow formulation, C2F brings a slight advantage overall, but

[1] Spoiler alert: One might add the word "yet", and prepare for a surprise in Chapter 6.

Table 5.3.: Average computation times (in seconds) for different variants of SCF and RAS formulations without and with the coarse-to-fine (C2F) model.

Instance set	SCF	+C2F	SCF+z	+C2F	SCF*	+C2F
`voronoi`	321.0	318.3	84.4	73.8	10.1	13.7
`large_voronoi`	3600.0	3600.0	2403.4	2876.9	201.2	1069.9
`sparse_voronoi`	3240.1	87.8	95.9	7.3	3.9	3.9
`grid`	3045.1	2960.2	1228.9	877.1	389.5	380.1

Instance set	RAS	+C2F	RAS*	+C2F
`voronoi`	8.8	9.7	3.6	4.6
`large_voronoi`	556.2	702.0	69.0	535.6
`sparse_voronoi`	6.0	3.6	2.1	4.7
`grid`	21.9	21.6	24.1	23.8

for the sparse instances, the difference is significant. If we add z variables, the effect is similar: A great improvement for `sparse_voronoi`, a slightly positive effect for the `voronoi` instances, and a negative effect for large instances. When considering the best flow formulation SCF*, however, the effect of C2F becomes mostly negative. There are still some sparse instances that benefit from the C2F formulation, but over the ten instances, the advantage disappears.

The difference in the grid instances is surprising, to say the least. The only difference between the grid graph and its coarse counterpart are the four "corners" of the grid, and since the root is the central node of the grid, these nodes are, generally speaking, the most unimportant ones. At closer look, the main issue of the grid instances is the dual bound. And while one formulation does not provide a better bound than the other, the leaps in closing the gap in Gurobi essentially determine the runtime. One could investigate this further to better understand this behavior, but since grid instances are not the focus of our work, we did not pursue this direction.

The combination of RAS with C2F does not have a clearly positive effect. While it benefits the sparse instances and it also has a positive effect on some other instances, the overall performance with C2F is worse than the original formulation. Again, with the optimal parameter setting RAS*, the C2F model is simply worse.

In conclusion, we can say that the C2F model can have a significant effect when compared with the basic variants. It seems, however, that all of our other improvements replace the advantage that comes from this model.

5.6.5 The Impact of Conflicts

The evaluation of the effect of our conflict cuts leads to similar results as for the C2F, as Table 5.4 shows. When combined with the original models SCF and RAS, the filtered conflicts often help the solution process. As the detailed results in Tables A.5 and A.6 show, the addition of conflicts to the basic model is better for the vast majority of instances.

The combination of conflict cuts and the strengthened formulations SCF* and RAS*, however, leads to longer solution times. Still, there are single instances where the conflicts

Table 5.4.: Average computation times (in seconds) for the basic and optimal variants of SCF and RAS without and with the addition of (essential) conflict pairs.

Instance set	SCF	+CP	SCF*	+CP
voronoi	321.0	205.6	10.1	10.8
large_voronoi	3600.0	3600.0	201.2	447.6
sparse_voronoi	3240.1	3240.1	3.9	5.2
grid	3045.1	2980.4	389.5	341.6

Instance set	RAS	+CP	RAS*	+CP
voronoi	8.8	7.6	3.6	3.9
large_voronoi	556.2	673.8	69.0	91.6
sparse_voronoi	6.0	7.0	2.1	2.4
grid	21.9	18.9	24.1	21.5

bring a significant improvement, but mostly, they affect the solution process negatively.

This is not a surprise when we reconsider the solutions of the LP relaxations in Figure 5.7. It is easy to believe that the merit of conflict cuts is greater in a very weak LP relaxation than in the strengthened formulation. Similar to the C2F approach, it seems as if our other strengthening improvements dominate the conflict cuts and, except for a few cases, make them obsolete.

5.6.6 Computational Analysis

We tested numerous combinations of the parameters and in the following, we present the setting that performed best overall and evaluate the impact of selected parameters.

For the best setting, we use the preprocessing with node fixing and removal, and we use implied nodes, binary arc variables z for SCF and RNS, root ring cuts, and fine path cuts. Concerning the extended indegree cuts, we found EIC-2 to perform best, but below, we also present results for EIC-3 and without EIC, i.e., only with indegree constraints. EIC-4 is mostly unnecessary for our instances and leads to extended running times. Finally, adding the color range constraints (5.15) obtained in the preprocessing is also a slight improvement, and evaluated below. As we have seen, the coarse-to-fine approach and the conflict cuts do not help with this optimal setting

We point out that the preprocessing as well as the implied nodes have a limited, but still slightly positive impact on our instances. The instances are simply not sparse enough for node fixing to occur, and the capacity bounds are designed to mostly represent instances *after* the preprocessing. This is completely different when we consider the BRCMWCS as a pricing problem in the next chapter, especially on real-world instances. In this case, the preprocessing has an enormous impact on the computation time.

Our base variant for the evaluation is RAS* and Table 5.5 shows our selection of variants and their average solution times. Here, –BC indicates that we do not use back-cuts in the separation, –CR stands for the exclusion of color range constraints, and +EIC-3 describes the inclusion of extended indegree cuts of level 3 to the RAS* setting. Consequently, –EIC means the exclusion of any EIC, and, finally, –SpC is the parameter setting of RAS* but

Table 5.5.: Average computation times (in seconds) for different variants of the optimal RAS formulation: without back-cuts (–BC), without color range constraints (–CR), with extended indegree cuts of level 3 (+EIC-3), without EIC (–EIC), without separation per component (–SpC).

Instance set	RAS*	–BC	–CR	+EIC-3	–EIC	–SpC
`voronoi`	3.6	3.5	3.8	3.7	6.0	6.1
`large_voronoi`	69.0	80.7	75.6	74.3	554.7	346.8
`sparse_voronoi`	2.1	2.0	1.8	1.9	4.6	15.1
`grid`	24.1	17.1	23.0	18.9	19.0	62.9

we compute a maximum r-v-flow for every v with $y_v > 1 - \varepsilon$ instead of separating per component as suggested in Section 5.1.3.

Starting with the last two columns, we observe that each of these proposed improvements has a significant impact on the runtime, and it is always better to use both options. The drop-off, in particular for large instances, is remarkable considering the fact that there is only a single ingredient missing.

For the comparison of the other variants, we consider the instance sets separately, starting with `voronoi`. While at first sight, it seems as if all variants perform similarly, the instance-wise results in Table A.7 show that the times for the single instances actually vary but balance over the set of 45 instances. What if we could determine the best parameter setting for a given instance? To assess the potential of this approach, we take the minimum time for each row and end up with an average runtime of 3.0 seconds. Interestingly, the optimal decision between using back-cuts or not already leads to 3.2 seconds on average. We did not study the prediction of optimal parameter sets from instance characteristics as our test set is much to small for this purpose.

For the `large_voronoi` instances, the variance over the single instances is similar. In this case, the best complement to the base variant is +EIC-3, as one of the two usually has the best or close to the best runtime. Again, with the assumption of an oracle that decides if we add EIC-2 as in the base model or EIC-3, the average runtime would drop to 61.8 seconds.

Concerning the sparse instances, all variants are very fast, with –CR having the slight edge on these ten instances. For the grid instances, we find that –BC and +EIC-3 are the two best variants. If we combine these two, i.e., we consider RAS* without back-cuts and with EIC-3 constraints, this variant is even better on grid instances, achieving an average runtime of 16.4 seconds.

Apart from the described variants, we also experimented with different branching priorities that are given to the solver. Our idea was that while fixing nodes or arcs far from the root has strong implications, the other branch, i.e., removing these elements, does often not affect the solution process. We experimented with branching first on nodes or arcs close to the root, but in this case, the fixing has not so much impact. Finally, to combine the advantages of both extremes, we tried to branch first on arcs that are moderately close to the root. More specifically, we compute a minimum-cardinality cut between $S = \{v : d(r, v) \leq \frac{1}{4} W_L\}$ and $T = \{v : d(r, v) \geq \frac{1}{2} W_L\}$, and start the branching on these arc variables. The motivation behind this is that both branches, i.e., to include or exclude

this arc, have noticeable implications. This method, however, ignores the profits of the nodes, and our tests show that the innate branching strategy in Gurobi is better.

To end the computational study of the BRCMWCS, we conclude that our methods drastically reduce the computation times across the board for the most important connectivity formulation (i.e., RAS) and for the SCF. Especially for the latter, we improve the basic formulation by orders of magnitudes. Concerning the RAS, the greatest effect comes from the introduced extended indegree cuts, but the complementary improvements further support the optimization procedure.

5.7 BRCMWCS on Trees

Many problems in combinatorial optimization are NP-hard in general but polynomially solvable on simpler instances such as trees. To end this chapter, we study the BRCMWCS with the condition that the given graph is a tree. This facilitates the connectivity condition in the sense that there is now a unique path from the root to each node. On the other hand, the tree structure is also very suited for a dynamic programming approach, and we will present such an approach in Section 5.7.1. Finally, we explore and evaluate the possibility of using the tree case as a primal heuristic for BRCMWCS on general graphs and to provide the heuristic solution as a warm start.

We start, however, by noting that the BRCMWCS on a tree is still NP-hard. To see this, recall that the complexity proof in Section 5.3 includes a reduction from the number partition problem to a BRCMWCS instance on a star graph. Therefore, there is no hope for a polynomial-time algorithm solving the BRCMWCS on a tree. The connectivity formulation in the IP, however, can be simplified.

IP Formulation for Connectivity: As mentioned in Section 5.5.1, every edge of a tree induces an implied node inequality: Including the vertex $v \neq r$ in any solution implies the inclusion of the predecessor $\pi(v)$ on the unique r-v-path. Magnanti and Wolsey [MW95] prove that these implied node inequalities suffice for a full description of the r-tree polytope if G is a tree. More specifically, the following constraints describe $conv(\mathcal{Y}_r)$:

$$y_r = 1 \tag{5.19a}$$

$$y_v \leq y_{\pi(v)} \qquad \forall v \in V \setminus \{r\} \tag{5.19b}$$

$$y_v \geq 0 \qquad \forall v \in V \tag{5.19c}$$

Note that the integrality of the y variables is relaxed and that, hence, a rooted connected subgraph of maximum weight can be computed with a linear program. Alternatively, the rooted MWCS on a tree can also be determined with a dynamic program as we show next.

5.7.1 Dynamic Programming Approach

For the presentation of the dynamic program, we adopt the notation for rooted trees from the approximation in Section 3.2.2.

A maximum weight r-subtree of an r-tree is usually determined by a dynamic program. The basic idea of a dynamic program on a rooted tree T^r is to recursively break down the problem to subtrees. Following the presentation of [MW95] for a dynamic program solving the rooted MWCS on a tree, we denote by $H(v)$ the optimal value of the rooted MWCS

in T_v^r. In a bottom-up approach we start by setting $H(v) = \max\{0, p_v\}$ for every leaf node $v \in T^r$. Considering an arbitrary vertex v and assuming that $H(u)$ is computed for all successors $u \in N^+(v)$, we can determine $H(v)$ as

$$H(v) = \max\{0, p_v + \sum_{u \in N^+(v)} H(u)\}.$$

Following this recursion, we can determine the optimal value $H(r)$ in time $\mathcal{O}(|V|)$. An optimal solution is now attained by computing the root component of $T^r - \{v : H(v) = 0\}$.

Magnanti and Wolsey [MW95] also propose a dynamic programming approach for the capacitated version, but only with unit weights and for the case $W_L = 0$, i.e., without a lower weight bound. In the following, we propose a dynamic program that is more suited to the setting of BRCMWCS.

To this end, we need integer weights $w \in \mathbb{N}_{\geq 0}^V$, and while this naturally restricts Δ, W_L, and W_U to non-negative integers, the profits can still be arbitrary. Our approach is derived from the dynamic program for the partially ordered knapsack problem by Johnson and Niemi [JN83]. This problem asks for a maximum-valued subset of vertices in a graph whose total weight does not exceed a given knapsack capacity, and which, together with any vertex v, contains every predecessor of v w.r.t. the given ordering. In our setting, the problem is extended by a lower capacity bound and the balancing condition.

One key result from [JN83] is that the bottom-up approach stated above is suboptimal and should be replaced by a "left-right" approach. The idea is to consider a smaller set of subproblems for the dynamic program by traversing the tree in a specific order.

Therefore, let $v_1, \ldots, v_n$ be a DFS ordering of T^r with $v_1 = r$. By $d(v) := |N^+(v)|$ we denote the number of children of node v. For $j \in [n]$ and $i \in \{0, 1, \ldots, d(v_j)\}$, we denote by $T[v_j, i]$ the subtree of T^r containing all nodes v_k with $k \leq j$ together with the first i children of v_j (in order of the indices) and all their descendants. Figure 5.16 depicts an example for such a subtree.

For ease of exposition we define

$$c_v = \begin{cases} w_v & , \text{ if } v_j \in V_r \\ -w_v & , \text{ if } v_j \in V_b \end{cases}.$$

For a given $\delta \in \mathbb{Z}$ and $\omega \in \mathbb{N}$ the subproblem $\mathrm{dp}[v_j, i, \delta, \omega]$ is to find the maximum profit subtree $S \subseteq T[v_j, i]$ such that $v_1, v_j \in S$, $c(S) = \delta$, and $w(S) = \omega$. The subproblems of

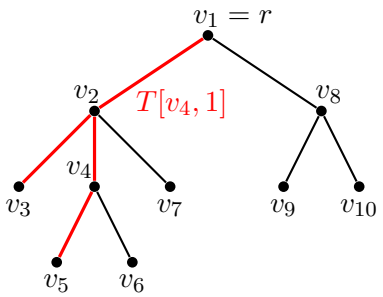


Figure 5.16.: A rooted tree with DFS ordering on V and the subtree $T[v_4, 1] = T[v_5, 0]$ highlighted.

the dynamic program are solved in a DFS style beginning from the root node r. We set

$$\mathrm{dp}[r, 0, \delta, \omega] = \begin{cases} p_r & \text{, if } \delta = c_r \text{ and } \omega = w_r, \\ -\infty & \text{otherwise.} \end{cases}$$

Let us next consider a node u together with its i-th child v. The solution of $\mathrm{dp}[u, i, \delta, \omega]$ can either use v or not. The former is equivalent to $\mathrm{dp}[v, d(v), \delta, \omega]$ and the latter to $\mathrm{dp}[u, i-1, \delta, \omega]$. Consequently,

$$\mathrm{dp}[u, i, \delta, \omega] = \max\left\{\mathrm{dp}[v, d(v), \delta, \omega],\ \mathrm{dp}[u, i-1, \delta, \omega]\right\}.$$

Finally, we have the case $\mathrm{dp}[v, 0, \delta, \omega]$, i.e., we exclude all children of $v \neq r$ from the solution. Again, we assume that v is the i-th child of its parent u and obtain

$$\mathrm{dp}[v, 0, \delta, \omega] = \mathrm{dp}[u,\ i-1,\ \delta - c_v,\ \omega - w_v] + p_v.$$

The integer-weighted BRCMWCS on T^r is now equivalent to

$$\max_{\delta \in [-\Delta, \Delta],\ \omega \in [W_L, W_U]} \mathrm{dp}[r, d(r), \delta, \omega],$$

and we need to consider $\mathcal{O}(n\,\Delta\,W_U)$ subproblems to solve it. Since each subproblem is solved in $\mathcal{O}(1)$, the presented dynamic program runs in pseudo-polynomial time: $\mathcal{O}(n\,\Delta\,W_U)$.

We did, however, not implement this approach because, first, scaling our rational weights to integers results in large values for Δ and W_U and, second, our tests show that the tree IP (5.19) is so fast that no alternative is needed in practice.

5.7.2 A Primal Heuristic for BRCMWCS

Given a BRCMWCS on a general graph G, we can compute a spanning tree T of G and solve the BRCMWCS on T. Clearly, any feasible solution on T is also feasible on G and, hence, we can use this as a primal heuristic for the problem on G. Furthermore, we can give this solution to the MIP solver as a warm start.

The first question is which spanning tree we use. For our approach, we use a BFS tree as spanning tree T because, first, it is very simple to compute and, second, it minimizes the number of implied nodes which has a positive effect on the number of feasible solutions in T.

The quality of our heuristic solution together with the effect of using it as a warm start is shown in Table 5.6. It is not perfectly clear how the quality of the heuristic solution should be measured. Note that the BRCMWCS allows for negative objective values and 0 is not a natural lower bound. We still opt for the ratio of the objective values obj_T and obj_G of the heuristic and exact approach, respectively, i.e.,

$$\mathrm{obj}[\%] = \frac{\mathrm{obj}_T}{\mathrm{obj}_G}.$$

In two cases this produces misleading values: For i-13 we have an optimum with value -178.8 and a heuristic solution with objective -507.1, and for i-33 we have an optimal value of 174.9 but a heuristic objective value of -3.1. All detailed results are given in Table A.8.

Table 5.6.: Performance of the BFS heuristic: average runtime (H), objective gap, and impact on the solution time if added to the optimal variants of RAS and SCF as warm start.

Instance set	H	obj[%]	RAS*	+WS	SCF*	+WS
`voronoi`	0.1	84.2%	3.6	5.7	10.1	68.6
`large_voronoi`	0.6	84.0%	69.0	560.1	201.2	2293.3
`sparse_voronoi`	0.7	89.8%	2.1	5.0	3.9	87.5
`grid`	0.1	61.0%	24.1	19.1	389.5	964.3

Overall, we observe that the problem is indeed solved much faster on a tree and that the heuristic mostly provides decent solutions. It is therefore surprising that the effect of providing this solution to Gurobi as a warm start is somewhere between bad and catastrophic. Even for the 6 out of 10 sparse instances for which the heuristic finds an optimal solution, adding the warm start is clearly worse overall. At closer look, the warm start solution negatively affects Gurobi's closing of the dual bound. In other experiments, we added the heuristic objective value as a lower bound constraint instead of using the warm start. The effects of this are, however, comparable to the warm start.

6

Designing Optimal Toll Sections

The time has come to tackle the problem that motivates our research: The Toll Section Design Problem. We have given a rough description of this problem in the introduction but here, more specifically in Section 6.1, we provide a more detailed motivation and a formal definition. Since we model this problem as a districting problem, we revisit and adapt the main solution approaches from Chapter 4 in Section 6.2. Pursuing the column generation approach, we investigate the respective pricing problem in Section 6.3. The pricing boils down to finding a capacitated MWCS with a complex objective function. We take a novel perspective and consider the pricing problem as a rooted MWCS for each possible center of a template. The resulting formulation allows for a much simpler objective function and we are able to transform this problem into the BRCMWCS. This enables us to apply the powerful reduction techniques presented in Chapter 5, and we propose two algorithmic enhancements for the column generation approach. In Section 6.4, we perform an extensive computational study on 24 real-world and 75 artificial instances that mimic the former, and show the tremendous effect of the proposed enhancements. After embedding the column generation into a full-fledged branch-and-price approach in Section 6.5, we dismiss the use of a primal heuristic for our covering problem in Section 6.6. We close the chapter by demonstrating that the methods developed in this thesis translate to other districting problems by employing our model to a districting problem from the literature.

6.1 The Toll Section Design Problem

Many countries rely on motorway tolls to fund growing investments for maintenance and extension of the networks. In Germany, a truck toll on motorways was introduced in 2005 and today, every truck weighing 7.5 tonnes or more has to pay a fee for every trip based on the traveled distance, the emission category of the truck, and its number of axles. In 2021, all trucks covered a total distance of 41.8 billion kilometers on the toll network, resulting in total revenues of over 7.5 billion euros [Bag; Bmd].

In contrast to many other toll systems that rely on toll stations at every entry and exit, the German toll system is user-friendly as it avoids these barriers. Instead, every trip is logged automatically with an on-board device via GPS or, alternatively, the drivers have to manually book their trip at toll station terminals or over the internet in advance. The toll enforcement is the responsibility of the German Federal Office for Goods Transport (BAG) and for this purpose, they use a combination of automatic and mobile controls. For the automatic control, around 300 cameras are installed at selected positions of the 14,000 kilometers spanning network but due to data protection, they are not always active and only survey a small fraction of the traffic. Complementary to the automatic control, mobile control teams consisting of one or two inspectors patrol the network in search for toll evaders.

An ongoing research project between the BAG and the Zuse Institute Berlin that started in 2010 studied different aspects of optimizing the toll control. This includes the strategically optimal placement of cameras for automatic controls as well as the development of strategies for an efficient integrated planning of automatic and mobile controls. The core of the project, however, is a model to optimize the duty planning of the mobile control teams which is subject to a variety of legal constraints. A duty consists of different parts, and in every duty part the team patrols a certain subpart of the toll network, called *section*. It is the decision of the control team where exactly the patrol is performed within the section. When the inspectors suspect a truck of evading the toll, they can pull the truck over to handle the case immediately.

For the optimization of the mobile tours, it is therefore not sufficient to find an optimal assignment of teams to sections at certain times. We also have to address the issue of designing the toll sections as this has a significant effect on the performed control tours. This effect was indeed shown in different experiments that are not part of this thesis. We computed optimal duty rosters based on the classical toll sections and our optimized sections. The objective of the rostering model is multi-dimensional but mainly, it maximizes the expected number of controlled trucks. Our experiments showed that the objective with our optimized toll sections was 30% higher than with the actual sections. This illustrates the enormous potential that lies in the optimization of toll sections. It is also intuitively clear that this design problem is just as important as the subsequent assignment problem. If the sections are designed in a bad manner, an optimal assignment cannot fix this.

In order to formulate an optimization problem, however, we need more specific information: What are the restrictions and what is the objective? From experience, inspectors prefer roads with higher truck traffic volume, as they expect more cases to handle. Therefore, if a section contains roads with high traffic volume but also other roads with rather low volume, the inspectors will often ignore the latter part of the section which contradicts an important objective of the toll control: the complete coverage of the network with controls. For this reason, it is preferred to build sections consisting of roads with similar traffic volume. Apart from the objective of aiming for homogeneous sections, there are a number of obvious conditions for the section design. Each section is subject to a lower and an upper length bound such that the total length fits to the typical time window of a duty part. Furthermore, both directions of a road section have to belong to the same section, allowing us to model the network as an undirected graph. Then, each section is obviously required to be connected and, finally, our sections have to cover the network, i.e., each edge has to be part of at least one section.

The modeling process is illustrated in Figure 6.2. Before the transformation to the line graph, we merge both directions of a road section to an edge. The length of such an edge uv is the sum of the lengths of (u, v) and (v, u) while the traffic volume is the average of the two arc values. In practice, these two arcs usually have the same length and very similar traffic volume (as can be observed in Figure 6.1a) which justifies this approach.

While the constraints on a possible section are clear, the objective of homogeneity needs further specification. In Section 4.1, we already discussed different methods to model homogeneity for vertex covering problems. Instead of transforming the measures to the edge covering case, we simply transform our problem onto the line graph. Now, each vertex v has a length w_v and an average traffic volume t_v, and we measure the traffic dissimilarity d_{uv} between two vertices u and v as $d_{uv} = |t_u - t_v|$. Let us reconsider the measures for homogeneity: The separation-based measures are not fitting because it is not important

(a) Actual network embedded in OpenStreetMap

(b) Network modeled as undirected graph

(c) Instance transformed to the line graph

Figure 6.1.: Modeling of the Toll Section Design Problem.

that a section's traffic is different to another section's traffic, but that it is internally homogeneous. Concerning the homogeneity-based measures, we dismiss the diameter and the radius since they are too vulnerable to single outliers and they do not differentiate between two sections with the same worst-case dissimilarity. Due to quadratically many summands, the clique measure penalizes larger sections unnecessarily hard. We are thus left with the star and sum-of-squares measure. The average traffic of a section and the quadratic penalization make the sum-of-squares measure hard to use in a MIP formulation.

The star objective, on the other hand, provides everything we want, especially in the weighted case: Two outliers are penalized more than one outlier, but few outliers (especially if they represent only a short subpart of the section) do not affect the overall penalty so much. It does not unreasonably penalize large sections and it is simple to model within an IP. Therefore, it is also widely used within the districting community to achieve compact ("round-shaped") districts. In this case, the district center, i.e., the node minimizing the cumulated distance to all vertices, is the geographically central vertex of the district. In Section 6.3, we will prove that with the star measure, the center of a section is the vertex with (weighted) median traffic within this section.

Figure 6.2.: Retransformed solution of the TSDP: An edge covering with homogeneous connected subgraphs within a given length range.

Before formally introducing the TSDP, we recall the result from Section 2.7 that it is sufficient to consider trees for the vertex covering, instead of arbitrary connected subgraphs.

Toll Section Design Problem (TSDP)

Instance: $G = (V, E)$, node weights $w, t \in \mathbb{R}^V_{\geq 0}$, and numbers $0 \leq W_L \leq W_U$

Problem: Find trees $T_1, \ldots, T_k \subseteq G$ for some $k \in \mathbb{N}$ with $W_L \leq w(T_i) \leq W_U$ for $i \in [k]$ and $V = \bigcup_{i=1}^{k} V(T_i)$ minimizing $\sum_{i=1}^{k} p(T_i)$ where

$$p(T) := \min_{u \in T} \sum_{v \in T} w_v \, |t_u - t_v| \qquad \text{for } T \subseteq G.$$

Given W_L and W_U, we use the notation $\mathcal{T}_{L,U}$ to denote the set of trees T in G with $W_L \leq w(T) \leq W_U$. Let us first settle the complexity question.

Complexity: The TSDP is NP-hard. A reduction from the number partitioning problem is straight forward. Given a multiset $S = \{s_1, \ldots, s_n\}$ of positive integers, the partitioning problem asks for a partition into two subsets S_1 and S_2 whose elements sum up to the same number. By considering a complete graph on S with vertex weights $w_i = s_i$ and arbitrary t, and by setting $W_L = W_U = \frac{1}{2} \sum_{i=1}^{n} s_i$, we can see that this TSDP instance has a feasible solution iff the partitioning instance is feasible.

Note that the reduction exploits that we can set $W_L = W_U$. While the general case remains open, we can prove NP-hardness under slightly modified conditions. If the cardinality k of a covering would be part of the input, we could reduce the bin packing problem to this TSDP version without the use of a lower weight bound W_L (cf. [CGP19]). This still does not take the special objective function into account. Let us therefore assume that the covering cardinality is not specified and that the dissimilarities are given as a matrix (d_{uv}) instead of the node induced dissimilarity. In this case, a reduction from bin packing is still possible. Given a bin packing instance with item weights $w_1, \ldots, w_n$ and bin capacity W, we consider the accordingly weighted complete graph and set $W_L = 0$ and $W_U = W$. By defining the dissimilarity matrix with values $d_{vv} = \frac{1}{w_v}$ and 0 off the diagonal, we attain that every subgraph has a penalty of 1. Hence, the objective of this TSDP is to minimize the number of covering trees, which is equivalent to the bin packing problem.

6.2 Column Generation (over Compact Model)

As our literature overview in Section 4.3 suggests, we can model the TSDP in different ways, namely, as a compact model or by dynamically generating feasible templates. In the following, we will detail both approaches with respect to the TSDP, and we will motivate our clear choice for the column generation approach.

6.2.1 Compact Model

As a compact model we could use the vertex-to-root formulation (VTR) and adapt it to the present problem. To this end, we have to add connectivity constraints and we have to incorporate our penalty function. Employing node separators for connectivity as in formulation (5.6), the resulting model could be written as follows.

$$\min_{x} \quad \sum_{r\in V}\sum_{v\in V} w_v \,|t_r - t_v|\, x_v^r \tag{6.1a}$$

$$\text{s.t.} \quad W_L\, x_r^r \;\le\; \sum_{v\in V} w_v\, x_v^r \;\le\; W_U \qquad \forall r \in V \tag{6.1b}$$

$$\sum_{r\in V} x_v^r \;\ge\; 1 \qquad \forall v \in V \tag{6.1c}$$

$$\sum_{w\in S} x_w^r \;\ge\; x_v^r \qquad \forall v \in V\setminus\{r\},\, \forall S \in \mathcal{N}(r,v),\, \forall r \in V \tag{6.1d}$$

$$x_v^r \;\in\; \{0,1\} \qquad \forall v, r \in V \tag{6.1e}$$

The binary variable x_v^r indicates if vertex v belongs to the template rooted at node r, and constraints (6.1c) ensure that every node is covered by a template. The lower capacity constraints (6.1b) should only be enforced if r is actually a center of a chosen template, and hence, the multiplication with x_r^r is necessary. In this instance, connectivity is modeled with node separator constraints (6.1d).

However, there are some issues with this formulation. First, it prevents two templates from having the same center. While this should not play a significant role for practical purposes, it can lead to suboptimal coverings. We could circumvent this by introducing multiple variables per possible root, but this blows up the formulation to impractical sizes, and we do not even know how many "copies" per root we would need. Second, and far more problematic, we lose the possibility to fix a huge portion of all variables due to symmetry as it was possible in the original VTR formulation. Note that due to our special penalty function which is reflected in the objective (6.1a), the root of a template is not easily exchangeable anymore.

In order to assess its practical performance, we implemented the compact model described in (6.1). As for the node separators for the pricing problem, we use the separation routine by Fischetti et al. [Fis+17] which only checks integer solutions, and we use back cuts. Our computational experiments on a sample set of instances confirm the poor performance of the compact model, compared to the column generation approach (also with node separators for connectivity). Instances that are solved within few seconds with the column generation approach take minutes with the compact model.

It is worth noting, however, that the results from the preprocessing described in Section 5.4 carry over to this formulation. More specifically, for a fixed root r, we can easily model discarded nodes ($x_v^r = 0$), fixed nodes ($x_v^r \ge x_r^r$), or conflict pairs ($x_v^r + x_w^r \le 1$).

6.2.2 Column Generation

A different model that we already stated in Section 4.3 uses the classical covering IP formulation. Instead of restating it here (see (4.1) if necessary), let us steer the presentation towards the column generation approach for this problem. With $p_T = p(T)$ we denote the inhomogeneity penalty for template T and $\mathcal{T}_{L,U}$ denotes the set of all feasible templates. The LP relaxation of (4.1) is called the master problem (MP), but here, we consider a subset $\mathcal{T}'_{L,U} \subseteq \mathcal{T}_{L,U}$ and restrict the (relaxed) covering problem to use only templates from $\mathcal{T}'_{L,U}$. This is called the restricted master problem (RMP) and can be stated as follows.

$$\min_{x} \quad \sum_{T \in \mathcal{T}'_{L,U}} p_T \, x_T \tag{6.2a}$$

$$\text{s.t.} \quad \sum_{T \ni v} x_T \;\geq\; 1 \qquad \forall v \in V \tag{6.2b}$$

$$x_T \;\geq\; 0 \qquad \forall T \in \mathcal{T}'_{L,U} \tag{6.2c}$$

Note that we can relax $x_T \in \{0,1\}$ to $x_T \geq 0$, since we have $p_T \geq 0$ and a minimization problem. When introducing dual variables (π_v) for the constraints (6.2b), we obtain the following restricted dual LP.

$$\max_{\pi} \quad \sum_{v \in V} \pi_v \tag{6.3a}$$

$$\text{s.t.} \quad \sum_{v \in T} \pi_v \;\leq\; p_T \qquad \forall T \in \mathcal{T}'_{L,U} \tag{6.3b}$$

$$\pi_v \;\geq\; 0 \qquad \forall v \in V \tag{6.3c}$$

Given an optimal solution x of (RMP) and corresponding dual values π, the reduced costs for a template T are therefore $p_T - \sum_{v \in T} \pi_v$. We know from duality theory that no template $T \in \mathcal{T}'_{L,U}$ has negative reduced costs. If we can show the same for all $T \in \mathcal{T}_{L,U}$, we can deduce that x is even optimal for (MP). If, on the other hand, we find a template $T \in \mathcal{T}_{L,U} \setminus \mathcal{T}'_{L,U}$ with negative reduced costs, we add a corresponding variable to (RMP) and iterate. As $|\mathcal{T}'_{L,U}|$ increases with every iteration (also called pricing round), this process is finite and eventually leads to an optimal solution for the master problem. Since the newly introduced variables are additional columns of the constraint matrix, this process is called column generation. The task to check if there are templates with negative reduced costs is called the pricing problem (PP). In our case, the corresponding optimization problem is

$$\min_{T \in \mathcal{T}_{L,U}} \; p_T - \sum_{v \in T} \pi_v. \tag{6.4}$$

As it is usual when employing a column generation approach, we not only add the single template that minimizes (6.4) but all templates with negative reduced costs that are found en route. While this increases the size of the RMP, it potentially reduces the number of pricing rounds that are necessary. For all of our tests, the solution time for the RMP is negligible compared to the optimization time of the PP.

Before we dive deeper into the pricing problem, let us briefly discuss the initial covering, i.e., the choice of $\mathcal{T}'_{L,U}$. Here, we construct the initial covering with an adapted depth-first search. Starting the DFS at a leaf (if possible), we create a template as soon as the lower weight bound is met. Then, we proceed the DFS from the last node but preferentially explore nodes that are not yet covered by any template. Again, whenever the lower weight bound is reached, we create a new template and iterate. This basic approach clearly yields a vertex covering with connected templates that satisfy the lower weight bound. In practice, the upper weight bound is also respected because the difference $W_U - W_L$ is larger than the sum of any two vertex weights. More sophisticated approaches for an initial covering are discussed in Section 6.6 but overall, these are not better than the described DFS covering.

To close this section, we emphasize that the column generation approach does not necessarily provide an optimal solution to the TSDP. First, the LP (6.2) can obviously have a fractional optimal solution which is not valid for the TSDP. A natural extension is to solve the respective IP version with all templates that are part of the final LP. This provides a feasible – however only heuristic – solution to the TSDP, as the column generation only guarantees optimal columns for the LP. In order to obtain a provably optimal solution to the TSDP, we have to extend the column generation to a branch-and-price approach. Indeed, we follow this path in Section 6.5 but find that the column generation heuristic essentially suffices for our problem.

6.3 Solving the Pricing Problem

Let us take a closer look at the pricing problem. We are now searching for a single feasible template that minimizes a combination of its penalty and the dual values on the vertices. Resubstituting for p_T yields

$$\min_{T \in \mathcal{T}_{L,U}} \left(\min_{u \in T} \sum_{v \in T} w_v \, |t_u - t_v| \right) - \sum_{v \in T} \pi_v.$$

We can tackle this problem in two different ways. The vertex u that serves as center for the template can either be variable, or we fix the center u and consider only templates for which u is the center. Again, we precompute all differences $d_{uv} := |t_u - t_v|$.

Variable Center Formulation: The advantage of this method is that we can solve the pricing with a single IP model. The drawback, however, is that the penalty term in our objective function requires us to indicate if a node v is chosen while the node u is the center of the resulting template. We can model this with a quadratic objective and use binary variables for the nodes of the chosen subgraph and other binary variables that indicate the single center. Equivalently, we can linearize the quadratic term from the objective and obtain quadratically many variables. This results in a formulation similar to the compact model (6.1) but with three major differences: First, the objective is different, and we use

$$\min_{x} \quad \sum_{r \in V} \sum_{v \in V} w_v \, d_{uv} \, x_v^r \; - \; \pi_v \, x_v^r. \tag{6.5}$$

On the other hand, we have to drop the covering constraints (6.1c) as we are only looking for a single template. Finally, the negative coefficients in the objective lead to the necessity of additional constraints that prevent the activation of too many roots or vertices. To account for this, we add the following constraints.

$$\begin{aligned} \sum_{r \in V} x_r^r \; &= \; 1 \\ \sum_{v \in V} x_v^r \; &\geq \; x_r^r \qquad\qquad \forall r \in V \end{aligned}$$

The latter constraints ensure that a root is active when another vertex is assigned to it, and the first constraint enforces that only one root is chosen.

Fixed Center Formulation: A second variant to solve the pricing problem is to consider the problem for fixed centers (also called roots). As we will see, this significantly simplifies the formulation. The question is, if it is better to solve one large optimization problem or many smaller problems, i.e., one for every vertex.

If we fix a vertex r to be the (potential) root of the template, we no longer have a quadratic term in the objective and, instead, obtain the following simpler IP formulation where we use the r-tree polytope $\mathcal{Y}_r$ from Chapter 5 for the rooted connectivity.

$$\min_{y} \quad \sum_{v \in V} (w_v \, d_{rv} \; - \; \pi_v) \; y_v \tag{6.6a}$$

$$\text{s.t.} \qquad W_L \; \leq \; \sum_{v \in V} w_v \, y_v \; \leq \; W_U \tag{6.6b}$$

$$y \; \in \; \mathcal{Y}_r \tag{6.6c}$$

The greatest difference in comparison to the variable center approach is clearly that we get rid of the quadratically many variables, and focus on the y variables that determine the subgraph. In addition, the objective (6.6a) is also simpler, as it contains only $|V|$ coefficients in contrast to the $|V|^2$ coefficients from (6.5). The combination of these two simplifications makes the problem much more amenable. Indeed, we found that it is far superior to solve a rooted IP (6.6) for every vertex, instead of solving the single, more complex IP of the variable center formulation. Therefore, let us further investigate the rooted formulation (6.6).

By defining $c_v := \pi_v - w_v \, d_{rv}$ and by replacing the objective (6.6a) with $\max_y \sum_{v \in V} c_v \, y_v$, we can transform the problem (6.6) into a maximization problem. Now, we are seeking a connected subgraph of maximum weight c that contains some root node r and that is capacitated from below and above with respect to vertex weights w. Note that c can take negative values and, hence, this problem is already very close to the BRCMWCS that we studied in the previous chapter.

6.3.1 Transformation into BRCMWCS

Let us take a closer look at the pricing problem with a fixed root r. Recall that r is the root of a template T iff $r \in \arg\min_{u \in T} \sum_{v \in T} w_v \, |t_u - t_v|$. We will proceed to show that r has the weighted median traffic in T.

Definition 6.1 (weighted median) *Given elements $t_1 < \cdots < t_n$ with positive weights $w_1, \ldots, w_n$ and $\Sigma_w := \sum_{i=1}^{n} w_i$, element t_k is a weighted median of (t, w) if*

$$\sum_{i=1}^{k-1} w_i \; \leq \; \frac{\Sigma_w}{2} \quad \textit{and} \quad \sum_{i=k+1}^{n} w_i \; \leq \; \frac{\Sigma_w}{2}.$$

As for the median of an even number of values, the weighted median can be not unique. If one of the inequalities is tight, there are two consecutive elements that satisfy the condition. In this case, we accept both elements as a weighted median. However, we do not accept any value in between the two. In particular, we do not take the arithmetic mean of both values as the weighted median. Consequently, we have the following.

Proposition 6.2 *Let $t_1 < \cdots < t_n$ and $w_1, \ldots, w_n > 0$.*
Then, t_u is a weighted median of (t, w) iff $u \in \operatorname{arg\,min}_{u \in [n]} \sum_{v \in [n]} w_v \, |t_u - t_v|$.

Proof. Suppose that t_u is a weighted median of (t, w) and consider some $j \in [n]$, w.l.o.g. $j < u$. Define $\Delta := t_u - t_j$ and note that $\Delta > 0$. Now,

$$\begin{aligned}
&\sum_{v \in [n]} w_v \, |t_j - t_v| - \sum_{v \in [n]} w_v \, |t_u - t_v| \\
&= \sum_{v=1}^{j} w_v \, (t_j - t_u) + \sum_{v=j+1}^{u-1} w_v \, (2t_v - t_j - t_u) + \sum_{v=u}^{n} w_v \, (t_u - t_j) \\
&\geq -\Delta \sum_{v=1}^{j} w_v - \Delta \sum_{v=j+1}^{u-1} w_v + \Delta \sum_{v=u}^{n} w_v \\
&= \Delta \left(\sum_{v=u}^{n} w_v - \sum_{v=1}^{u-1} w_v \right) \geq 0,
\end{aligned}$$

proving the first implication. For the reverse, note that if t_j is not a weighted median of (t, w), the first inequality is strict. □

While the demands (t_v) of a given TSDP instance are not necessarily pairwise different, we can transform the instance to an equivalent one with pairwise different demands: Whenever two demands are equal, we modify one of the two by a small ε. If those changes are sufficiently small, we know that an optimal solution with the adjusted demands is also optimal with respect to the original demands.

Also note that the dual values in the pricing objective do not affect the root of the resulting tree to have weighted median demand, as the total dual costs depend only on the resulting tree, and not on its root.

Now that we have established that the root node indeed has weighted median demand, the transformation into a BRCMWCS instance is possible. Recall the definition of this problem:

Balanced, Rooted, and Capacitated MWCS (BRCMWCS)

Instance: A graph $G = (V, E)$ with $V = V_b \dot\cup V_r$, node weights $w \in \mathbb{R}^V_{\geq 0}$ and $c \in \mathbb{R}^V$, as well as numbers $0 \leq W_L \leq W_U$, $\Delta \geq 0$, and a root node $r \in V$.

Problem: Find $V' \subseteq V$ such that $G[V']$ is connected with $w(V') \in [W_L, W_U]$, $r \in V'$, and $|w(V' \cap V_b) - w(V' \cap V_r)| \leq \Delta$ while maximizing $c(V')$.

While the graph G, the node weights w and c, and the root r directly carry over from the pricing problem for a fixed root, we have to specify the remaining ingredients. We use the node bipartition

$$V_b := \{v \in V \,:\, t_v \leq t_r\}, \qquad V_r := \{v \in V \,:\, t_v > t_r\}.$$

Note that the root is colored blue, but in order to not mess up the weighted median, we set $\Delta := w_r$ and then, $w_r := 0$. We reduce the lower and upper weight bounds W_L and

W_U by Δ to account for the fact that the root weight was set to 0. Now, it is easy to prove that r has weighted median demand in V' iff $|w(V' \cap V_b) - w(V' \cap V_r)| \leq \Delta$. Consequently, we can solve the pricing problem as a BRCMWCS for every node.

It is, however, not necessary to force the balancing condition. If we construct a feasible template T (feasible in terms of the TSDP) from r such that t_r is not the weighted median of the traffic values in T, then Proposition 6.2 gives us

$$\sum_{v \in T} w_v \, d_{rv} \; - \; \pi_v \; > \; p(T) - \pi(T).$$

The template T might still have a negative objective value (when considered from the "wrong" root r) and could be added to the RMP in this iteration. When considering the problem for the center of T as root, however, we are guaranteed to find T or another feasible template that has even lower reduced costs. To prevent the multiple addition of the same template in one pricing round, we maintain all templates in $\mathcal{T}'_{L,U}$ in a prefix tree.

Note that the preprocessing routines that we developed for the BRCMWCS are independent of c. Therefore, the preprocessing is independent of the dual values, and we can setup the pricing problem for each root in the beginning, and in each pricing round, we only have to adjust the coefficients in the objective function. In particular, this allows us to dismiss a considerable number of nodes as potential roots, since these will never have a weighted median demand in any feasible template. Detailed results on this are reported in Section 6.4.4.

Another advantage of the fixed center formulation is that it is simple to parallelize, as we can solve the pricing for each possible root independently. With increasing numbers of CPU cores, parallelization gains more significance. Since our pricing problem is a MIP that is already solved on multiple cores, we did not implement the parallel solving of different roots. For a large scale problem, however, this possibility is just another argument to use the fixed center approach over the variable center formulation.

6.3.2 Spanning Tree Heuristic

In the pricing problem, our goal is to find templates with negative reduced costs. There is no need, however, to find a template with minimal reduced costs. Only in order to prove that no template with negative reduced costs exists, we have to optimally solve the pricing problem. This insight gives rise to a common approach in column generation: We solve the pricing problem heuristically, add the newly found templates to the RMP, and iterate. Only if the heuristic does not find any templates of negative reduced costs, we have to perform a "full pricing round" where we apply the exact formulation.

Consequently, let us investigate possibilities for an efficient pricing heuristic. In Section 5.7, we have seen that the BRCMWCS on trees allows for a much simpler IP formulation for connectivity. While this approach works with any spanning tree, we choose one that is particularly suited for the median objective. To this end, we consider a bidirected version of G and set arc weights $\varphi_{u,v} := |t_v - t_r|$ for arc (u, v). The chosen spanning tree is now a shortest path tree from r with respect to φ.

We also experimented with different spanning trees. In particular, we used a BFS tree and also a shortest path tree from r with respect to the arc weights $\Phi_{u,v} = w_v \, |t_v - t_r| - \pi_v$. The latter, however, include dual values and, hence, the spanning tree is different in every pricing round and we have to rebuild the IP model. Without duals, on the other hand, we

only have to adjust the objective function. Also note that the inclusion of the duals can lead to negative arc weights (and cycles), and we have to adjust the shortest path computation. Our experiments show that these trees provide no advantage over the shortest path tree w.r.t. φ. We also followed the path of using multiple heuristics, i.e., if our standard heuristic does not find any templates, we divert to another spanning tree as specified above. However, this approach also showed no improvements as the second heuristic was not able to find new templates either.

6.3.3 Local Search

Recall that the reduced costs of subgraph S are $\overline{p}(S) := p(S) - \pi(S)$. We present a local search method that aims at quickly identifying subgraphs of negative reduced costs by adding or removing single nodes from an already promising subgraph. As such promising subgraphs we use all graphs that were found in the current pricing optimization, since these already have negative reduced costs. In addition, we perform the local search on all graphs that were used in the current solution of the restricted master problem (RMP), i.e., trees T with $x_T > 0$ in the optimal RMP solution. Note that due to complementary slackness, these subgraphs have reduced costs of 0.

In order to find good modifications of a subgraph S, we have to recompute reduced costs of slightly changed graphs, and are therefore particularly interested in the change of penalty. More specifically, given a tree $T \in \mathcal{T}_{L,U}$ with penalty $p(T)$ and a node v, we need to efficiently determine $p(T+v)$. Recomputing the weighted median and recalculating the weighted sum of differences is inefficient. With more stored information and suitable data structures, however, one can efficiently compute $p(T+v)$ from $p(T)$. We refrain from the technical details of this method, and instead focus on a different approach. For a tree T with weighted median traffic $\hat{t}$ and $v \notin T$, we define the potential of v as $\varphi(v) := w_v \, |t_v - \hat{t}|$, and note that

$$p(T+v) \;\leq\; p(T) + \varphi(v).$$

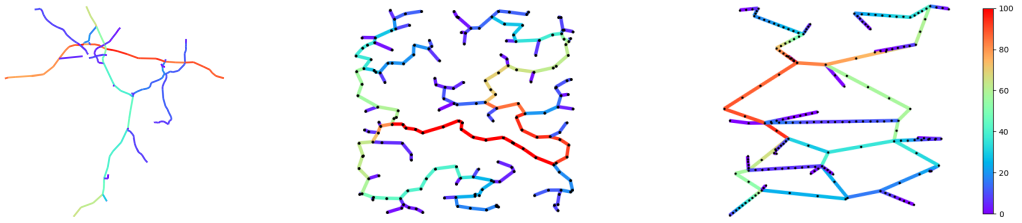
Thus, we overestimate the reduced costs as $\tilde{p}(T+v) := p(T) + \varphi(v) - \pi(T+v)$. The error of this estimation can be as large as $w(T)\,|t_v - \hat{t}|$, but shows to be small in practice if t_v is close to $\hat{t}$.

For the local search, we consider a tree T and two parameters k^+ and $k^- \in \mathbb{N}$. Now we iteratively determine the best k^+ neighbors of T w.r.t. $\tilde{p}$. For any of the 2^{k^+} subsets S^+, we check if $T + S^+$ is feasible, and after checking with a prefix tree that the same set of nodes is not already part of the template pool, we include it. Furthermore, we determine the best k^- nodes in T that can be feasibly removed from $G[V(T)]$ (independently). Again, for every subset S^-, we include any feasible new graph $T + S^+ - S^-$.

6.4 Computational Study

Our computational study serves two main goals: We compare the different connectivity formulations from Chapter 5 in the context of the column generation approach, and we measure the impact of three algorithmic enhancements that we proposed: The pricing heuristic, the local search, and the preprocessing from Chapter 5 applied to the pricing instances.

We start, however, with a detailed introduction of our test instance set, before we describe the test setting and our experiments in Section 6.4.2.



(a) German motorways, region around Hanover (b) Random tree (c) Voronoi network

Figure 6.3.: Exemplary transit network instances for the three instance classes.

6.4.1 Test Instances

The basis for our study are 24 real-world instances that constitute the German motorway network. These instances stem from the aforementioned research project on optimal truck toll control with the German Federal Office for Goods Transport. In order to have more instances, that can also be made publicly available, we analyzed the instances and generate two types of random transit networks with artificial traffic volume that resemble the real-world instances: random trees and random networks based on Voronoi diagrams. Typical instances for the three types are displayed in Figure 6.3. The code for our transit network generator is made publicly available in a GitHub repository[1] and the instances of this study are also available online[2].

As we transform the original instances to the line graph, we perform this transformation also for the generated transit networks. In order to stress this fact, we label all instances with `_lg`. Note that for trees, the line graph can lose the sparseness of the original network, as the extreme example of a star graph mapping to a clique shows. Since induced paths map to induced paths, however, the simple structure of the resulting `tree_lg` instances mostly prevails (cf. Table 6.1).

German motorways: The German motorway network is divided into 24 districts and each district is usually covered by 10 to 15 sections. We thoroughly analyzed these networks and their traffic distribution in order to generate realistic transit network instances. The subnetworks are sparse and often tree-like. Denser networks can be found in the Ruhr area, but are still planar and have a maximum degree of 4. As one would expect, the edge lengths practically satisfy the triangle inequality.

The distribution of trips is not uniform, i.e., not equal for each origin-destination pair. Instead, we found that it follows a shifted Pareto distribution. While for a detailed discussion, we refer the reader to [Sch18], we note here that this distribution is also known as the "80:20 rule" and is often used to describe the distribution of wealth in a society. With respect to the traffic distribution this roughly means that few origin-destination pairs make up for the majority of the traffic.

The `ger_lg` instances are grouped with respect to the magnitude of the computation time for the single-commodity flow formulation without any heuristics into four groups: tiny, small, medium, and large.

[1] https://github.com/stephanschwartz/transit_network_generator
[2] https://github.com/stephanschwartz/transit_network_instances

Table 6.1.: Selected properties of considered instance sets: # denotes the number of instances, tw the treewidth, $b\,[\%]$ the average fraction of edges that are bridges, and $p\,[\%]$ the fraction of instances that are planar. The values relate to the actual instances, i.e., the line graphs of the described transit networks.

Instance set	#	$\lvert V\rvert$	$\lvert E\rvert$	tw	$b\,[\%]$	$p\,[\%]$
ger_lg_tiny	4	100.8	116.5	3.0	52.8%	100.0%
ger_lg_small	9	154.3	193.0	3.3	38.7%	77.8%
ger_lg_medium	9	207.8	270.0	3.9	36.9%	22.2%
ger_lg_large	2	245.5	341.5	5.0	26.4%	0.0%
tree_lg	25	199.0	242.6	2.8	47.0%	100.0%
voronoi_lg_medium	9	203.3	241.7	4.0	37.6%	11.1%
voronoi_lg_large	41	205.8	262.8	4.7	26.2%	7.3%

Random trees: To mimic sparser instances of the German motorways, we consider 25 random trees. We build a tree from random points in a rectangle by using Kruskal's algorithm on the complete graph with Euclidean edge weights.

Concerning the artificial traffic on the network, we consider two possibilities: In a gravity model, we randomly choose a certain number of centers and assign a random population to each center. Also, each center increases the population of neighboring nodes. The demand of any origin-destination pair is now given by the product of the populations of the origin and the destination, and we assume a shortest path. Alternatively, we can use random traffic for each origin-destination pair. Starting from every node, we randomly choose a number of destinations and assign a random demand on the respective shortest path. In addition, we boost the traffic starting at leaf nodes to model the passage to neighboring networks. For the traffic demand in our transit networks, we use a combination of these two models. Thereby, we account for the typical dominance of certain origin-destination pairs as the Pareto distribution suggests and the results indeed look very realistic.

Voronoi networks: To mimic a denser transit network, we create Voronoi networks as already described in Section 5.6.1. The resulting networks truly resemble typical motorway networks and in order to add realistic traffic flow, we use a two-level approach. We consider two nested Voronoi graphs, where one layer represents a priority network where traveling is generally faster. For road networks, this is a differentiation between primary and secondary roads while for public transit networks this can model the difference between trains and buses. For the artificial traffic within the Voronoi networks we use the same combination of gravity model and random traffic as for the trees. However, the shortest paths are now computed with respect to the travel time which is assumed to be proportional to the length but faster on the priority level (in our case twice as fast). This leads to realistic results, as Figure 6.3 shows. We consider 50 of these Voronoi instances that are split into two groups based on the computation time of the SCF without any heuristics.

Table 6.1 lists some key characteristics of the different instance sets. The complexity of an instance depends on multiple factors. Apart from the graph size, the complexity seems to rise with increasing tree width, or decreasing fraction of bridges. Also, the problem seems more amenable on planar graphs.

Let us also briefly consider the distribution of node weights. Concerning the German motorway network instances, the mean weight is 7.7 with a standard deviation of 6.4. For the artificial instances, the mean node weight is 3.3 and the standard deviation is 1.6. For all instances, we set $W_L = 100, W_U = 200$.

6.4.2 Test Setting

For the experiments, we used the identical hardware as in Chapter 5, i.e., Intel Xeon E3-1245 CPUs with 3.70GHz and 32GB RAM. Again, we used Gurobi 9.5 with default settings[3] as LP and MIP solver and our code is written in Python 3.8. In contrast to the experiments from the previous chapter, we set a time limit of 24 hours and, again, instances with memory errors are set to the time limit.

For the fixed center pricing, we actually create BRCMWCS instances and therefore, almost all techniques discussed in Chapter 5 transfer to these experiments. Indeed, we strengthen the SCF and RNS formulations with z variables combined with indegree and 2-cycle cuts. Whenever we activate the preprocessing, we also determine implied nodes, extended indegree cuts (EIC-2) and root ring cuts and add those to the respective BRCMWCS. Note that all these concepts are independent of the profits and can be determined before the first pricing round. In each iteration of the column generation, we only have to adjust the objective of the BRCMWCS depending on the current dual values that are given by the RMP solution.

We performed numerous tests beyond the experiments reported here in order to find the best parameter setting for the pricing. These tests revealed that while the profit independent methods are beneficial, the other methods, i.e., fine path cuts and conflict pairs, are not helpful in the current form. Recall that we computed the fine path cuts for the ten nodes with highest profit. Already in Chapter 5, these cuts led only to a small improvement but for the experiments here, it is not worth investing the time of determining the cuts in each pricing round, as the problems are usually solved in fractions of a second. The situation is similar for the conflict pairs where the profits are used for a filtering of essential conflicts. We have seen that these were not even helpful for the BRCMWCS instances considered in the previous Chapter.

Our tests also revealed that it is better not to enforce the balancing condition. Therefore, our pricing problem is not really a BRCMWCS, but a rooted and capacitated MWCS. The best approach, however, is to treat the problem as a BRCMWCS for the preprocessing and the generation of implied nodes and root rings, and then drop the balancing constraints in the IP formulation. As discussed above, these constraints are not necessary to prove that no templates of negative reduced costs exist. In the same spirit, it is also slightly better to drop the color range constraints. We believe that in combination with the very important decision to add suboptimal templates that were found on the way to the optimum, this leads to the generation of more promising templates and, hence, to fewer pricing rounds.

Again, we stress that the solution time of the RMP is not relevant with regard to the total time and, consequently, our methods aim to facilitate the pricing process. The optimization procedure with algorithmic enhancements is depicted in Figure 6.4.

The focus of the experiments discussed next is the comparison of the connectivity models in the context of the TSDP and the evaluation of the impact of our enhancements.

[3] Obviously, we use the dual simplex method for the RMP.

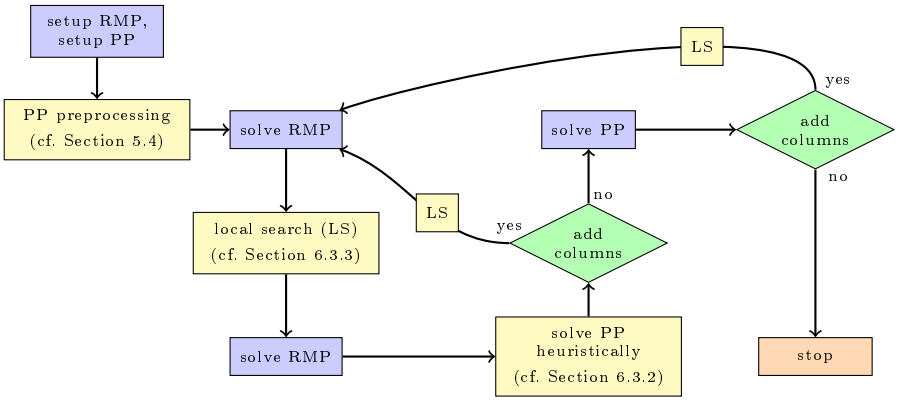


Figure 6.4.: Column generation procedure with all algorithmic enhancements (in yellow) integrated.

6.4.3 Comparison of Connectivity Models

We start by comparing the different connectivity formulations for the pricing problem in Chapter 5, but here, we report the total runtime of the column generation heuristic. The quality of the formulations can vary from the results for the BRCMWCS because, first, the actual pricing instances are different from the instances considered in Chapter 5. Second, we do not solve the actual BRCMWCS but drop the balancing constraint which can have a different effect on each connectivity formulation. And finally, we are not only interested in the optimal solution, but also use other feasible solutions found along the way. It is possible that one connectivity formulation finds more or better feasible solutions within the process than the other formulations.

Since we want to study the performance of the connectivity formulations, we disable the pricing heuristic which would otherwise dominate the solution process. However, we enable the median induced preprocessing and add further templates via local search. In addition, we compare the formulations in the best variant, i.e., with the pricing heuristic enabled. The results are given in Table 6.2.

Concerning the upper table, i.e., without the pricing heuristic, we see that overall, RAS is still the best formulation. However, the instance-wise results in Table B.1 show that for selected instances, the SCF or RNS have a significant advantage over the RAS. Detailed tests show that the runtime is mainly influenced by the number of pricing rounds and for a few instances, the SCF is able to exit the pricing loop earlier. Over the entirety of the test set, however, the RAS formulation performs best.

This is surprising when compared to the results from [BSS23] where the SCF and, in particular, the SCF+C2F formulation were superior to the RAS formulation. In that comparison, however, we did not consider z variables for SCF and RNS (but for SCF+C2F) as introduced in Section 5.5.3. Also, we did not include any strengthening cuts introduced in Section 5.5 and for RAS, we did separate for every vertex instead for every connected component. As shown in the computational study in Chapter 5, the new improvements essentially replace the C2F approach, and the extended indegree cuts as well as the separation per component provide a huge boost for the RAS. This explains why in our tests, the C2F approach loses its edge, and RAS is the new favorite.

Table 6.2.: Average computation times (in seconds) for different connectivity formulations without and with the inclusion of the pricing heuristic.

–PL

Instance set	RAS	+C2F	SCF	+C2F	RNS	MCF
ger_lg_tiny	0.7	0.9	0.9	0.9	0.7	2.9
ger_lg_small	4.8	6.4	6.0	6.1	4.1	36.7
ger_lg_medium	42.1	39.9	40.0	43.6	40.9	643.6
ger_lg_large	311.8	396.1	452.8	454.7	1408.6	46816.6
tree_lg	20.9	29.5	34.7	32.5	19.1	346.3
voronoi_lg_medium	171.3	248.0	161.2	178.3	155.2	4294.9
voronoi_lg_large	239.3	384.2	274.7	325.1	263.2	66815.1

HPL

Instance set	RAS	+C2F	SCF	+C2F	RNS	MCF
ger_lg_tiny	1.2	1.4	1.3	1.3	1.2	2.3
ger_lg_small	5.2	5.6	5.2	5.1	4.9	15.7
ger_lg_medium	24.8	30.5	25.5	24.6	28.6	235.1
ger_lg_large	105.0	122.9	111.2	110.6	108.3	43756.7
tree_lg	17.9	19.9	19.2	19.1	17.8	73.5
voronoi_lg_medium	66.3	88.8	66.5	65.3	63.1	758.4
voronoi_lg_large	92.0	146.2	93.9	96.5	90.6	2199.8

Very interestingly, the advantage of the RAS formulation vanishes if the pricing heuristic is activated, as shown in the lower table. Our extended tests show that for the vast majority of instances, the number of full pricing rounds as well as the average time per full pricing is similar for the RAS, SCF, and RNS formulations. When considering that RAS was the clear champion for the BRCMWCS, it is indeed surprising that for the column generation approach with optimal settings, all three formulations, RAS, SCF, and RNS, are on the same level.

6.4.4 Algorithmic Enhancements

All of the applied techniques aim to reduce the time spent for the pricing optimization. Be it with fewer full pricing rounds by the spanning tree heuristic, by the local search to reduce the number of pricing rounds, or by the preprocessing on the pricing instances that reduces the problem sizes considerably. Here, we measure the impact of the different improvements. Table 6.3 provides the respective computation times for the column generation heuristic for the RAS and SCF formulation. Most remarkable is that the three improvements are essentially independent, i.e., the reduction effects stack up if combined.

Pricing Heuristic: The aim of the pricing heuristic is to reduce the number of calls to the exact formulation and, instead, find templates of negative reduced costs heuristically.

Table 6.3.: Average computation times (in seconds) for the RAS and SCF formulation when including combinations of the pricing heuristic (H), the preprocessing (P), or the local search (L).

RAS

Instance set	---	--L	-P-	H--	-PL	H-L	HP-	HPL
ger_lg_tiny	5.1	2.8	2.4	2.5	1.4	2.1	1.4	1.2
ger_lg_small	50.8	24.5	14.0	24.1	7.3	10.3	8.4	5.2
ger_lg_medium	388.8	192.5	103.6	129.2	52.8	52.3	39.5	24.8
ger_lg_large	2549.6	1274.1	886.8	839.3	363.2	236.6	293.0	105.0
tree_lg	143.6	73.2	67.0	54.8	30.5	35.1	29.1	17.9
voronoi_lg_medium	846.2	466.4	287.9	147.7	198.3	78.5	79.5	66.3
voronoi_lg_large	1292.3	686.6	485.2	273.9	271.0	115.5	125.1	92.0

SCF

Instance set	---	--L	-P-	H--	-PL	H-L	HP-	HPL
ger_lg_tiny	5.5	3.5	2.6	2.6	1.6	2.4	1.5	1.3
ger_lg_small	45.0	22.2	17.0	19.7	8.6	10.3	9.5	5.2
ger_lg_medium	368.9	177.4	111.8	101.6	50.4	55.2	43.6	25.5
ger_lg_large	3835.2	2043.5	1087.0	589.0	500.9	218.6	278.2	111.2
tree_lg	283.9	154.5	94.1	59.6	44.6	43.0	25.8	19.2
voronoi_lg_medium	852.0	596.4	299.1	146.7	183.9	83.1	81.0	66.5
voronoi_lg_large	1872.8	1186.9	518.0	326.1	303.8	123.6	134.6	93.9

Indeed, the effect is just as desired: For the arc separator formulation, we reduce the number of full pricing rounds by 72% for ger_lg, by 92% for tree_lg, and by 80% for voronoi_lg. In many cases, the heuristic pricing already leads to the optimal solution. In 29% of the ger_lg instances, in 100% of the tree_lg instances, and in 12% of the voronoi_lg instances, the exact optimization was only called once to prove optimality. Our comparison shows that the integration of the pricing heuristic has the largest overall impact of the three improvements.

Preprocessing: With the fixed center formulation, we essentially solve a BRCMWCS for every vertex of the graph in a single pricing round. The preprocessing routine in Section 5.4 that exploits that the center (or root) of a template has weighted median traffic within the template has three potential benefits: First, it can eliminate potential roots by proving that no balanced connected subgraph within the capacity range can be constructed from this root. Therefore, this node cannot be the center of a feasible template and we can ignore the node as potential root in the pricing problem. Second, for the remaining roots, it can also shrink the relevant subgraph by discarding irrelevant nodes, leading to smaller pricing problems. And third, it can fix nodes to further simplify the pricing problems.

Table 6.4 shows the effect of the preprocessing on these three aspects. On average, our algorithm is able to exclude more than every third vertex as potential root for the

Table 6.4.: Effects of the median induced preprocessing.

Instance set	roots eliminated	discarded	fixed
`ger_lg_tiny`	40.0 %	40.8 %	18.1 %
`ger_lg_small`	40.8 %	37.8 %	13.4 %
`ger_lg_medium`	37.7 %	30.3 %	7.0 %
`ger_lg_large`	27.8 %	12.4 %	1.1 %
`tree_lg`	34.9 %	38.5 %	9.6 %
`voronoi_lg_medium`	23.8 %	4.5 %	1.7 %
`voronoi_lg_large`	19.9 %	4.2 %	1.1 %

`ger_lg` and `tree_lg` instances, and over 20% of all vertices for the `voronoi_lg` instances. While this alone leads to a significant reduction in computation time, the routine discards a considerable number of nodes in the relevant subgraph for the `ger_lg` and `tree_lg` instances, and is also able to fix a fair amount of the remaining nodes. Concerning the `ger_lg` instances with RAS, we observe that the preprocessing even has a larger effect than the pricing heuristic.

Local Search: The local search aims at generating promising template graphs in order to reduce the number of iterations in the column generation. For the model introduced in Section 6.3.3 we opt for $k^+ = 6$ and $k^- = 1$, even though, for large problems, the choice of a larger k^+ , e.g., $k^+ = 10$, was even better. Furthermore, our computations show that it is most beneficial to apply the local search after solving the restricted master problem (for each subgraph used in the optimal solution) and after the pricing (for each subgraph with negative reduced costs). With this setup, the number of iterations drops by 62% for the `ger_lg` instances, by 31% for `tree_lg`, and by 49% for the `voronoi_lg` instances.

In conclusion of the computational study, we can state that all of the proposed enhancements massively reduce the computation times. The choice of the connectivity model (among RAS, SCF, and RNS) is not decisive anymore when the preprocessing and the strengthening from Chapter 5, the pricing heuristic, and the local search are used. This study, however, was only concerned with the column generation heuristic which solves the TSDP restricted to all columns that were generated for its LP relaxation.

6.5 Branch-and-Price Approach

To assess the quality of the column generation heuristic, we expand it to a branch-and-price approach. Our branching is along the decision, whether a specific template $T \in \mathcal{T}_{L,U}$ is in the solution or not. The resulting changes of the pricing problem are marginal. Since with the condition $x_T = 1$ all vertices in T are already covered, the corresponding covering constraints (6.2b) are obsolete. Therefore, we set the respective dual variables to 0. Note that this adjustment only changes the objective function of the pricing problem. With the condition $x_T = 0$, on the other hand, we have to make sure that T is not a feasible solution

of the pricing problem anymore. We achieve this by including the constraint

$$\sum_{v \in V(T)} y_v - \sum_{v \in N(T)} y_v \leq |V(T)| - 1,$$

where $N(T)$ are the neighbors of T in G. This assures that if all vertices of T are chosen, at least one vertex outside of T is chosen as well.

In the branch-and-bound tree, we repeatedly select the node with the smallest objective value, apply our branching, and solve both of the resulting subproblems. We employ the most fractional branching, i.e., we select a template T with corresponding value closest to 0.5 in the current solution and consider the two child nodes with $x_T = 0$ and $x_T = 1$, respectively.

We implemented the described approach and tested it on all of our instances with the RAS formulation for connectivity, all algorithmic enhancements enabled, and a time limit of 24 hours. The results are presented in Table 6.5. A first observation is that many instances only need few branching steps to prove optimality. We point out that the averages for the number of branching nodes as well as the total branch-and-price time are heavily influenced by few outliers. Again, we refer to the appendix (Table B.6) for detailed results. We can also observe that the columns from the root relaxation already allow for an excellent solution of the TSDP. As the last column shows, for 82 out of 99 instances, all optimal templates for the TSDP are generated within the root LP; in 47 of these cases, the optimality could even be proven in the root node. The listed gap is between the (best) IP objective after the branch-and-price (or, respectively, with all columns generated in the root) and the best LP objective after the branching. We see that the dual gaps are very small over all instances. With respect to the root columns, the worst gap is at 2.3% and for 94 out of the 99 instances, our solution has a gap of less than 1%. These results substantiate the strength of the formulation.

We also experimented with a different branching strategy that differentiates if two adjacent vertices belong to a common template or not. While this approach proved to be superior for the partitioning case (which is discussed hereafter), it has a major drawback for the covering problem: In contrast to the partitioning case, we cannot merge two vertices for one subbranch, as there may be optimal templates that contain only one of the two vertices, while another template contains both.

Table 6.5.: Computational results for the branch-and-price approach.

		Time [s]		Gap		
Instance set	nodes	B&P	root	B&P	root	# opt root
`ger_lg_tiny`	3.8	5.5	1.2	0.0%	0.0%	4/4
`ger_lg_small`	2.1	15.0	5.2	0.0%	0.0%	9/9
`ger_lg_medium`	67.2	2450.0	24.8	0.0%	0.0%	9/9
`ger_lg_large`	1.5	236.4	105.0	0.0%	0.0%	2/2
`tree_lg`	35.3	1377.1	17.9	0.0%	0.1%	21/25
`voronoi_lg_medium`	43.9	4014.9	66.3	0.0%	0.1%	8/9
`voronoi_lg_large`	81.6	9497.6	92.0	0.1%	0.2%	29/41

6.6 No Primal Heuristic for TSDP

Many problems immensely benefit from primal heuristics and, as we have seen in Section 6.4.4, this is also the case for our pricing problem. Therefore, it seems natural to explore a similar approach for the TSDP. Can we find a "good" initial covering to help the solution process?

One idea is related to the graph coarsening method that we briefly described in Section 4.2: Multiple vertices are contracted to a single vertex, until a sought number of vertices remains, or other optimization methods are applied to the coarsened graph [Bic11]. In our case, we can use the homogeneity for the coarsening. Whenever two adjacent vertices u and v have very similar traffic values t_u and t_v, they are likely put into the same template. Hence, we contract the edge uv to the new vertex x with $w_x = w_u + w_v$ and $t_x = \frac{w_u t_u + w_v t_v}{w_u + w_v}$. Note that these parameters do not affect the computation of a weighted median.

Such an approach was carried out in [Sur20]. The goal was to iteratively merge two sufficiently similar adjacent vertices, to obtain fairly homogeneous *snippets*. The iteration stops if there are no sufficiently similar neighbors left, or if the snippets would exceed a weight threshold after another merging. The covering problem is then solved on the coarsened graph which is significantly smaller than the original instance. Using the resulting templates as an initial covering for the column generation problem, however, did not prove to have any advantage over our heuristic covering described in Section 6.2.2.

While we know from duality theory that we have optimal templates when there are no templates with negative reduced costs, the converse is not true. In order to assess the potential of a primal covering heuristic, we used an optimal solution of each instance as initial covering. In addition, we used local search to add more "nearly optimal" templates to the template pool, once with our standard parameters (LS1) and once with an aggressive setting that generates even more similar templates (LS2). The results are discouraging, as Table 6.6 shows. In many instances, optimal templates at the beginning have no or even a negative impact on the running time. Contrary to our expectation, not even the number of pricing rounds is reduced dramatically: On average over the 99 instances, it decreases from 16.7 rounds to 15.4 with LS1, while 25 instances even need more pricing rounds than without the optimal solution given initially. Now, if the optimal solution is not a good starting point for the search, then it seems pointless to invest any more capacities into finding a good covering heuristically and, hence, the title of this section is justified.

Table 6.6.: Effects of different initial coverings on optimization time (in seconds).

Instance set	SCF	optimal, LS1	optimal, LS2
`ger_lg_tiny`	1.3	1.2	1.3
`ger_lg_small`	5.2	4.9	5.7
`ger_lg_medium`	25.5	26.2	30.6
`ger_lg_large`	111.2	125.5	157.9
`tree_lg`	19.2	17.7	25.7
`voronoi_lg_medium`	66.5	57.6	62.0
`voronoi_lg_large`	93.9	87.7	98.3

6.7 From TSDP to Districting

To close this thesis, we demonstrate that our approach for the TSDP extends to general districting problems. The most important difference is the search for disjoint templates, i.e., districting is primarily concerned with vertex partitioning. Furthermore, the number of districts is usually given. We can adapt model (6.2) to account for these two changes: The covering constraints (6.2b) become partitioning constraints by replacing the inequality sign with an equality sign. This leads to dual variables without sign restriction, but the sign of the dual variables is irrelevant in the further course. Furthermore, we add the additional constraint

$$\sum_{T \in \mathcal{T}_{L,U}} x_T = k$$

in order to fix the number of templates. This constraint changes the objective function of the pricing problem, but when we consider a center inducing objective for the compactness of the districts (e.g., star, radius, or sum-of-squares), it remains a rooted and capacitated MWCS for every root in the fixed center formulation. Note, however, that the differences for the compactness measure are not node induced, but based on the Euclidean distance between two nodes. Therefore, we cannot apply our methods that exploit the balancing condition from Chapter 5, but whenever possible, we use their counterparts that are based solely on the capacity bounds. For instance, we compute root rings until the root component reaches the lower capacity bound, and add the corresponding root ring cuts. All methods with regard to the rooted and capacitated MWCS directly translate to the districting.

Publicly available districting instances are rare. We are aware of political districting instances considered by Validi and Buchanan [VB22], and the commercial territory design instances by Salazar-Aguilar, Ríos-Mercado, and Cabrera-Ríos [SRC11]. The former use the cut objective while the latter consider the star (and the radius) objective. Therefore, we detail the latter work that aims at partitioning a graph, representing a geographic area, into a fixed number of business districts. Each district needs to be connected and two capacity constraints with lower and upper bounds balance the number of customers and the sales volume among the districts. Both of these numbers, however, may only deviate by 5% from the average number of customers (or sales volume, respectively) per district.

The authors implement an exact and a heuristic model. The exact IP formulation, called Median-Based Territory Design Problem (MTDP), was already proposed but not implemented in [Seg+07]. The model is compact and uses a vertex-to-root formulation. The connectivity is ensured by a special set of node separator inequalities. As there are exponentially many of these inequalities, the MTDP is solved with branch-and-cut. Complementary to this approach, the authors propose a vertex-to-cluster formulation which becomes an integer quadratic problem because the center of each cluster has to be specified as well (see [SRC11] for details). The resulting model, denoted by QMTDP, is solved heuristically using DICOPT.

We apply our adapted approach to all large instances with 200 vertices from [SRC11]. A typical instance is depicted in Figure 6.5. The tight capacity bounds and the partitioning make it much harder to find a feasible solution as a base for the restricted master problem. To circumvent this problem, we make use of an artificial base by constructing an arbitrary partitioning of the vertices, which may lead to templates that do not meet the requirements

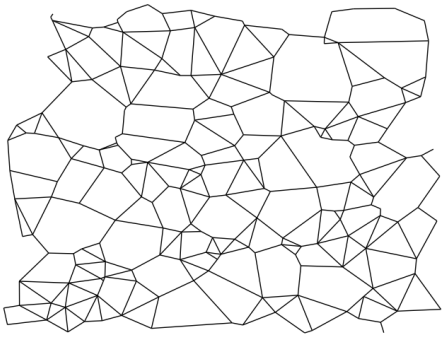


Figure 6.5.: Exemplary districting instance from [SRC11].

of a district. However, we assign prohibitively large penalties to these templates, so that none of them is part of an optimal solution. A similar method is used in [CGP19].

Since the integer version of the root LP has infeasibility issues, we use the described branch-and-price algorithm. In contrast to the covering case, however, the branching is done on the edges such that both endpoints are forced to be either in the same or in different districts. The advantage of the partitioning case is that we can either contract or delete the edge in the subsequent branch. We do not follow a particular rule but select any edge of an active template for the branching. After solving the LP relaxation of a branching node, we also solve the IP version of the restricted master problem with all previously generated districts. If we obtain an integer solution without templates from the artificial base we terminate. Analogously to the TSDP, this means that our approach is only heuristic but again, we find that the generated solutions are excellent.

We start with an "internal" comparison of our methods and different settings. While we considered the RAS, SCF, and RNS formulations with and without extended indegree cuts (of depth 2) and root ring cuts, we only report on a subset of the results in Table 6.7. Surprisingly, we find that the inclusion of extended indegree cuts has no positive effect.

Table 6.7.: Computation times (in seconds) for applying our heuristic approach to the instances from [SRC11] with different formulations and the inclusion of extended indegree cuts (+EIC-2) or root ring cuts (+RRC).

Instance	RAS	RAS +EIC-2	RAS +RRC	SCF +RRC	RNS +RRC
1	269.9	240.6	173.3	155.6	131.4
2	249.5	273.2	141.7	197.8	115.9
3	333.6	313.7	164.9	202.7	126.0
4	277.5	292.0	392.8	234.3	149.3
5	571.0	829.7	794.5	570.0	498.0
6	803.7	631.2	620.0	491.7	368.3
7	1017.2	711.4	371.9	1233.4	819.0
8	287.5	236.8	157.1	206.1	131.4
9	614.3	679.3	387.1	625.6	348.9
10	422.0	474.5	412.0	277.0	323.1

The root ring cuts, on the other hand, really facilitate the optimization process. This is not only the case for the RAS but also for the SCF and RNS formulation, although we decided not to include the respective optimization times in favor of a clearer presentation. We also find that the RAS and RNS formulations are best suited for the given instances, and that the times drastically vary for single instances.

Now we compare the RNS+RRC variant to the results from [SRC11]. We did not reimplement the MTDP and QMTDP, but take the values from the paper. Table 6.8 contains the objective values, the respective gaps to the MTDP objective, and the computation times for the approaches from [SRC11] and our variant (denoted as B&P). The time limit of two hours was introduced in [SRC11] but is not relevant for our heuristic approach. We observe that our transformed branch-and-price approach is far superior to the QMTDP heuristic in terms of quality and solution time. Note that for instance 9, the first solution of the B&P (found after a few minutes) is even better than the incumbent of the exact MTDP approach after two hours.

If we continue the branch-and-price procedure after the first solution, we find that closing the gap is an issue, just as for the TSDP. The results are still positive: For each of the six instances with a timeout for the exact approach, we are able to find better incumbent solutions within the same time with B&P. Let us exemplarily consider instance 5 where the first solution of the branch-and-price procedure is remarkably bad compared to the other instances. After 1000 seconds, we find a solution with smaller objective value than the QMTDP, and after 1500 seconds, we even dominate the incumbent of the MTDP after two hours.

With regard to the LP bound, already our heuristic solution is excellent as it is provably within 1% of the optimum for 7 of the 10 instances. At the end of the time limit, we have a gap of less than 1% for all 10 instances, and while it is even below 0.5% for 8 instances, we are able to prove the optimality only for instance 1. Unfortunately, we cannot compare our LP bounds to the ones of the MTDP (for timeout instances) since we did not reimplement the approach and the authors of [SRC11] do not report on the strength of their formulation.

Table 6.8.: Comparison of the exact (MTDP) and heuristic (QMTDP) solutions from [SRC11] to the first feasible solution of the presented branch-and-price approach on instances from [SRC11].

Inst	Objective value			Gap [%]		Time [s]		
	MTDP	QMTDP	B&P	QMTDP	B&P	MTDP	QMTDP	B&P
1	10422.0	11523	10422.0	10.6	0.0	1116	28	131
2	10646.1	11425	10657.9	7.3	0.1	7200	966	116
3	10846.8	11443	10880.9	5.5	0.3	1468	7200	126
4	11122.0	11443	11237.5	2.9	1.0	7200	3618	149
5	10878.1	11097	11374.6	2.0	4.6	7200	1193	498
6	10499.3	10746	10551.9	2.3	0.5	7200	1871	368
7	11061.0	11686	11276.4	5.7	1.9	7200	1088	819
8	10659.5	11205	10698.6	5.1	0.4	2641	592	131
9	11470.3	11648	11466.4	1.5	0.0	7200	1263	349
10	11043.8	11780	11044.8	6.7	0.0	1211	2349	323

The performance of our adapted branch-and-price-and-cut procedure is impressive, especially, if we consider that our approach is not designed for this particular problem. For example, we do not exploit the tight capacity bounds to exclude possible roots or reduce the problem size of each pricing instance. Also, the local search does not provide a large benefit, as we did not adapt it to promote feasibility w.r.t. the capacities. We strongly believe that such problem specific adaptions would lead to immense improvements for the considered districting problem. Finally, there is room for improvement concerning the branch-and-bound approach. A more sophisticated selection strategy and branching decision as well as an efficient implementation can boost the optimization as well.

But also without these adaptions and improvements, we are able to show that the newly proposed approach, i.e, a column generation where we split the pricing problem into many subproblems with fixed centers, can provide a great benefit for general districting problems. In particular, our approach is able to find excellent solutions very fast and is well suited for larger districting problems. We have seen that various methods that we developed with regard to the TSDP also carry over to this problem class. This shows that the contributions of this thesis are able to advance the state of the art in the growing field of districting problems.

A

Detailed Results for BRCMWCS

Welcome to the appendix of my thesis. In this first part, we provide instance-wise results for experiments performed in Chapter 5, i.e., concerning the BRCMWCS. In particular, we consider

- different connectivity formulations for the BRCMWCS:
  - for SCF, MCF, RAS, RNS as well as SCF+z and RNS+z in Table A.1,
  - for SCF*, MCF*, RAS*, RNS* in Table A.2,
- the impact of the coarse-to-fine approach:
  - for SCF, SCF+z, and SCF* in Table A.3,
  - for RAS and RAS* in Table A.4,
- the effect of adding conflict pairs:
  - for SCF and SCF* in Table A.5,
  - for RAS and RAS* in Table A.6,
- different variants of the optimal RAS formulation in Table A.7,
- the effect of the warm start heuristic in Table A.8.

There is no accompanying text to describe the results. Instead, we refer to these detailed results at several points in the thesis, e.g., to say that the computation times vary within a single instance set. We believe that you, dear reader, are able to interpret the tables by yourself. They are mainly provided for the sake of completeness, and to showcase the variation within the single groups.

Table A.1.: Computation times (in seconds) for different connectivity formulations in the basic variants and with z variables.

Instance	SCF	MCF	RAS	RNS	SCF+z	RNS+z
i–11	483.3	3600.0	10.0	3600.0	102.2	3600.0
i–12	958.2	3600.0	23.9	3600.0	165.0	3600.0
i–13	91.4	3600.0	14.1	3600.0	105.4	3600.0
i–14	151.3	3600.0	13.8	3600.0	79.0	3600.0
i–15	41.2	3600.0	9.3	3600.0	39.1	3600.0
i–16	63.0	3600.0	10.2	3600.0	102.2	3600.0
i–17	204.2	3600.0	16.3	3600.0	124.9	3600.0
i–18	131.0	3600.0	11.3	3600.0	97.2	3600.0
i–19	230.2	3600.0	8.0	3600.0	82.4	3600.0
i–21	64.5	1271.2	3.6	3317.0	87.0	3286.9
i–22	66.0	1880.2	3.5	1344.4	55.8	2814.9
i–23	34.6	2508.6	6.3	3600.0	10.2	3600.0
i–24	35.2	1473.0	6.5	3600.0	110.2	3600.0
i–25	34.3	3600.0	6.7	3600.0	23.9	3600.0
i–26	32.0	3600.0	5.1	1437.7	15.5	1519.9
i–27	132.9	3460.8	7.3	3600.0	78.6	3600.0
i–28	143.6	1476.0	8.3	3600.0	106.2	3600.0
i–29	144.0	3600.0	7.5	3600.0	60.1	3600.0
i–31	1817.2	3600.0	4.7	3600.0	75.2	3600.0
i–32	1367.0	3600.0	7.3	3600.0	108.5	3600.0
i–33	61.4	3600.0	9.3	3600.0	35.0	3600.0
i–34	485.7	3600.0	18.8	3600.0	113.1	3600.0
i–35	44.6	3600.0	11.8	3600.0	59.5	3600.0
i–36	80.1	3600.0	10.3	3600.0	114.1	3600.0
i–37	137.7	3600.0	8.8	3600.0	104.7	3600.0
i–38	244.5	3600.0	20.0	3600.0	110.4	3600.0
i–39	63.6	3600.0	7.2	3600.0	110.9	3600.0
i–41	182.2	3600.0	2.6	3600.0	69.9	3600.0
i–42	483.4	3600.0	7.0	3600.0	88.0	3600.0
i–43	65.0	2369.3	6.4	3600.0	41.4	3600.0
i–44	49.3	3600.0	9.2	3600.0	99.5	3600.0
i–45	42.1	2765.4	7.4	3600.0	66.6	519.5
i–46	37.8	2844.3	5.7	3600.0	25.7	2558.7
i–47	44.6	829.7	4.7	3600.0	68.4	742.3
i–48	350.5	3034.5	8.3	3600.0	126.9	3600.0
i–49	1367.0	3600.0	12.2	3600.0	146.6	3600.0
i–51	3600.0	3600.0	10.1	3600.0	291.5	3600.0
i–52	347.2	3600.0	4.9	3600.0	77.6	3600.0
i–53	86.0	3600.0	9.8	3600.0	90.0	3600.0
i–54	28.6	846.8	3.4	3600.0	19.6	3600.0
i–55	36.9	3600.0	7.6	3600.0	26.6	2448.4
i–56	33.1	3600.0	6.5	3600.0	34.2	536.6
i–57	212.9	2665.3	8.7	3600.0	87.3	3600.0
i–58	72.8	1816.2	3.6	3600.0	72.6	760.9
i–59	61.9	3350.2	6.2	3600.0	90.2	1159.4

Instance	SCF	MCF	RAS	RNS	SCF+z	RNS+z
l–1	3600.0	3600.0	662.2	3600.0	2430.6	3600.0
l–2	3600.0	3600.0	741.7	3600.0	3600.0	3600.0
l–3	3600.0	3600.0	477.8	3600.0	3600.0	3600.0
l–4	3600.0	3600.0	626.6	3600.0	1771.0	3600.0
l–5	3600.0	3600.0	228.3	3600.0	1167.7	3600.0
l–6	3600.0	3600.0	625.3	3600.0	2306.9	3600.0
l–7	3600.0	3600.0	259.8	3600.0	950.1	3600.0
l–8	3600.0	3600.0	909.8	3600.0	3600.0	3600.0
l–9	3600.0	3600.0	332.0	3600.0	1007.8	3600.0
l–10	3600.0	3600.0	698.8	3600.0	3600.0	3600.0
s–1	3600.0	1274.3	3.0	421.5	80.5	419.3
s–2	3600.0	3600.0	8.7	3600.0	189.7	3600.0
s–3	3600.0	3600.0	7.1	3600.0	94.8	3600.0
s–4	1.1	78.5	2.5	3600.0	2.8	2.3
s–5	3600.0	3600.0	4.1	3600.0	54.0	3600.0
s–6	3600.0	3600.0	6.5	3600.0	77.5	3600.0
s–7	3600.0	3600.0	10.0	3600.0	159.4	3600.0
s–8	3600.0	3600.0	5.0	3600.0	83.4	3600.0
s–9	3600.0	3600.0	8.5	3600.0	150.4	3600.0
s–10	3600.0	3600.0	4.5	3600.0	66.6	35.7
g–1	3600.0	3600.0	30.4	3600.0	1461.7	3600.0
g–2	3600.0	3600.0	32.6	3600.0	3600.0	3600.0
g–3	3600.0	3600.0	13.2	3600.0	1090.0	3600.0
g–4	3600.0	3600.0	13.9	3600.0	714.0	3600.0
g–5	434.8	3600.0	5.1	3600.0	84.7	3600.0
g–6	3364.8	3600.0	20.5	3600.0	227.1	3600.0
g–7	3600.0	3600.0	17.6	3600.0	546.9	3600.0
g–8	1451.0	3600.0	17.4	3600.0	198.4	3600.0
g–9	3600.0	3600.0	55.2	3600.0	2933.4	3600.0
g–10	3600.0	3600.0	13.6	3600.0	1432.8	3600.0

Table A.2.: Computation times (in seconds) for different connectivity formulations in the optimal variants.

Instance	SCF*	MCF*	RAS*	RNS*
i–11	20.8	3600.0	6.4	32.5
i–12	105.6	3600.0	17.4	285.1
i–13	20.3	3600.0	9.2	57.5
i–14	11.7	3600.0	5.7	39.0
i–15	5.0	2102.5	3.8	2.2
i–16	6.4	3600.0	7.8	26.2
i–17	30.2	3600.0	10.6	165.7
i–18	10.9	2511.0	6.7	7.8
i–19	5.8	2496.7	3.4	17.4
i–21	0.3	46.7	0.2	0.2
i–22	2.7	184.9	1.0	1.0
i–23	3.6	1268.6	1.7	28.2
i–24	3.4	1304.6	2.2	1.5
i–25	1.6	404.7	1.4	1.1
i–26	1.1	434.1	2.0	3.2
i–27	2.0	255.3	1.1	1.0
i–28	3.0	202.4	1.5	1.7
i–29	3.7	355.2	1.9	2.2
i–31	11.4	3600.0	4.0	42.1
i–32	56.6	3464.2	6.6	3600.0
i–33	4.6	3309.9	4.8	140.2
i–34	13.1	3600.0	7.7	3600.0
i–35	4.3	437.1	1.4	1.1
i–36	2.2	1008.7	1.5	5.3
i–37	2.5	497.3	1.8	1.5
i–38	6.5	940.1	1.8	3.7
i–39	3.4	1034.9	1.9	2.7
i–41	5.6	3446.9	2.9	2.4
i–42	6.0	3600.0	2.8	5.5
i–43	3.1	1600.1	1.8	2.5
i–44	8.0	3505.3	3.9	36.6
i–45	4.0	1615.0	2.4	1.9
i–46	2.7	1017.9	1.4	2.3
i–47	5.0	553.2	1.3	1.7
i–48	7.5	2147.2	4.7	6.2
i–49	17.1	3600.0	4.6	42.4
i–51	16.1	3600.0	4.0	148.2
i–52	10.6	3600.0	2.0	27.6
i–53	4.9	3600.0	2.2	21.9
i–54	3.8	980.5	1.2	1.6
i–55	2.5	615.4	1.8	3.0
i–56	2.3	1358.9	1.2	2.2
i–57	3.6	1231.8	1.6	3.3
i–58	3.5	467.3	1.4	2.2
i–59	3.6	492.4	3.6	1.1

Instance	SCF*	MCF*	RAS*	RNS*
l–1	90.9	3600.0	90.0	3600.0
l–2	186.6	3600.0	96.8	3600.0
l–3	444.7	3600.0	56.0	3600.0
l–4	155.9	3600.0	90.3	3600.0
l–5	139.9	3600.0	26.1	3600.0
l–6	137.5	3600.0	93.2	3600.0
l–7	25.6	3600.0	8.7	65.8
l–8	514.2	3600.0	113.4	3600.0
l–9	117.6	3600.0	30.9	3600.0
l–10	198.7	3600.0	84.9	3600.0
s–1	8.5	1107.8	2.9	2.2
s–2	3.4	3600.0	2.0	1.8
s–3	2.7	3600.0	1.8	1.4
s–4	3.1	68.7	2.2	2.0
s–5	3.7	3600.0	1.8	1.7
s–6	3.1	3600.0	1.4	1.4
s–7	5.0	3600.0	3.2	2.7
s–8	1.6	3600.0	1.3	1.4
s–9	2.8	3600.0	2.3	1.5
s–10	4.5	3600.0	1.5	0.9
g–1	438.8	3600.0	46.6	3600.0
g–2	347.8	3600.0	30.0	3600.0
g–3	461.7	3600.0	11.2	3600.0
g–4	226.5	3600.0	25.8	3600.0
g–5	60.7	3600.0	9.1	3600.0
g–6	126.8	3600.0	25.8	3600.0
g–7	353.5	3600.0	6.8	3600.0
g–8	126.4	3600.0	17.7	3600.0
g–9	1360.2	3600.0	46.8	3600.0
g–10	392.9	3600.0	21.2	3600.0

Table A.3.: Computation times (in seconds) for different variants of the SCF formulation without and with the coarse-to-fine (C2F) model.

Instance	SCF	+C2F	SCF+z	+C2F	SCF*	+C2F
i–11	483.3	314.8	102.2	101.9	20.8	28.8
i–12	958.2	1558.0	165.0	259.0	105.6	147.6
i–13	91.4	72.6	105.4	70.2	20.3	27.3
i–14	151.3	241.4	79.0	93.8	11.7	19.7
i–15	41.2	32.4	39.1	23.0	5.0	4.5
i–16	63.0	50.0	102.2	89.7	6.4	6.8
i–17	204.2	317.7	124.9	125.5	30.2	29.6
i–18	131.0	144.9	97.2	100.9	10.9	10.2
i–19	230.2	303.5	82.4	82.1	5.8	11.6
i–21	64.5	73.2	87.0	91.8	0.3	0.4
i–22	66.0	60.7	55.8	35.4	2.7	1.9
i–23	34.6	31.2	10.2	9.7	3.6	4.3
i–24	35.2	31.2	110.2	76.1	3.4	3.2
i–25	34.3	24.7	23.9	20.3	1.6	1.5
i–26	32.0	37.6	15.5	13.5	1.1	1.6
i–27	132.9	139.8	78.6	74.8	2.0	2.0
i–28	143.6	107.1	106.2	98.8	3.0	3.1
i–29	144.0	126.7	60.1	66.5	3.7	4.0
i–31	1817.2	1186.2	75.2	64.7	11.4	11.0
i–32	1367.0	1019.3	108.5	104.1	56.6	100.4
i–33	61.4	76.2	35.0	29.3	4.6	5.9
i–34	485.7	418.8	113.1	96.2	13.1	28.4
i–35	44.6	47.7	59.5	35.0	4.3	3.1
i–36	80.1	85.7	114.1	76.9	2.2	3.1
i–37	137.7	273.7	104.7	92.5	2.5	4.1
i–38	244.5	347.7	110.4	79.8	6.5	6.0
i–39	63.6	123.0	110.9	68.7	3.4	5.3
i–41	182.2	503.8	69.9	56.5	5.6	8.1
i–42	483.4	765.4	88.0	81.1	6.0	9.2
i–43	65.0	65.4	41.4	56.3	3.1	5.0
i–44	49.3	40.8	99.5	86.7	8.0	5.6
i–45	42.1	32.6	66.6	33.1	4.0	4.0
i–46	37.8	35.4	25.7	19.7	2.7	2.4
i–47	44.6	52.9	68.4	65.5	5.0	5.1
i–48	350.5	237.2	126.9	73.7	7.5	6.1
i–49	1367.0	854.0	146.6	119.5	17.1	27.4
i–51	3600.0	3600.0	291.5	211.2	16.1	29.7
i–52	347.2	430.1	77.6	83.8	10.6	9.2
i–53	86.0	100.9	90.0	65.7	4.9	7.8
i–54	28.6	31.2	19.6	14.2	3.8	2.9
i–55	36.9	30.3	26.6	22.0	2.5	3.2
i–56	33.1	39.1	34.2	30.5	2.3	2.8
i–57	212.9	146.2	87.3	76.7	3.6	3.4
i–58	72.8	63.6	72.6	89.1	3.5	5.5
i–59	61.9	49.8	90.2	56.0	3.6	2.4

Instance	SCF	+C2F	SCF+z	+C2F	SCF*	+C2F
l–1	3600.0	3600.0	2430.6	2722.5	90.9	1036.4
l–2	3600.0	3600.0	3600.0	3600.0	186.6	1588.8
l–3	3600.0	3600.0	3600.0	3600.0	444.7	1238.9
l–4	3600.0	3600.0	1771.0	2983.2	155.9	1224.4
l–5	3600.0	3600.0	1167.7	1504.8	139.9	248.3
l–6	3600.0	3600.0	2306.9	3600.0	137.5	1541.5
l–7	3600.0	3600.0	950.1	947.4	25.6	105.7
l–8	3600.0	3600.0	3600.0	3600.0	514.2	1711.9
l–9	3600.0	3600.0	1007.8	2626.8	117.6	823.2
l–10	3600.0	3600.0	3600.0	3584.7	198.7	1180.0
s–1	3600.0	15.3	80.5	4.0	8.5	3.4
s–2	3600.0	247.8	189.7	15.5	3.4	8.6
s–3	3600.0	128.4	94.8	11.6	2.7	4.8
s–4	1.1	1.7	2.8	1.2	3.1	1.2
s–5	3600.0	59.7	54.0	8.1	3.7	3.5
s–6	3600.0	85.8	77.5	5.6	3.1	3.1
s–7	3600.0	183.7	159.4	8.0	5.0	4.3
s–8	3600.0	113.7	83.4	8.0	1.6	3.6
s–9	3600.0	35.0	150.4	8.1	2.8	4.0
s–10	3600.0	6.7	66.6	2.9	4.5	2.5
g–1	3600.0	3600.0	1461.7	1226.2	438.8	324.3
g–2	3600.0	3600.0	3600.0	1123.0	347.8	425.4
g–3	3600.0	3600.0	1090.0	440.1	461.7	240.9
g–4	3600.0	3600.0	714.0	868.0	226.5	265.6
g–5	434.8	315.2	84.7	80.6	60.7	76.4
g–6	3364.8	2519.4	227.1	207.4	126.8	121.9
g–7	3600.0	3600.0	546.9	395.8	353.5	187.9
g–8	1451.0	1567.8	198.4	187.0	126.4	108.4
g–9	3600.0	3600.0	2933.4	3600.0	1360.2	1510.4
g–10	3600.0	3600.0	1432.8	643.3	392.9	539.6

Table A.4.: Computation times (in seconds) for different variants of the RAS formulation without and with the coarse-to-fine (C2F) model.

Instance	RAS	+C2F	RAS*	+C2F
i–11	10.0	20.0	6.4	10.1
i–12	23.9	55.1	17.4	26.1
i–13	14.1	33.3	9.2	8.5
i–14	13.8	17.9	5.7	8.3
i–15	9.3	13.3	3.8	4.2
i–16	10.2	9.7	7.8	5.4
i–17	16.3	17.7	10.6	10.9
i–18	11.3	13.2	6.7	6.6
i–19	8.0	5.0	3.4	4.1
i–21	3.6	3.4	0.2	0.3
i–22	3.5	2.8	1.0	1.8
i–23	6.3	5.7	1.7	1.8
i–24	6.5	5.0	2.2	3.3
i–25	6.7	7.5	1.4	1.9
i–26	5.1	6.4	2.0	1.8
i–27	7.3	5.8	1.1	1.3
i–28	8.3	5.6	1.5	2.1
i–29	7.5	8.1	1.9	2.9
i–31	4.7	8.4	4.0	3.7
i–32	7.3	6.2	6.6	8.1
i–33	9.3	9.7	4.8	6.2
i–34	18.8	16.1	7.7	9.7
i–35	11.8	11.8	1.4	1.8
i–36	10.3	10.3	1.5	2.0
i–37	8.8	7.2	1.8	3.1
i–38	20.0	9.8	1.8	3.2
i–39	7.2	6.8	1.9	3.1
i–41	2.6	4.6	2.9	3.4
i–42	7.0	5.9	2.8	3.3
i–43	6.4	6.3	1.8	2.7
i–44	9.2	6.5	3.9	2.7
i–45	7.4	6.1	2.4	3.4
i–46	5.7	7.2	1.4	2.7
i–47	4.7	3.1	1.3	1.6
i–48	8.3	8.7	4.7	4.9
i–49	12.2	9.5	4.6	6.0
i–51	10.1	11.2	4.0	13.8
i–52	4.9	2.5	2.0	2.6
i–53	9.8	8.5	2.2	5.5
i–54	3.4	5.8	1.2	2.9
i–55	7.6	6.2	1.8	1.8
i–56	6.5	6.4	1.2	2.5
i–57	8.7	5.6	1.6	2.5
i–58	3.6	3.8	1.4	1.4
i–59	6.2	5.6	3.6	2.2

Instance	RAS	+C2F	RAS*	+C2F
l–1	662.2	789.3	90.0	632.5
l–2	741.7	1172.5	96.8	885.1
l–3	477.8	464.8	56.0	738.2
l–4	626.6	866.5	90.3	735.9
l–5	228.3	283.8	26.1	139.6
l–6	625.3	684.7	93.2	619.5
l–7	259.8	130.4	8.7	31.1
l–8	909.8	1213.3	113.4	735.2
l–9	332.0	280.8	30.9	189.2
l–10	698.8	1133.7	84.9	649.8
s–1	3.0	3.1	2.9	5.1
s–2	8.7	5.2	2.0	7.2
s–3	7.1	4.4	1.8	4.6
s–4	2.5	2.9	2.2	1.9
s–5	4.1	4.0	1.8	5.3
s–6	6.5	3.4	1.4	3.5
s–7	10.0	5.0	3.2	5.5
s–8	5.0	3.6	1.3	5.2
s–9	8.5	2.7	2.3	6.8
s–10	4.5	1.6	1.5	2.0
g–1	30.4	30.2	46.6	32.5
g–2	32.6	30.2	30.0	33.7
g–3	13.2	12.6	11.2	11.0
g–4	13.9	20.4	25.8	19.8
g–5	5.1	4.9	9.1	5.7
g–6	20.5	17.0	25.8	35.5
g–7	17.6	9.7	6.8	11.1
g–8	17.4	17.8	17.7	21.9
g–9	55.2	54.5	46.8	36.7
g–10	13.6	19.2	21.2	29.8

Table A.5.: Computation times (in seconds) for the basic and optimal variants of SCF without and with the addition of (essential) conflict pairs.

Instance	SCF	SCF+CP	SCF*	SCF*+CP
i–11	483.3	481.1	20.8	23.2
i–12	958.2	738.2	105.6	138.9
i–13	91.4	63.5	20.3	15.5
i–14	151.3	113.3	11.7	12.9
i–15	41.2	29.8	5.0	5.4
i–16	63.0	51.1	6.4	6.0
i–17	204.2	211.2	30.2	30.4
i–18	131.0	60.9	10.9	12.0
i–19	230.2	114.3	5.8	9.5
i–21	64.5	13.1	0.3	0.4
i–22	66.0	28.4	2.7	1.9
i–23	34.6	28.6	3.6	4.1
i–24	35.2	29.0	3.4	5.4
i–25	34.3	24.6	1.6	1.9
i–26	32.0	21.9	1.1	1.3
i–27	132.9	21.9	2.0	1.6
i–28	143.6	56.5	3.0	2.8
i–29	144.0	41.1	3.7	3.3
i–31	1817.2	1385.1	11.4	7.5
i–32	1367.0	947.7	56.6	29.2
i–33	61.4	61.6	4.6	4.4
i–34	485.7	193.7	13.1	21.9
i–35	44.6	27.1	4.3	3.9
i–36	80.1	38.6	2.2	3.0
i–37	137.7	40.4	2.5	2.3
i–38	244.5	75.3	6.5	5.7
i–39	63.6	65.2	3.4	3.0
i–41	182.2	397.9	5.6	6.0
i–42	483.4	386.4	6.0	6.7
i–43	65.0	74.1	3.1	4.0
i–44	49.3	52.8	8.0	6.4
i–45	42.1	32.5	4.0	4.0
i–46	37.8	32.8	2.7	3.0
i–47	44.6	38.2	5.0	5.1
i–48	350.5	200.9	7.5	8.8
i–49	1367.0	731.0	17.1	20.3
i–51	3600.0	1811.6	16.1	23.0
i–52	347.2	210.8	10.6	13.5
i–53	86.0	76.3	4.9	5.4
i–54	28.6	25.3	3.8	3.2
i–55	36.9	23.1	2.5	4.5
i–56	33.1	21.9	2.3	2.3
i–57	212.9	61.4	3.6	3.6
i–58	72.8	85.9	3.5	4.4
i–59	61.9	26.6	3.6	3.1

Instance	SCF	SCF+CP	SCF*	SCF*+CP
l–1	3600.0	3600.0	90.9	119.4
l–2	3600.0	3600.0	186.6	538.5
l–3	3600.0	3600.0	444.7	989.3
l–4	3600.0	3600.0	155.9	192.2
l–5	3600.0	3600.0	139.9	95.1
l–6	3600.0	3600.0	137.5	334.4
l–7	3600.0	3600.0	25.6	37.4
l–8	3600.0	3600.0	514.2	1587.7
l–9	3600.0	3600.0	117.6	240.3
l–10	3600.0	3600.0	198.7	341.2
s–1	3600.0	3600.0	8.5	12.9
s–2	3600.0	3600.0	3.4	6.7
s–3	3600.0	3600.0	2.7	4.2
s–4	1.1	1.5	3.1	4.4
s–5	3600.0	3600.0	3.7	4.8
s–6	3600.0	3600.0	3.1	4.1
s–7	3600.0	3600.0	5.0	4.9
s–8	3600.0	3600.0	1.6	2.3
s–9	3600.0	3600.0	2.8	4.5
s–10	3600.0	3600.0	4.5	2.9
g–1	3600.0	3600.0	438.8	301.9
g–2	3600.0	3600.0	347.8	360.7
g–3	3600.0	3600.0	461.7	513.6
g–4	3600.0	3600.0	226.5	393.5
g–5	434.8	393.2	60.7	62.3
g–6	3364.8	2941.4	126.8	109.8
g–7	3600.0	3600.0	353.5	243.3
g–8	1451.0	1269.8	126.4	103.1
g–9	3600.0	3600.0	1360.2	868.4
g–10	3600.0	3600.0	392.9	459.6

Table A.6.: Computation times (in seconds) for the basic and optimal variants of RAS without and with the addition of (essential) conflict pairs.

Instance	RAS	RAS+CP	RAS*	RAS*+CP
i–11	10.0	7.8	6.4	9.3
i–12	23.9	22.8	17.4	22.1
i–13	14.1	16.4	9.2	7.9
i–14	13.8	17.4	5.7	6.3
i–15	9.3	10.6	3.8	3.9
i–16	10.2	14.2	7.8	5.6
i–17	16.3	34.7	10.6	9.0
i–18	11.3	12.2	6.7	5.3
i–19	8.0	6.7	3.4	3.9
i–21	3.6	0.9	0.2	0.2
i–22	3.5	1.4	1.0	1.0
i–23	6.3	7.1	1.7	3.6
i–24	6.5	6.2	2.2	2.7
i–25	6.7	5.6	1.4	1.9
i–26	5.1	3.8	2.0	2.2
i–27	7.3	3.7	1.1	1.2
i–28	8.3	4.9	1.5	1.4
i–29	7.5	5.7	1.9	2.4
i–31	4.7	3.8	4.0	5.1
i–32	7.3	5.5	6.6	6.0
i–33	9.3	6.3	4.8	4.7
i–34	18.8	16.6	7.7	8.1
i–35	11.8	3.7	1.4	2.3
i–36	10.3	6.6	1.5	1.7
i–37	8.8	3.7	1.8	1.6
i–38	20.0	7.0	1.8	4.3
i–39	7.2	5.7	1.9	1.9
i–41	2.6	4.5	2.9	3.0
i–42	7.0	5.2	2.8	3.6
i–43	6.4	6.8	1.8	2.2
i–44	9.2	5.7	3.9	3.5
i–45	7.4	6.0	2.4	2.1
i–46	5.7	6.2	1.4	1.5
i–47	4.7	3.5	1.3	1.4
i–48	8.3	9.2	4.7	4.0
i–49	12.2	9.6	4.6	5.7
i–51	10.1	8.4	4.0	5.1
i–52	4.9	5.3	2.0	2.3
i–53	9.8	9.2	2.2	4.9
i–54	3.4	5.0	1.2	1.1
i–55	7.6	2.5	1.8	1.5
i–56	6.5	4.9	1.2	1.9
i–57	8.7	2.5	1.6	2.1
i–58	3.6	4.2	1.4	2.4
i–59	6.2	4.1	3.6	2.5

Instance	RAS	RAS+CP	RAS*	RAS*+CP
l–1	662.2	855.1	90.0	93.0
l–2	741.7	785.0	96.8	88.4
l–3	477.8	882.6	56.0	90.2
l–4	626.6	470.3	90.3	90.2
l–5	228.3	345.4	26.1	64.2
l–6	625.3	828.0	93.2	173.8
l–7	259.8	372.6	8.7	10.6
l–8	909.8	849.2	113.4	150.8
l–9	332.0	554.7	30.9	74.9
l–10	698.8	794.7	84.9	80.3
s–1	3.0	6.0	2.9	3.4
s–2	8.7	11.6	2.0	2.4
s–3	7.1	7.5	1.8	1.8
s–4	2.5	2.8	2.2	2.3
s–5	4.1	4.8	1.8	2.1
s–6	6.5	8.1	1.4	1.8
s–7	10.0	10.2	3.2	3.9
s–8	5.0	6.0	1.3	2.3
s–9	8.5	8.3	2.3	2.7
s–10	4.5	5.2	1.5	1.2
g–1	30.4	21.5	46.6	32.7
g–2	32.6	24.6	30.0	24.9
g–3	13.2	11.5	11.2	13.7
g–4	13.9	18.3	25.8	18.8
g–5	5.1	8.4	9.1	5.2
g–6	20.5	18.2	25.8	20.8
g–7	17.6	6.8	6.8	8.2
g–8	17.4	13.9	17.7	20.0
g–9	55.2	50.3	46.8	46.1
g–10	13.6	15.8	21.2	24.5

Table A.7.: Computation times (in seconds) for different variants of the optimal RAS formulation

Instance	RAS*	–BC	–CR	+EIC-3	–EIC	–SpC
i–11	6.4	7.2	9.8	5.9	8.0	9.7
i–12	17.4	15.2	19.1	18.2	23.1	27.3
i–13	9.2	10.6	9.0	9.9	15.3	16.0
i–14	5.7	6.1	8.2	6.8	12.0	13.7
i–15	3.8	3.7	4.6	3.8	7.4	12.9
i–16	7.8	4.1	6.1	4.4	9.6	10.8
i–17	10.6	11.7	8.0	12.2	14.3	19.5
i–18	6.7	5.8	6.8	7.1	10.3	9.1
i–19	3.4	2.7	2.2	4.4	9.9	13.4
i–21	0.2	0.2	0.2	0.2	0.8	0.3
i–22	1.0	1.0	1.0	1.2	1.6	1.4
i–23	1.7	2.0	2.1	1.7	5.2	2.4
i–24	2.2	2.3	2.0	2.5	3.1	2.2
i–25	1.4	1.3	1.6	1.5	3.1	2.4
i–26	2.0	1.7	0.9	2.4	3.9	2.2
i–27	1.1	1.2	1.7	2.1	1.4	1.8
i–28	1.5	1.4	1.6	1.5	2.1	1.7
i–29	1.9	1.5	1.5	1.5	3.4	2.3
i–31	4.0	3.8	6.2	3.3	5.9	9.6
i–32	6.6	4.4	7.0	5.0	6.7	6.4
i–33	4.8	4.1	4.9	5.7	8.5	7.8
i–34	7.7	8.3	8.5	7.8	13.9	13.1
i–35	1.4	2.1	1.4	2.5	3.4	2.7
i–36	1.5	1.4	1.6	1.4	3.5	2.0
i–37	1.8	1.8	1.5	1.5	2.5	2.5
i–38	1.8	1.6	2.7	2.1	4.2	3.4
i–39	1.9	1.8	2.0	2.4	3.2	5.0
i–41	2.9	2.8	2.9	3.0	3.3	4.7
i–42	2.8	2.9	3.0	3.3	4.7	5.1
i–43	1.8	1.8	2.2	3.5	5.8	3.5
i–44	3.9	2.0	2.0	3.0	6.5	4.8
i–45	2.4	2.2	2.3	2.8	6.9	3.9
i–46	1.4	1.7	1.8	1.8	4.0	3.5
i–47	1.3	1.0	1.0	0.9	2.3	2.5
i–48	4.7	3.7	4.5	4.5	7.6	7.3
i–49	4.6	5.8	5.2	7.4	8.4	7.6
i–51	4.0	4.8	5.5	5.3	5.4	5.1
i–52	2.0	1.6	2.4	2.3	3.6	3.1
i–53	2.2	3.8	3.9	2.3	7.2	2.2
i–54	1.2	2.1	1.1	1.4	2.4	2.4
i–55	1.8	1.6	1.5	1.4	2.5	2.3
i–56	1.2	2.2	2.2	1.9	3.8	3.8
i–57	1.6	2.1	1.7	1.6	3.4	6.3
i–58	1.4	2.5	2.5	0.9	1.8	1.5
i–59	3.6	2.6	2.1	1.8	3.1	5.5

Instance	RAS*	–BC	–CR	+EIC-3	–EIC	–SpC
l–1	90.0	63.1	92.3	82.7	655.5	515.3
l–2	96.8	120.7	101.6	108.6	588.0	575.1
l–3	56.0	101.4	64.7	57.1	492.1	346.8
l–4	90.3	113.2	82.6	54.7	632.4	191.5
l–5	26.1	26.6	38.5	58.6	274.1	103.7
l–6	93.2	101.3	82.3	91.2	699.9	426.3
l–7	8.7	5.8	8.9	8.3	332.2	44.2
l–8	113.4	148.2	145.8	86.3	751.5	798.6
l–9	30.9	43.5	59.9	80.8	386.8	187.7
l–10	84.9	82.9	79.4	115.1	734.4	279.1
s–1	2.9	2.4	1.8	2.3	3.3	6.4
s–2	2.0	1.9	2.2	2.1	6.5	13.5
s–3	1.8	1.9	1.1	1.7	6.3	12.9
s–4	2.2	2.3	1.6	1.8	1.9	11.4
s–5	1.8	1.8	1.2	1.8	3.8	5.1
s–6	1.4	1.4	1.2	1.4	6.3	26.9
s–7	3.2	3.2	3.4	3.2	5.6	26.1
s–8	1.3	1.4	1.9	1.9	3.6	3.9
s–9	2.3	2.4	2.3	2.2	6.9	38.6
s–10	1.5	1.5	0.9	1.0	2.1	5.9
g–1	46.6	22.5	26.9	28.6	30.4	71.2
g–2	30.0	21.2	27.2	40.1	21.7	63.3
g–3	11.2	13.2	17.6	13.2	17.0	45.0
g–4	25.8	15.5	20.9	12.9	16.7	51.3
g–5	9.1	4.8	5.0	3.6	4.9	33.1
g–6	25.8	21.5	23.9	18.1	21.6	89.2
g–7	6.8	8.5	15.3	7.8	5.5	37.3
g–8	17.7	11.1	16.9	9.1	14.5	68.0
g–9	46.8	33.7	54.8	41.0	38.6	118.0
g–10	21.2	18.7	21.3	15.1	18.9	52.5

Table A.8.: Performance of the BFS heuristic: average runtime (H), objective gap, and impact on the solution time if added to the optimal variants of RAS and SCF.

Instance	H	obj[%]	RAS*	+WS	SCF*	+WS
i–11	0.0	89.3%	6.4	5.3	20.8	126.0
i–12	0.1	54.7%	17.4	24.2	105.6	319.7
i–13	0.1	283.6%	9.2	12.2	20.3	111.5
i–14	0.1	88.5%	5.7	12.7	11.7	135.0
i–15	0.1	67.3%	3.8	5.4	5.0	15.3
i–16	0.1	90.5%	7.8	7.4	6.4	41.4
i–17	0.0	92.2%	10.6	18.4	30.2	114.7
i–18	0.1	79.8%	6.7	7.7	10.9	80.5
i–19	0.1	57.7%	3.4	3.6	5.8	81.5
i–21	0.0	72.7%	0.2	0.7	0.3	2.2
i–22	0.1	42.1%	1.0	2.2	2.7	13.0
i–23	0.0	100.0%	1.7	2.9	3.6	14.2
i–24	0.1	69.9%	2.2	2.9	3.4	105.7
i–25	0.0	88.8%	1.4	3.0	1.6	10.1
i–26	0.1	66.7%	2.0	4.1	1.1	7.3
i–27	0.0	76.5%	1.1	1.6	2.0	8.4
i–28	0.1	63.8%	1.5	2.2	3.0	65.7
i–29	0.1	100.0%	1.9	1.8	3.7	47.3
i–31	0.1	67.6%	4.0	4.1	11.4	70.3
i–32	0.1	80.6%	6.6	6.4	56.6	134.4
i–33	0.1	-1.8%	4.8	7.6	4.6	27.8
i–34	0.1	91.7%	7.7	27.4	13.1	144.9
i–35	0.1	82.7%	1.4	4.0	4.3	12.8
i–36	0.1	83.1%	1.5	3.7	2.2	29.6
i–37	0.1	70.4%	1.8	4.7	2.5	50.6
i–38	0.1	97.6%	1.8	2.8	6.5	72.7
i–39	0.1	87.7%	1.9	3.7	3.4	78.1
i–41	0.1	79.9%	2.9	3.3	5.6	56.0
i–42	0.0	80.3%	2.8	5.1	6.0	86.9
i–43	0.0	72.5%	1.8	3.4	3.1	30.4
i–44	0.1	92.5%	3.9	5.1	8.0	48.2
i–45	0.0	100.0%	2.4	3.1	4.0	32.4
i–46	0.1	85.1%	1.4	3.2	2.7	20.1
i–47	0.1	75.8%	1.3	2.5	5.0	57.4
i–48	0.1	92.8%	4.7	6.9	7.5	82.1
i–49	0.1	98.9%	4.6	7.9	17.1	164.8
i–51	0.0	80.2%	4.0	4.8	16.1	251.8
i–52	0.1	78.6%	2.0	4.6	10.6	76.3
i–53	0.0	79.0%	2.2	5.0	4.9	57.8
i–54	0.1	81.2%	1.2	4.2	3.8	21.7
i–55	0.1	85.3%	1.8	2.5	2.5	14.2
i–56	0.0	92.4%	1.2	3.5	2.3	25.0
i–57	0.0	83.4%	1.6	2.4	3.6	69.4
i–58	0.0	93.0%	1.4	2.6	3.5	48.7
i–59	0.0	92.8%	3.6	2.4	3.6	24.4

Instance	H	obj[%]	RAS*	+WS	SCF*	+WS
l–1	0.8	74.1%	90.0	547.4	90.9	2283.0
l–2	1.1	79.5%	96.8	867.5	186.6	3600.0
l–3	0.4	96.9%	56.0	476.1	444.7	1533.9
l–4	0.8	85.9%	90.3	536.7	155.9	2637.9
l–5	0.5	84.3%	26.1	283.5	139.9	1617.8
l–6	0.3	94.4%	93.2	613.7	137.5	1179.8
l–7	0.6	72.7%	8.7	234.1	25.6	1174.0
l–8	0.6	84.0%	113.4	672.1	514.2	3600.0
l–9	0.4	84.8%	30.9	376.3	117.6	1706.9
l–10	0.8	83.8%	84.9	994.2	198.7	3600.0
s–1	0.9	100.0%	2.9	3.1	8.5	63.3
s–2	0.6	61.2%	2.0	8.0	3.4	117.2
s–3	0.9	92.5%	1.8	6.2	2.7	90.7
s–4	1.0	100.0%	2.2	1.7	3.1	10.1
s–5	0.7	100.0%	1.8	4.2	3.7	63.5
s–6	0.6	100.0%	1.4	5.0	3.1	169.8
s–7	0.7	54.3%	3.2	7.8	5.0	89.0
s–8	0.7	90.0%	1.3	5.5	1.6	68.5
s–9	0.8	100.0%	2.3	6.0	2.8	119.6
s–10	0.3	100.0%	1.5	2.1	4.5	83.3
g–1	0.0	57.3%	46.6	20.7	438.8	546.2
g–2	0.1	63.5%	30.0	28.9	347.8	2385.2
g–3	0.1	67.4%	11.2	16.8	461.7	761.2
g–4	0.1	61.6%	25.8	17.4	226.5	552.1
g–5	0.1	62.5%	9.1	5.0	60.7	68.9
g–6	0.1	56.4%	25.8	18.9	126.8	165.9
g–7	0.0	74.8%	6.8	13.8	353.5	588.6
g–8	0.1	59.7%	17.7	12.1	126.4	342.0
g–9	0.1	54.4%	46.8	38.0	1360.2	3600.0
g–10	0.0	52.2%	21.2	19.2	392.9	632.6

B

Detailed Results for TSDP

The second part of the appendix contains detailed results for the experiments carried out in Chapter 6, i.e., regarding the Toll Section Design Problem. In particular, we report instance-wise results for

- different connectivity formulations for the TSDP:
  - without the pricing heuristic in Table B.1,
  - with usage of the pricing heuristic in Table B.2,
- the impact of different enhancements (pricing heuristic, preprocessing, local search):
  - for RAS in Table B.3,
  - for SCF in Table B.4,
- the effect of the preprocessing algorithm,
- the branch-and-price approach.

Table B.1.: Computation times (in seconds) for different connectivity formulations when the pricing heuristic is disabled (but with preprocessing and local search).

Instance	RAS	+C2F	SCF	+C2F	RNS	MCF
ger_lg_tiny_1	0.5	0.9	1.0	0.9	0.5	1.9
ger_lg_tiny_2	0.6	0.6	0.7	0.5	0.6	1.5
ger_lg_tiny_3	0.7	1.0	0.8	0.6	0.7	1.6
ger_lg_tiny_4	1.0	1.3	1.2	1.5	1.0	6.6
ger_lg_small_1	2.0	3.3	2.9	2.4	1.8	16.9
ger_lg_small_2	2.8	3.8	4.3	3.6	2.1	22.8
ger_lg_small_3	4.4	5.6	5.5	5.4	3.0	27.2
ger_lg_small_4	1.5	2.4	2.0	1.4	1.4	6.3
ger_lg_small_5	10.8	13.9	10.1	12.3	8.6	89.1
ger_lg_small_6	3.9	5.4	7.6	6.4	4.7	31.1
ger_lg_small_7	1.9	3.2	3.4	3.1	1.7	13.8
ger_lg_small_8	4.1	6.1	7.5	7.6	4.5	45.4
ger_lg_small_9	11.5	13.9	10.8	12.7	8.8	77.5
ger_lg_medium_1	12.3	13.4	20.2	16.5	13.0	202.1
ger_lg_medium_2	10.9	14.8	17.4	19.5	10.1	194.3
ger_lg_medium_3	19.4	21.5	23.3	23.6	14.0	227.3
ger_lg_medium_4	48.5	55.5	36.9	36.9	61.3	767.0
ger_lg_medium_5	46.3	61.3	77.7	62.3	63.2	1434.0
ger_lg_medium_6	21.5	28.6	27.3	26.1	25.7	318.6
ger_lg_medium_7	20.1	23.4	24.1	22.5	17.0	273.2
ger_lg_medium_8	175.2	116.4	102.0	150.4	141.9	2108.4
ger_lg_medium_9	24.9	23.9	31.3	34.2	22.1	267.5
ger_lg_large_1	138.3	253.9	216.3	218.5	183.8	7233.1
ger_lg_large_2	485.2	538.3	689.2	690.9	2633.4	86400.0
tree_lg_1	19.2	32.2	34.4	33.6	17.9	402.4
tree_lg_2	17.7	25.8	26.8	23.3	15.8	323.3
tree_lg_3	16.7	30.2	34.2	37.3	15.3	262.3
tree_lg_4	36.5	43.3	56.6	44.9	33.8	587.4
tree_lg_5	17.1	25.0	24.6	30.3	15.4	330.8
tree_lg_6	12.1	21.1	20.0	19.8	11.1	224.9
tree_lg_7	13.3	18.7	19.5	17.5	11.9	143.8
tree_lg_8	27.1	25.9	34.8	32.4	24.9	290.2
tree_lg_9	20.7	28.9	35.1	30.6	19.0	324.3
tree_lg_10	23.0	30.7	32.9	28.2	21.4	317.5
tree_lg_11	21.1	31.0	35.0	40.8	19.4	406.9
tree_lg_12	25.8	32.9	38.4	36.9	23.6	568.9
tree_lg_13	25.6	41.1	46.5	38.9	23.4	474.4
tree_lg_14	33.4	28.7	54.4	50.1	30.4	389.2
tree_lg_15	19.3	27.5	29.8	32.6	17.9	359.4
tree_lg_16	31.4	43.0	55.2	43.9	29.2	382.3
tree_lg_17	15.8	21.8	27.0	24.4	14.2	300.3
tree_lg_18	19.0	24.1	31.5	28.4	17.5	309.2
tree_lg_19	21.7	31.1	26.8	21.0	19.2	199.8
tree_lg_20	11.8	16.8	22.6	18.5	10.8	146.8
tree_lg_21	21.1	38.1	42.6	47.4	19.4	383.2
tree_lg_22	12.7	20.1	27.0	25.8	11.7	240.9
tree_lg_23	19.2	32.7	31.5	29.7	18.0	262.6
tree_lg_24	17.7	24.7	34.5	32.7	16.2	335.9
tree_lg_25	23.1	40.9	47.0	43.8	21.0	689.9

Instance	RAS	+C2F	SCF	+C2F	RNS	MCF
voronoi_lg_medium_1	148.1	217.4	156.3	156.7	159.2	3200.8
voronoi_lg_medium_2	129.5	140.6	152.0	169.2	110.3	3895.6
voronoi_lg_medium_3	174.4	256.2	162.5	173.3	135.2	4417.3
voronoi_lg_medium_4	129.5	259.1	192.6	226.1	108.5	5266.4
voronoi_lg_medium_5	184.7	280.2	156.1	201.4	179.0	4720.5
voronoi_lg_medium_6	279.1	227.0	145.7	131.6	168.0	3795.6
voronoi_lg_medium_7	127.5	145.2	117.0	132.4	135.6	2086.2
voronoi_lg_medium_8	208.9	437.8	227.5	231.6	221.5	5858.7
voronoi_lg_medium_9	159.9	268.2	141.4	182.1	179.4	5413.3
voronoi_lg_large_1	414.4	653.7	553.1	665.7	481.7	86400.0
voronoi_lg_large_2	130.3	169.1	144.1	154.7	144.3	4432.8
voronoi_lg_large_3	111.4	185.9	132.9	117.1	133.4	3560.9
voronoi_lg_large_4	140.8	228.2	170.5	180.8	137.4	4858.9
voronoi_lg_large_5	96.7	174.8	177.8	165.2	90.9	5816.9
voronoi_lg_large_6	275.2	424.7	298.7	351.9	310.1	86400.0
voronoi_lg_large_7	153.6	289.7	194.2	231.7	142.5	86400.0
voronoi_lg_large_8	241.4	397.3	287.8	250.4	258.5	86400.0
voronoi_lg_large_9	162.2	204.1	166.8	174.1	148.7	5848.0
voronoi_lg_large_10	138.4	182.2	149.0	173.0	139.4	5620.3
voronoi_lg_large_11	193.4	314.2	269.0	284.1	234.3	86400.0
voronoi_lg_large_12	315.2	549.2	373.8	464.2	338.7	86400.0
voronoi_lg_large_13	334.3	570.9	472.8	462.0	964.7	86400.0
voronoi_lg_large_14	187.7	298.9	215.7	336.0	198.7	86400.0
voronoi_lg_large_15	168.5	290.7	179.0	202.7	156.5	86400.0
voronoi_lg_large_16	222.7	361.5	205.9	215.7	205.9	86400.0
voronoi_lg_large_17	380.6	711.2	379.6	357.5	346.5	86400.0
voronoi_lg_large_18	391.7	547.5	382.8	460.2	403.7	86400.0
voronoi_lg_large_19	434.3	746.3	586.2	599.9	621.1	86400.0
voronoi_lg_large_20	113.2	180.5	120.5	125.4	115.2	4296.2
voronoi_lg_large_21	388.9	498.0	349.7	470.0	285.0	86400.0
voronoi_lg_large_22	195.8	257.4	195.3	216.0	202.8	86400.0
voronoi_lg_large_23	219.4	325.0	259.4	372.3	213.3	86400.0
voronoi_lg_large_24	284.0	455.1	254.9	422.7	210.5	86400.0
voronoi_lg_large_25	473.3	676.3	522.8	605.2	773.4	86400.0
voronoi_lg_large_26	257.6	497.8	323.8	390.5	300.6	86400.0
voronoi_lg_large_27	129.7	214.2	123.6	214.1	102.7	5994.5
voronoi_lg_large_28	188.5	412.8	204.9	284.8	198.6	86400.0
voronoi_lg_large_29	115.1	259.5	193.2	321.6	103.7	86400.0
voronoi_lg_large_30	248.8	479.5	232.5	361.7	225.3	86400.0
voronoi_lg_large_31	306.9	613.3	471.7	506.2	358.9	15186.1
voronoi_lg_large_32	257.3	275.3	296.0	289.7	222.0	86400.0
voronoi_lg_large_33	293.4	425.5	330.9	369.1	263.0	86400.0
voronoi_lg_large_34	277.3	463.2	275.3	370.0	267.1	86400.0
voronoi_lg_large_35	263.9	350.5	229.7	261.4	234.1	5405.3
voronoi_lg_large_36	209.1	307.3	234.5	276.9	180.4	86400.0
voronoi_lg_large_37	262.8	373.1	289.8	263.0	253.3	86400.0
voronoi_lg_large_38	231.4	353.0	288.7	446.4	217.3	86400.0
voronoi_lg_large_39	239.0	458.9	241.5	329.8	186.3	86400.0
voronoi_lg_large_40	199.2	277.1	264.5	343.1	208.4	86400.0
voronoi_lg_large_41	164.0	299.2	221.6	241.3	211.3	86400.0

Table B.2.: Computation times (in seconds) for different connectivity formulations with pricing heuristic, preprocessing and local search.

Instance	RAS	+C2F	SCF	+C2F	RNS	MCF
ger_lg_tiny_1	1.0	1.3	1.1	1.1	1.0	1.6
ger_lg_tiny_2	1.0	1.2	1.0	1.0	1.0	1.5
ger_lg_tiny_3	1.2	1.2	1.2	1.2	1.2	2.0
ger_lg_tiny_4	1.8	1.9	1.9	1.9	1.8	4.1
ger_lg_small_1	3.7	4.5	4.0	4.0	3.7	15.7
ger_lg_small_2	3.7	5.0	4.0	4.0	3.6	9.7
ger_lg_small_3	4.4	5.0	4.7	4.7	4.4	13.2
ger_lg_small_4	1.9	2.0	1.9	1.9	1.9	3.5
ger_lg_small_5	12.9	12.0	10.6	9.4	10.8	41.6
ger_lg_small_6	4.5	5.1	4.9	5.5	4.3	12.5
ger_lg_small_7	2.9	3.2	3.2	3.2	3.0	8.0
ger_lg_small_8	5.4	5.7	5.8	5.9	5.4	16.9
ger_lg_small_9	7.0	7.7	7.4	7.4	6.8	20.6
ger_lg_medium_1	13.0	16.1	13.8	13.1	12.7	79.5
ger_lg_medium_2	14.8	20.3	17.6	13.3	15.2	104.1
ger_lg_medium_3	12.1	15.5	13.3	13.4	12.1	62.3
ger_lg_medium_4	27.4	31.5	27.1	29.7	31.5	219.3
ger_lg_medium_5	25.6	28.5	28.0	27.8	26.1	317.3
ger_lg_medium_6	18.0	21.3	21.1	19.6	18.1	144.4
ger_lg_medium_7	15.8	20.3	16.8	17.2	14.9	108.5
ger_lg_medium_8	70.9	96.1	63.1	62.7	104.5	964.6
ger_lg_medium_9	26.0	24.6	28.6	25.0	22.7	116.3
ger_lg_large_1	60.1	81.4	71.5	72.8	57.4	1113.4
ger_lg_large_2	149.8	164.4	150.9	148.4	159.2	86400.0
tree_lg_1	21.0	24.9	22.9	22.3	20.8	84.6
tree_lg_2	14.1	15.3	15.3	15.3	14.2	59.4
tree_lg_3	18.2	19.7	19.6	19.5	18.2	61.0
tree_lg_4	24.1	28.0	25.8	25.6	24.1	96.6
tree_lg_5	15.2	16.0	16.2	16.1	15.0	60.9
tree_lg_6	12.1	13.3	13.2	13.1	12.0	46.4
tree_lg_7	11.2	12.6	11.9	11.8	11.0	45.0
tree_lg_8	17.4	20.9	18.6	18.4	17.3	62.0
tree_lg_9	21.0	25.5	22.5	22.5	20.8	88.6
tree_lg_10	18.7	20.4	20.2	20.1	18.7	74.8
tree_lg_11	17.1	19.7	18.6	18.4	16.9	69.8
tree_lg_12	20.3	22.4	21.7	21.6	20.1	95.1
tree_lg_13	21.8	23.2	23.4	23.2	21.9	81.0
tree_lg_14	20.1	21.4	21.7	21.5	19.9	101.6
tree_lg_15	17.9	19.3	19.2	19.3	18.0	67.4
tree_lg_16	21.6	23.7	22.7	22.5	21.2	84.0
tree_lg_17	14.8	16.1	16.2	16.0	14.8	66.4
tree_lg_18	17.8	19.2	19.1	19.2	17.6	71.7
tree_lg_19	17.1	18.7	17.6	17.6	16.8	58.2
tree_lg_20	12.2	13.5	13.4	13.3	12.3	50.2
tree_lg_21	20.8	22.0	22.5	22.4	20.8	91.1
tree_lg_22	14.9	16.8	16.0	15.8	14.6	61.3
tree_lg_23	19.3	20.3	20.1	20.3	19.2	68.7
tree_lg_24	16.3	18.3	17.5	17.4	16.1	80.8
tree_lg_25	22.7	27.4	24.2	24.6	22.6	110.0

Instance	RAS	+C2F	SCF	+C2F	RNS	MCF
voronoi_lg_medium_1	72.9	144.6	74.2	86.3	70.6	933.9
voronoi_lg_medium_2	35.7	49.8	37.4	37.9	34.8	246.9
voronoi_lg_medium_3	52.6	74.4	48.4	52.5	50.3	755.1
voronoi_lg_medium_4	80.0	99.8	88.5	75.7	77.7	1329.6
voronoi_lg_medium_5	61.2	80.2	60.3	59.6	56.6	747.3
voronoi_lg_medium_6	51.8	55.6	50.1	48.0	50.3	329.4
voronoi_lg_medium_7	72.3	84.1	51.9	71.3	62.6	320.0
voronoi_lg_medium_8	81.4	107.0	102.0	82.4	78.9	1095.2
voronoi_lg_medium_9	88.8	103.4	85.4	73.6	85.9	1068.6
voronoi_lg_large_1	105.6	251.4	149.8	158.5	142.0	4913.1
voronoi_lg_large_2	74.8	113.0	79.3	64.4	70.5	1305.9
voronoi_lg_large_3	53.3	69.9	66.1	65.1	51.7	760.5
voronoi_lg_large_4	69.1	114.1	66.9	68.8	65.6	1045.5
voronoi_lg_large_5	39.6	48.6	44.3	45.7	39.6	353.0
voronoi_lg_large_6	125.6	274.9	116.0	126.0	121.1	3010.2
voronoi_lg_large_7	46.4	58.8	51.3	50.9	45.9	750.7
voronoi_lg_large_8	104.7	178.5	106.9	105.3	102.3	1824.7
voronoi_lg_large_9	84.0	211.0	115.8	119.9	91.9	2599.0
voronoi_lg_large_10	50.3	75.5	53.5	55.4	49.5	702.3
voronoi_lg_large_11	67.0	131.7	62.5	76.4	64.8	1640.5
voronoi_lg_large_12	196.7	264.3	126.3	165.2	130.3	5117.5
voronoi_lg_large_13	130.4	167.4	145.9	118.9	135.2	3962.6
voronoi_lg_large_14	60.9	124.1	63.1	65.9	58.3	941.9
voronoi_lg_large_15	80.1	118.0	65.4	72.2	62.6	2137.3
voronoi_lg_large_16	86.1	176.3	93.4	94.6	98.7	2095.8
voronoi_lg_large_17	127.6	354.2	156.4	162.5	121.3	3339.7
voronoi_lg_large_18	157.2	270.6	189.7	152.4	192.8	3144.1
voronoi_lg_large_19	214.5	465.3	176.0	215.8	265.6	6824.9
voronoi_lg_large_20	45.6	55.1	44.5	43.2	44.6	321.5
voronoi_lg_large_21	142.6	227.5	124.6	147.6	128.8	3933.1
voronoi_lg_large_22	56.7	67.7	56.6	56.6	57.4	944.5
voronoi_lg_large_23	87.8	84.1	86.4	76.4	83.8	1293.3
voronoi_lg_large_24	89.6	91.2	89.2	78.0	86.8	1982.4
voronoi_lg_large_25	222.1	280.8	224.7	188.1	204.6	7539.8
voronoi_lg_large_26	81.8	111.6	77.5	86.6	78.1	1482.0
voronoi_lg_large_27	46.5	55.8	50.9	53.5	44.7	773.7
voronoi_lg_large_28	47.2	67.7	49.3	49.2	46.6	570.0
voronoi_lg_large_29	67.8	96.2	76.6	88.4	64.3	1822.7
voronoi_lg_large_30	70.9	99.8	93.9	80.5	69.5	1673.5
voronoi_lg_large_31	114.1	217.2	103.9	153.1	117.8	3581.1
voronoi_lg_large_32	106.4	131.0	101.8	112.0	104.2	2259.4
voronoi_lg_large_33	86.3	111.6	91.7	112.5	84.9	2251.6
voronoi_lg_large_34	104.8	101.6	81.5	88.9	73.9	1988.9
voronoi_lg_large_35	105.3	133.2	89.8	68.9	102.9	1597.8
voronoi_lg_large_36	73.3	94.0	81.0	82.3	71.0	1547.8
voronoi_lg_large_37	71.7	89.6	77.5	75.1	70.9	1224.6
voronoi_lg_large_38	74.7	129.3	102.3	114.1	84.0	2235.9
voronoi_lg_large_39	90.7	145.4	90.7	91.2	78.0	1881.2
voronoi_lg_large_40	56.3	64.6	66.0	66.2	55.5	1579.9
voronoi_lg_large_41	56.1	73.1	59.2	59.1	53.3	1237.8

Table B.3.: Computation times (in seconds) for RAS when including combinations of the pricing heuristic (`H`), the preprocessing (`P`), or the local search (`L`).

Instance	`---`	`--L`	`-P-`	`H--`	`-PL`	`H-L`	`HP-`	`HPL`
ger_lg_tiny_1	3.5	2.1	2.4	2.0	1.1	1.6	1.2	1.0
ger_lg_tiny_2	3.5	2.1	2.0	1.9	1.2	1.7	1.2	1.0
ger_lg_tiny_3	4.0	2.9	2.0	2.2	1.4	1.9	1.3	1.2
ger_lg_tiny_4	9.4	4.3	3.2	4.0	2.0	3.3	2.0	1.8
ger_lg_small_1	17.7	7.6	7.0	7.3	3.7	6.0	4.3	3.7
ger_lg_small_2	19.5	12.4	10.3	9.3	4.4	5.3	3.9	3.7
ger_lg_small_3	40.5	24.9	10.8	18.7	6.7	9.8	7.5	4.4
ger_lg_small_4	9.1	4.2	6.2	4.4	2.5	2.9	3.0	1.9
ger_lg_small_5	104.6	46.1	32.4	33.5	15.4	22.8	20.6	12.9
ger_lg_small_6	47.8	20.6	11.6	15.1	6.0	10.2	5.9	4.5
ger_lg_small_7	13.5	9.4	6.4	9.1	3.4	4.8	3.8	2.9
ger_lg_small_8	109.7	45.1	18.6	104.3	6.9	11.5	17.9	5.4
ger_lg_small_9	94.6	50.6	22.4	14.9	16.3	19.3	8.4	7.0
ger_lg_medium_1	217.4	107.7	46.6	94.2	18.1	37.7	19.1	13.0
ger_lg_medium_2	106.6	61.3	29.0	70.5	15.8	21.7	14.2	14.8
ger_lg_medium_3	171.7	98.2	46.4	32.0	26.7	18.5	18.6	12.1
ger_lg_medium_4	763.8	298.0	148.2	238.1	64.5	93.8	63.6	27.4
ger_lg_medium_5	684.1	336.4	151.5	128.6	59.6	82.6	43.4	25.6
ger_lg_medium_6	248.0	114.8	64.1	115.8	29.4	28.3	19.9	18.0
ger_lg_medium_7	216.6	139.0	41.0	65.9	26.5	33.8	15.5	15.8
ger_lg_medium_8	786.2	451.8	326.9	328.7	199.4	105.8	125.4	70.9
ger_lg_medium_9	305.0	125.0	78.7	88.6	35.0	48.2	35.4	26.0
ger_lg_large_1	1178.9	493.4	414.1	368.5	177.4	73.6	201.8	60.1
ger_lg_large_2	3920.3	2054.8	1359.5	1310.1	549.1	399.5	384.1	149.8
tree_lg_1	150.7	77.5	57.6	61.6	29.8	38.3	33.0	21.0
tree_lg_2	123.8	69.3	59.1	52.6	26.0	28.3	23.2	14.1
tree_lg_3	165.5	74.8	58.3	60.8	24.7	40.2	28.9	18.2
tree_lg_4	222.9	81.4	105.3	71.9	51.7	39.1	43.2	24.1
tree_lg_5	160.6	65.4	52.7	47.5	25.3	37.0	24.6	15.2
tree_lg_6	126.0	56.0	34.3	43.1	18.4	29.3	19.0	12.1
tree_lg_7	120.2	50.3	38.1	50.6	19.3	26.2	17.5	11.2
tree_lg_8	141.7	88.6	77.9	54.8	37.5	36.7	28.1	17.4
tree_lg_9	143.0	79.9	69.2	55.7	32.0	36.1	42.0	21.0
tree_lg_10	167.6	67.6	73.7	54.4	34.3	33.6	34.0	18.7
tree_lg_11	120.2	51.9	52.5	50.4	31.4	30.7	28.7	17.1
tree_lg_12	119.0	74.0	69.6	42.8	36.0	32.7	25.9	20.3
tree_lg_13	159.0	86.0	86.8	58.8	39.2	48.4	32.9	21.8
tree_lg_14	183.3	78.0	124.5	71.3	45.3	40.3	33.5	20.1
tree_lg_15	178.9	94.0	76.0	54.6	28.7	37.2	28.7	17.9
tree_lg_16	164.3	109.8	78.5	54.0	42.5	37.4	42.5	21.6
tree_lg_17	113.1	78.1	56.2	54.3	23.7	31.0	23.7	14.8
tree_lg_18	119.7	81.9	54.1	51.4	27.8	34.8	27.9	17.8
tree_lg_19	130.4	69.9	46.6	50.6	29.5	35.6	29.6	17.1
tree_lg_20	92.1	45.8	41.9	41.4	18.6	24.2	18.6	12.2
tree_lg_21	136.6	67.9	72.0	59.5	32.4	38.7	32.5	20.8
tree_lg_22	119.7	59.2	45.7	53.4	19.9	29.6	20.0	14.9
tree_lg_23	128.6	68.1	65.0	40.7	28.3	33.9	28.4	19.3
tree_lg_24	148.9	72.5	92.9	59.3	27.2	35.5	27.3	16.3
tree_lg_25	154.8	83.0	85.6	73.7	33.9	42.0	33.9	22.7

Instance	---	--L	-P-	H--	-PL	H-L	HP-	HPL
voronoi_lg_medium_1	848.7	454.5	269.6	210.3	173.0	133.6	121.7	72.9
voronoi_lg_medium_2	787.7	375.5	302.8	68.2	156.5	45.4	49.2	35.7
voronoi_lg_medium_3	701.7	511.5	226.6	116.4	200.4	70.2	53.4	52.6
voronoi_lg_medium_4	989.1	519.3	303.1	149.8	158.2	70.1	71.0	80.0
voronoi_lg_medium_5	765.1	492.6	288.8	196.7	209.8	72.0	71.5	61.2
voronoi_lg_medium_6	812.4	490.1	286.5	87.2	313.2	63.2	57.1	51.8
voronoi_lg_medium_7	1172.7	456.2	270.7	151.7	149.0	70.5	97.2	72.3
voronoi_lg_medium_8	830.4	510.7	352.1	198.9	233.7	127.6	95.6	81.4
voronoi_lg_medium_9	708.0	387.3	290.9	149.9	190.9	54.0	98.8	88.8
voronoi_lg_large_1	2398.8	1042.9	824.0	322.5	471.0	208.4	221.5	105.6
voronoi_lg_large_2	848.6	447.6	278.8	230.0	154.4	92.2	92.3	74.8
voronoi_lg_large_3	795.2	385.3	299.4	154.8	140.4	66.4	84.1	53.3
voronoi_lg_large_4	901.4	493.2	327.1	213.9	164.5	89.2	83.2	69.1
voronoi_lg_large_5	936.5	480.9	247.1	88.9	122.7	49.1	42.9	39.6
voronoi_lg_large_6	1209.7	618.4	546.6	419.6	305.9	160.5	181.1	125.6
voronoi_lg_large_7	988.9	524.7	377.2	86.5	179.4	47.3	54.8	46.4
voronoi_lg_large_8	1058.5	642.2	484.8	273.1	273.9	118.7	126.7	104.7
voronoi_lg_large_9	1162.2	559.8	384.1	327.5	191.6	87.8	166.8	84.0
voronoi_lg_large_10	765.7	435.2	276.3	246.9	165.3	40.8	67.5	50.3
voronoi_lg_large_11	1003.6	757.7	463.7	214.2	222.5	88.5	86.5	67.0
voronoi_lg_large_12	1965.0	886.1	948.2	414.2	345.9	145.3	123.5	196.7
voronoi_lg_large_13	1873.1	889.7	740.4	336.8	372.1	162.9	199.9	130.4
voronoi_lg_large_14	968.9	528.1	387.7	196.8	216.6	136.1	160.1	60.9
voronoi_lg_large_15	1170.6	591.6	422.5	481.5	199.6	111.9	86.8	80.1
voronoi_lg_large_16	1208.4	747.8	431.3	555.4	250.8	112.7	181.9	86.1
voronoi_lg_large_17	1684.6	780.0	667.0	459.1	418.1	161.7	219.2	127.6
voronoi_lg_large_18	1712.5	877.9	715.7	452.5	436.9	186.1	246.3	157.2
voronoi_lg_large_19	2592.9	908.2	1537.7	730.3	480.2	244.8	272.1	214.5
voronoi_lg_large_20	841.1	437.5	231.5	88.1	132.2	48.3	46.3	45.6
voronoi_lg_large_21	1982.1	826.5	720.9	471.3	425.1	161.7	208.7	142.6
voronoi_lg_large_22	1472.0	603.9	385.4	193.2	224.0	93.4	67.9	56.7
voronoi_lg_large_23	1115.5	605.1	367.2	194.5	251.5	72.3	118.4	87.8
voronoi_lg_large_24	1633.1	712.6	426.3	206.3	315.6	120.8	118.4	89.6
voronoi_lg_large_25	2077.3	1158.4	998.5	486.8	525.8	328.1	266.9	222.1
voronoi_lg_large_26	1091.7	647.5	594.7	335.4	288.9	127.4	144.3	81.8
voronoi_lg_large_27	945.3	551.8	263.5	137.9	155.3	61.7	59.0	46.5
voronoi_lg_large_28	976.6	608.1	329.2	127.1	212.6	52.4	62.1	47.2
voronoi_lg_large_29	977.1	577.4	268.4	102.2	143.7	60.7	62.7	67.8
voronoi_lg_large_30	1139.4	880.5	381.8	174.5	281.3	98.1	92.7	70.9
voronoi_lg_large_31	1540.2	857.7	555.5	467.1	338.6	157.3	183.3	114.1
voronoi_lg_large_32	1051.8	667.6	424.0	241.7	289.3	90.2	120.8	106.4
voronoi_lg_large_33	1625.1	999.5	592.5	274.5	324.3	134.0	139.2	86.3
voronoi_lg_large_34	1289.2	714.5	513.8	195.7	309.7	84.4	132.8	104.8
voronoi_lg_large_35	1112.1	634.9	371.0	201.4	289.0	139.9	70.7	105.3
voronoi_lg_large_36	1041.6	608.4	340.6	122.9	242.4	66.3	82.8	73.3
voronoi_lg_large_37	887.6	651.6	352.0	86.3	296.5	68.4	74.4	71.7
voronoi_lg_large_38	1149.1	597.5	400.8	281.4	257.1	130.9	109.4	74.7
voronoi_lg_large_39	1159.5	839.3	327.2	214.3	269.2	96.2	97.2	90.7
voronoi_lg_large_40	1578.5	759.2	368.7	101.0	238.2	58.4	59.6	56.3
voronoi_lg_large_41	1052.0	615.0	319.9	321.8	188.4	173.0	113.9	56.1

Table B.4.: Computation times (in seconds) for SCF when including combinations of the pricing heuristic (`H`), the preprocessing (`P`), or the local search (`L`).

Instance	`---`	`--L`	`-P-`	`H--`	`-PL`	`H-L`	`HP-`	`HPL`
ger_lg_tiny_1	5.0	3.1	2.6	2.1	1.6	1.9	1.3	1.1
ger_lg_tiny_2	4.0	2.4	1.6	2.0	1.2	1.9	1.2	1.0
ger_lg_tiny_3	4.1	2.9	2.0	2.3	1.5	2.2	1.4	1.2
ger_lg_tiny_4	9.1	5.4	4.1	4.1	2.3	3.7	2.1	1.9
ger_lg_small_1	14.6	9.8	9.3	7.2	4.8	6.2	4.6	4.0
ger_lg_small_2	24.8	12.1	12.9	9.2	6.2	6.3	4.2	4.0
ger_lg_small_3	34.9	19.4	14.1	15.7	7.9	10.9	8.3	4.7
ger_lg_small_4	12.3	5.9	7.7	4.5	3.1	3.4	3.2	1.9
ger_lg_small_5	87.2	42.0	26.5	39.9	14.6	19.1	16.4	10.6
ger_lg_small_6	42.4	27.1	15.0	19.2	10.0	11.8	8.0	4.9
ger_lg_small_7	17.9	10.0	8.5	9.1	5.2	5.9	4.2	3.2
ger_lg_small_8	88.6	30.2	36.6	58.2	10.5	12.6	27.2	5.8
ger_lg_small_9	82.2	42.9	22.0	14.0	15.0	16.8	9.3	7.4
ger_lg_medium_1	175.7	107.4	46.8	47.4	26.2	41.8	23.4	13.8
ger_lg_medium_2	123.1	42.7	44.4	35.6	22.8	35.4	18.8	17.6
ger_lg_medium_3	192.5	102.9	53.9	28.0	30.2	19.6	21.3	13.3
ger_lg_medium_4	579.4	241.6	120.5	221.0	51.3	51.2	90.6	27.1
ger_lg_medium_5	826.8	452.4	219.2	103.8	92.4	71.0	54.6	28.0
ger_lg_medium_6	224.2	111.0	64.7	55.9	34.4	35.5	22.6	21.1
ger_lg_medium_7	163.8	102.5	52.5	50.6	30.8	32.2	17.7	16.8
ger_lg_medium_8	831.9	320.2	308.1	246.4	122.2	122.5	103.4	63.1
ger_lg_medium_9	202.4	115.9	96.2	125.4	43.4	87.6	39.7	28.6
ger_lg_large_1	1384.7	775.8	499.9	287.7	253.5	163.3	211.7	71.5
ger_lg_large_2	6285.8	3311.2	1674.2	890.3	748.2	273.8	344.7	150.9
tree_lg_1	281.0	161.2	91.7	65.8	44.6	46.4	31.0	22.9
tree_lg_2	235.0	139.3	68.9	57.4	34.8	35.6	19.6	15.3
tree_lg_3	301.7	155.1	77.3	66.1	42.3	48.9	24.0	19.6
tree_lg_4	407.5	191.6	158.0	77.5	70.4	48.4	37.0	25.8
tree_lg_5	241.0	119.9	84.3	51.8	32.7	45.3	22.3	16.2
tree_lg_6	219.9	116.1	62.4	46.8	26.7	35.7	18.1	13.2
tree_lg_7	234.2	134.4	61.5	54.2	25.4	32.1	15.2	11.9
tree_lg_8	368.3	189.2	84.8	60.5	44.6	45.1	22.9	18.6
tree_lg_9	303.2	173.6	120.6	60.8	46.6	44.6	32.7	22.5
tree_lg_10	272.7	140.6	104.1	58.0	44.5	40.1	28.7	20.2
tree_lg_11	218.4	119.1	86.3	53.7	45.1	37.7	25.4	18.6
tree_lg_12	249.5	124.4	105.7	46.3	49.0	39.6	23.5	21.7
tree_lg_13	351.8	217.5	145.8	64.4	59.4	59.0	29.0	23.4
tree_lg_14	330.1	190.8	115.3	78.5	67.3	50.5	31.7	21.7
tree_lg_15	274.5	156.2	91.4	59.2	39.7	44.9	27.5	19.2
tree_lg_16	276.3	199.7	113.0	59.7	66.3	45.8	30.8	22.7
tree_lg_17	234.2	98.8	107.5	58.3	35.3	38.4	19.7	16.2
tree_lg_18	226.1	149.0	69.5	56.0	40.5	43.0	25.3	19.1
tree_lg_19	273.8	149.1	72.2	53.9	34.8	43.1	25.6	17.6
tree_lg_20	236.6	116.9	56.8	46.4	29.8	30.3	16.1	13.4
tree_lg_21	349.5	201.9	105.9	66.0	54.5	47.2	27.8	22.5
tree_lg_22	227.5	132.1	75.3	56.9	35.1	35.7	23.1	16.0
tree_lg_23	198.7	134.7	72.8	45.0	41.0	40.9	24.1	20.1
tree_lg_24	440.4	165.0	101.0	65.5	44.6	43.7	24.1	17.5
tree_lg_25	345.6	187.0	120.4	80.6	59.3	52.4	39.5	24.2

Instance	---	--L	-P-	H--	-PL	H-L	HP-	HPL
voronoi_lg_medium_1	715.6	550.5	285.6	203.0	179.7	133.9	122.1	74.2
voronoi_lg_medium_2	921.5	684.9	251.4	71.4	176.9	56.0	51.6	37.4
voronoi_lg_medium_3	851.5	607.4	233.4	116.6	183.0	82.8	55.8	48.4
voronoi_lg_medium_4	947.9	664.7	291.9	183.6	218.8	72.0	70.9	88.5
voronoi_lg_medium_5	979.8	735.2	280.6	196.0	180.1	80.3	65.1	60.3
voronoi_lg_medium_6	840.5	665.6	333.0	89.3	168.2	67.1	52.7	50.1
voronoi_lg_medium_7	778.5	422.4	295.8	155.0	136.2	71.4	115.8	51.9
voronoi_lg_medium_8	948.8	574.1	373.2	175.5	249.4	127.1	121.7	102.0
voronoi_lg_medium_9	684.1	463.0	347.4	129.7	162.4	57.6	73.3	85.4
voronoi_lg_large_1	4057.5	2000.8	924.9	472.0	601.1	212.0	223.5	149.8
voronoi_lg_large_2	1182.9	913.8	325.4	265.3	166.4	135.0	90.6	79.3
voronoi_lg_large_3	1027.1	497.0	280.3	169.0	158.4	77.7	85.8	66.1
voronoi_lg_large_4	1199.5	653.7	302.4	212.4	194.3	94.2	90.8	66.9
voronoi_lg_large_5	1057.8	567.7	343.1	88.1	204.8	57.6	46.4	44.3
voronoi_lg_large_6	1637.9	1117.3	624.6	372.1	328.6	167.6	184.0	116.0
voronoi_lg_large_7	1130.3	757.4	352.0	81.2	219.4	52.4	57.8	51.3
voronoi_lg_large_8	1666.0	1009.4	501.1	218.7	315.1	128.8	151.9	106.9
voronoi_lg_large_9	1747.1	796.0	400.2	274.5	188.4	100.4	208.1	115.8
voronoi_lg_large_10	1071.9	558.7	326.5	237.6	172.1	48.2	83.8	53.5
voronoi_lg_large_11	1373.3	803.0	471.2	270.6	296.3	102.6	91.6	62.5
voronoi_lg_large_12	2430.1	1823.8	828.5	578.4	404.3	181.0	164.8	126.3
voronoi_lg_large_13	1972.4	1341.3	725.2	279.3	508.5	177.6	147.3	145.9
voronoi_lg_large_14	1535.0	696.6	431.1	242.9	239.5	177.0	197.8	63.1
voronoi_lg_large_15	1519.5	1126.8	319.2	439.3	207.0	121.6	102.3	65.4
voronoi_lg_large_16	1908.9	1210.4	469.1	530.8	230.2	112.3	194.2	93.4
voronoi_lg_large_17	2744.9	1370.1	731.3	638.1	419.3	188.0	252.8	156.4
voronoi_lg_large_18	2750.8	1583.8	799.9	618.8	419.6	202.4	220.9	189.7
voronoi_lg_large_19	4039.0	2247.4	1073.4	833.9	631.0	243.1	311.6	176.0
voronoi_lg_large_20	1277.6	871.8	246.3	93.9	140.8	55.0	47.0	44.5
voronoi_lg_large_21	3225.1	1892.4	794.7	809.2	381.4	178.2	199.6	124.6
voronoi_lg_large_22	1414.3	1032.1	436.0	200.7	219.2	84.6	65.0	56.6
voronoi_lg_large_23	1985.7	1386.6	515.8	216.8	288.2	84.7	92.1	86.4
voronoi_lg_large_24	2106.7	1543.0	502.8	272.4	283.4	95.6	82.0	89.2
voronoi_lg_large_25	3373.9	2642.7	1109.3	964.5	571.5	212.0	281.3	224.7
voronoi_lg_large_26	1882.7	1417.5	697.4	428.8	357.2	125.8	178.0	77.5
voronoi_lg_large_27	1229.5	720.5	258.3	125.2	147.6	68.6	62.7	50.9
voronoi_lg_large_28	1477.6	975.5	401.4	145.8	229.5	61.9	63.5	49.3
voronoi_lg_large_29	1784.5	922.0	357.5	107.3	221.8	76.3	71.9	76.6
voronoi_lg_large_30	1599.6	1198.0	459.1	308.6	261.2	100.4	98.4	93.9
voronoi_lg_large_31	2598.4	1627.7	685.6	663.3	503.3	185.8	254.3	103.9
voronoi_lg_large_32	1560.3	813.9	444.9	246.8	327.0	105.7	170.4	101.8
voronoi_lg_large_33	1630.5	1247.6	562.1	323.2	360.6	146.0	126.5	91.7
voronoi_lg_large_34	2389.6	1285.3	528.3	269.8	304.4	147.4	143.6	81.5
voronoi_lg_large_35	1069.0	749.7	400.7	152.5	253.9	130.8	72.5	89.8
voronoi_lg_large_36	1377.3	1036.0	505.7	189.2	260.1	78.8	112.7	81.0
voronoi_lg_large_37	1592.0	974.6	505.6	110.1	320.5	75.8	78.0	77.5
voronoi_lg_large_38	1267.8	1207.2	421.5	255.5	312.9	126.4	116.1	102.3
voronoi_lg_large_39	1582.7	1564.5	410.8	291.0	267.2	112.6	105.7	90.7
voronoi_lg_large_40	2226.0	1228.0	425.4	105.7	297.8	70.6	69.1	66.0
voronoi_lg_large_41	2082.3	1251.6	338.1	265.0	243.8	164.9	122.7	59.2

Table B.5.: Effects of the preprocessing algorithm.

Instance	roots eliminated	discarded	fixed
ger_lg_tiny_1	42.5%	37.3 %	16.8 %
ger_lg_tiny_2	44.7%	35.8 %	17.6 %
ger_lg_tiny_3	39.6%	39.6 %	18.6 %
ger_lg_tiny_4	33.1%	47.0 %	18.7 %
ger_lg_small_1	34.4%	37.5 %	13.9 %
ger_lg_small_2	36.8%	42.5 %	17.0 %
ger_lg_small_3	47.9%	38.4 %	11.8 %
ger_lg_small_4	46.9%	29.0 %	17.6 %
ger_lg_small_5	40.0%	36.8 %	8.5 %
ger_lg_small_6	37.8%	38.6 %	14.5 %
ger_lg_small_7	34.4%	41.2 %	18.4 %
ger_lg_small_8	49.7%	39.6 %	13.0 %
ger_lg_small_9	38.9%	34.6 %	7.1 %
ger_lg_medium_1	44.1%	39.2 %	13.7 %
ger_lg_medium_2	41.4%	26.7 %	9.1 %
ger_lg_medium_3	45.4%	22.3 %	3.5 %
ger_lg_medium_4	33.6%	38.0 %	5.9 %
ger_lg_medium_5	25.3%	24.9 %	2.0 %
ger_lg_medium_6	28.1%	23.9 %	3.2 %
ger_lg_medium_7	43.3%	33.4 %	9.0 %
ger_lg_medium_8	35.1%	24.7 %	4.8 %
ger_lg_medium_9	43.3%	35.2 %	11.4 %
ger_lg_large_1	23.1%	8.9 %	1.4 %
ger_lg_large_2	32.4%	15.5 %	0.9 %
tree_lg_1	29.6%	38.6 %	8.6 %
tree_lg_2	37.2%	40.3 %	12.0 %
tree_lg_3	37.2%	40.5 %	10.7 %
tree_lg_4	33.2%	36.2 %	7.3 %
tree_lg_5	37.7%	41.9 %	8.9 %
tree_lg_6	38.7%	39.4 %	12.6 %
tree_lg_7	46.2%	42.6 %	11.1 %
tree_lg_8	33.7%	37.3 %	9.4 %
tree_lg_9	28.6%	38.2 %	9.5 %
tree_lg_10	31.2%	36.0 %	9.1 %
tree_lg_11	32.7%	37.2 %	8.3 %
tree_lg_12	33.2%	42.3 %	9.6 %
tree_lg_13	38.7%	37.8 %	6.9 %
tree_lg_14	38.7%	32.6 %	7.0 %
tree_lg_15	30.2%	38.4 %	9.2 %
tree_lg_16	32.2%	36.1 %	8.7 %
tree_lg_17	34.7%	45.1 %	11.5 %
tree_lg_18	35.7%	38.6 %	7.4 %
tree_lg_19	38.7%	40.3 %	11.6 %
tree_lg_20	37.7%	38.1 %	13.3 %
tree_lg_21	27.6%	33.0 %	8.7 %
tree_lg_22	38.2%	41.1 %	11.3 %
tree_lg_23	29.6%	36.7 %	8.8 %
tree_lg_24	38.7%	38.7 %	10.6 %
tree_lg_25	33.7%	38.1 %	9.1 %

Instance	roots eliminated	discarded	fixed
voronoi_lg_medium_1	27.9%	10.2 %	2.6 %
voronoi_lg_medium_2	23.2%	2.8 %	3.0 %
voronoi_lg_medium_3	30.5%	5.2 %	1.0 %
voronoi_lg_medium_4	11.3%	1.6 %	1.5 %
voronoi_lg_medium_5	17.7%	6.3 %	1.8 %
voronoi_lg_medium_6	32.0%	1.1 %	0.9 %
voronoi_lg_medium_7	36.1%	7.3 %	1.4 %
voronoi_lg_medium_8	19.1%	5.3 %	1.9 %
voronoi_lg_medium_9	16.7%	1.9 %	1.2 %
voronoi_lg_large_1	13.9%	0.8 %	0.6 %
voronoi_lg_large_2	25.2%	13.6 %	1.6 %
voronoi_lg_large_3	25.5%	4.5 %	2.3 %
voronoi_lg_large_4	19.5%	3.9 %	1.7 %
voronoi_lg_large_5	27.6%	5.0 %	1.7 %
voronoi_lg_large_6	17.0%	3.4 %	1.0 %
voronoi_lg_large_7	16.4%	1.0 %	0.5 %
voronoi_lg_large_8	14.1%	3.7 %	1.2 %
voronoi_lg_large_9	24.4%	11.3 %	1.9 %
voronoi_lg_large_10	21.6%	5.7 %	1.5 %
voronoi_lg_large_11	16.5%	11.8 %	1.7 %
voronoi_lg_large_12	15.5%	3.4 %	0.5 %
voronoi_lg_large_13	19.9%	0.7 %	0.4 %
voronoi_lg_large_14	22.9%	1.7 %	0.4 %
voronoi_lg_large_15	23.5%	3.0 %	0.7 %
voronoi_lg_large_16	21.8%	12.7 %	1.7 %
voronoi_lg_large_17	20.0%	0.8 %	0.6 %
voronoi_lg_large_18	19.8%	6.0 %	0.7 %
voronoi_lg_large_19	15.9%	0.7 %	0.8 %
voronoi_lg_large_20	28.1%	3.3 %	0.9 %
voronoi_lg_large_21	19.7%	4.6 %	0.5 %
voronoi_lg_large_22	25.2%	12.0 %	1.6 %
voronoi_lg_large_23	16.9%	4.2 %	1.6 %
voronoi_lg_large_24	21.8%	2.6 %	1.4 %
voronoi_lg_large_25	18.8%	0.8 %	0.6 %
voronoi_lg_large_26	18.4%	1.6 %	0.9 %
voronoi_lg_large_27	20.4%	7.8 %	1.1 %
voronoi_lg_large_28	17.2%	2.3 %	1.1 %
voronoi_lg_large_29	23.0%	0.6 %	0.6 %
voronoi_lg_large_30	17.5%	5.1 %	1.4 %
voronoi_lg_large_31	13.9%	6.2 %	1.0 %
voronoi_lg_large_32	24.8%	1.6 %	1.4 %
voronoi_lg_large_33	20.5%	1.1 %	0.8 %
voronoi_lg_large_34	15.5%	2.4 %	0.8 %
voronoi_lg_large_35	13.6%	8.0 %	1.5 %
voronoi_lg_large_36	16.6%	0.9 %	0.6 %
voronoi_lg_large_37	20.5%	1.3 %	1.5 %
voronoi_lg_large_38	17.5%	4.3 %	0.5 %
voronoi_lg_large_39	19.5%	2.6 %	1.4 %
voronoi_lg_large_40	21.2%	3.3 %	0.6 %
voronoi_lg_large_41	25.2%	5.5 %	1.2 %

Table B.6.: Computational results for the branch-and-price approach.

		Time [s]		Gap		
Instance	nodes	B&P	root	B&P	root	# opt root
ger_lg_tiny_1	10	15.0	1.0	0.0	0.0	1
ger_lg_tiny_2	3	4.1	1.0	0.0	0.0	1
ger_lg_tiny_3	1	1.1	1.2	0.0	0.0	1
ger_lg_tiny_4	1	1.7	1.8	0.0	0.0	1
ger_lg_small_1	1	3.6	3.7	0.0	0.0	1
ger_lg_small_2	1	3.8	3.7	0.0	0.0	1
ger_lg_small_3	1	4.9	4.4	0.0	0.0	1
ger_lg_small_4	1	1.9	1.9	0.0	0.0	1
ger_lg_small_5	1	13.3	12.9	0.0	0.0	1
ger_lg_small_6	8	49.7	4.5	0.0	0.0	1
ger_lg_small_7	1	3.0	2.9	0.0	0.0	1
ger_lg_small_8	1	5.8	5.4	0.0	0.0	1
ger_lg_small_9	4	48.6	7.0	0.0	0.0	1
ger_lg_medium_1	56	1027.9	13.0	0.0	0.0	1
ger_lg_medium_2	20	308.7	14.8	0.0	0.0	1
ger_lg_medium_3	1	12.6	12.1	0.0	0.0	1
ger_lg_medium_4	511	20284.0	27.4	0.0	0.0	1
ger_lg_medium_5	2	79.9	25.6	0.0	0.0	1
ger_lg_medium_6	1	18.8	18.0	0.0	0.0	1
ger_lg_medium_7	12	217.2	15.8	0.0	0.0	1
ger_lg_medium_8	1	72.9	70.9	0.0	0.0	1
ger_lg_medium_9	1	27.6	26.0	0.0	0.0	1
ger_lg_large_1	1	60.7	60.1	0.0	0.0	1
ger_lg_large_2	2	412.1	149.8	0.0	0.0	1
tree_lg_1	1	21.8	21.0	0.0	0.0	1
tree_lg_2	1	14.5	14.1	0.0	0.0	1
tree_lg_3	41	1309.2	18.2	0.0	0.2	0
tree_lg_4	1	25.4	24.1	0.0	0.0	1
tree_lg_5	504	15697.3	15.2	0.0	0.1	0
tree_lg_6	8	152.3	12.1	0.0	0.0	1
tree_lg_7	4	77.1	11.2	0.0	0.3	0
tree_lg_8	3	79.8	17.4	0.0	0.0	1
tree_lg_9	269	15719.1	21.0	0.0	0.0	1
tree_lg_10	1	20.1	18.7	0.0	0.0	1
tree_lg_11	17	532.1	17.1	0.0	0.0	1
tree_lg_12	1	21.1	20.3	0.0	0.0	1
tree_lg_13	1	23.0	21.8	0.0	0.0	1
tree_lg_14	1	21.3	20.1	0.0	0.0	1
tree_lg_15	1	19.4	17.9	0.0	0.0	1
tree_lg_16	1	22.5	21.6	0.0	0.0	1
tree_lg_17	2	42.9	14.8	0.0	0.0	1
tree_lg_18	2	49.7	17.8	0.0	0.0	1
tree_lg_19	1	17.4	17.1	0.0	0.0	1
tree_lg_20	1	12.7	12.2	0.0	0.0	1
tree_lg_21	1	21.5	20.8	0.0	0.0	1
tree_lg_22	15	365.6	14.9	0.0	0.0	1
tree_lg_23	2	53.4	19.3	0.0	2.3	0
tree_lg_24	1	17.0	16.3	0.0	0.0	1
tree_lg_25	3	91.7	22.7	0.0	0.0	1

Instance	nodes	Time [s]		Gap		# opt root
		B&P	root	B&P	root	
voronoi_lg_medium_1	4	374.0	72.9	0.0	0.0	1
voronoi_lg_medium_2	2	93.7	35.7	0.0	0.0	1
voronoi_lg_medium_3	204	18814.7	52.6	0.0	0.0	1
voronoi_lg_medium_4	1	81.1	80.0	0.0	0.0	1
voronoi_lg_medium_5	177	16089.2	61.2	0.0	0.6	0
voronoi_lg_medium_6	1	56.3	51.8	0.0	0.0	1
voronoi_lg_medium_7	1	76.0	72.3	0.0	0.0	1
voronoi_lg_medium_8	4	457.0	81.4	0.0	0.0	1
voronoi_lg_medium_9	1	92.3	88.8	0.0	0.0	1
voronoi_lg_large_1	1	109.5	105.6	0.0	0.0	1
voronoi_lg_large_2	62	5081.5	74.8	0.0	0.1	0
voronoi_lg_large_3	1	55.7	53.3	0.0	0.0	1
voronoi_lg_large_4	1	69.9	69.1	0.0	0.0	1
voronoi_lg_large_5	1	41.9	39.6	0.0	0.0	1
voronoi_lg_large_6	626	86400.0	125.6	0.5	0.8	0
voronoi_lg_large_7	251	17690.4	46.4	0.0	0.0	1
voronoi_lg_large_8	33	4298.7	104.7	0.0	0.5	0
voronoi_lg_large_9	3	268.0	84.0	0.0	0.5	0
voronoi_lg_large_10	55	4666.2	50.3	0.0	0.0	1
voronoi_lg_large_11	15	1381.6	67.0	0.0	0.0	1
voronoi_lg_large_12	2	383.1	196.7	0.0	0.0	1
voronoi_lg_large_13	107	19323.3	130.4	0.0	0.0	1
voronoi_lg_large_14	17	1573.2	60.9	0.0	0.0	1
voronoi_lg_large_15	86	13252.0	80.1	0.0	1.5	0
voronoi_lg_large_16	114	10756.1	86.1	0.0	0.1	0
voronoi_lg_large_17	5	829.0	127.6	0.0	0.0	1
voronoi_lg_large_18	473	86400.0	157.2	0.4	0.5	0
voronoi_lg_large_19	2	618.9	214.5	0.0	0.0	1
voronoi_lg_large_20	1	46.7	45.6	0.0	0.0	1
voronoi_lg_large_21	1	144.4	142.6	0.0	0.0	1
voronoi_lg_large_22	134	9534.9	56.7	0.0	0.1	0
voronoi_lg_large_23	964	86400.0	87.8	1.9	1.9	0
voronoi_lg_large_24	15	1578.4	89.6	0.0	0.5	0
voronoi_lg_large_25	1	227.2	222.1	0.0	0.0	1
voronoi_lg_large_26	10	1440.8	81.8	0.0	1.0	0
voronoi_lg_large_27	1	46.4	46.5	0.0	0.0	1
voronoi_lg_large_28	1	49.2	47.2	0.0	0.0	1
voronoi_lg_large_29	21	2296.0	67.8	0.0	1.9	0
voronoi_lg_large_30	1	71.5	70.9	0.0	0.0	1
voronoi_lg_large_31	63	7288.5	114.1	0.0	0.0	1
voronoi_lg_large_32	5	616.0	106.4	0.0	0.0	1
voronoi_lg_large_33	1	88.7	86.3	0.0	0.0	1
voronoi_lg_large_34	158	15213.4	104.8	0.0	0.0	1
voronoi_lg_large_35	9	804.3	105.3	0.0	0.0	1
voronoi_lg_large_36	1	75.5	73.3	0.0	0.0	1
voronoi_lg_large_37	1	73.4	71.7	0.0	0.0	1
voronoi_lg_large_38	1	75.3	74.7	0.0	0.0	1
voronoi_lg_large_39	98	10016.1	90.7	0.0	0.0	1
voronoi_lg_large_40	1	59.0	56.3	0.0	0.0	1
voronoi_lg_large_41	1	56.7	56.1	0.0	0.0	1

Bibliography

[AB96] I. Averbakh and O. Berman. "A heuristic with worst-case analysis for minimax routing of two travelling salesmen on a tree". In: *Discrete Applied Mathematics* 68.1-2 (1996), pp. 17–32.

[AB97] I. Averbakh and O. Berman. "$(p-1)(p+1)$-approximate algorithms for p-traveling salesmen problems on a tree with minmax objective". In: *Discrete Applied Mathematics* 75.3 (1997), pp. 201–216.

[ABH17] N. Alon, T. Bohman, and H. Huang. "More on the bipartite decomposition of random graphs". In: *Journal of Graph Theory* 84.1 (2017), pp. 45–52.

[ABP93] E. Agasi, R. I. Becker, and Y. Perl. "A shifting algorithm for constrained min-max partition on trees". In: *Discrete Applied Mathematics* 45.1 (1993), pp. 1–28.

[ABR92] S. G. de Amorim, J.-P. Barthélemy, and C. C. Ribeiro. "Clustering and clique partitioning: simulated annealing and tabu search approaches". In: *Journal of Classification* 9.1 (1992), pp. 17–41.

[AHL06] E. M. Arkin, R. Hassin, and A. Levin. "Approximations for minimum and min-max vehicle routing problems". In: *Journal of Algorithms* 59.1 (2006), pp. 1–18.

[AK20] Z. Ales and A. Knippel. "The K-partitioning problem: Formulations and branch-and-cut". In: *Networks* 76 (2020), pp. 323–349.

[AKP16] Z. Ales, A. Knippel, and A. Pauchet. "Polyhedral combinatorics of the k-partitioning problem with representative variables". In: *Discrete Applied Mathematics* 211 (2016), pp. 1–14.

[ÁMLM13a] E. Álvarez-Miranda, I. Ljubić, and P. Mutzel. "The maximum weight connected subgraph problem". In: *Facets of Combinatorial Optimization*. Springer, 2013, pp. 245–270.

[ÁMLM13b] E. Álvarez-Miranda, I. Ljubić, and P. Mutzel. "The rooted maximum node-weight connected subgraph problem". In: *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer. 2013, pp. 300–315.

[ÁMS17] E. Álvarez-Miranda and M. Sinnl. "A Relax-and-Cut framework for large-scale maximum weight connected subgraph problems". In: *Computers & Operations Research* 87 (2017), pp. 63–82.

[AT85] N. Alon and M. Tarsi. "Covering multigraphs by simple circuits". In: *SIAM Journal on Algebraic Discrete Methods* 6.3 (1985), pp. 345–350.

[AVJ98] J. Amilhastre, M.-C. Vilarem, and P. Janssen. "Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs". In: *Discrete Applied Mathematics* 86.2-3 (1998), pp. 125–144.

[Ach+20] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger. "Presolve reductions in mixed integer programming". In: *INFORMS Journal on Computing* 32.2 (2020), pp. 473–506.

[Ale+07] L. Aleksandrov, H. Djidjev, H. Guo, and A. Maheshwari. "Partitioning planar graphs with costs and weights". In: *Journal of Experimental Algorithmics* 11 (2007).

[Alo86] N. Alon. "Covering graphs by the minimum number of equivalence relations". In: *Combinatorica* 6.3 (1986), pp. 201–206.

[Ami+02] S. J. d'Amico, S.-J. Wang, R. Batta, and C. M. Rump. "A simulated annealing approach to police district design". In: *Computers & Operations Research* 29.6 (2002), pp. 667–684.

[Ane80] Y. P. Aneja. "An integer linear programming approach to the Steiner problem in graphs". In: *Networks* 10.2 (1980), pp. 167–178.

[Anj+13] M. F. Anjos, F. Liers, G. Pardella, and A. Schmutzer. "Engineering branch-and-cut algorithms for the equicut problem". In: *Discrete Geometry and Optimization*. Springer, 2013, pp. 17–32.

[Apo+08] N. Apollonio, I. Lari, F. Ricca, B. Simeone, and J. Puerto. "Polynomial algorithms for partitioning a tree into single-center subtrees to minimize flat service costs". In: *Networks* 51.1 (2008), pp. 78–89.

[App+02] D. Applegate, W. Cook, S. Dash, and A. Rohe. "Solution of a min-max vehicle routing problem". In: *INFORMS Journal on computing* 14.2 (2002), pp. 132–143.

[Arm+08a] M. Armbruster, M. Fügenschuh, C. Helmberg, and A. Martin. "A comparative study of linear and semidefinite branch-and-cut methods for solving the minimum graph bisection problem". In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 2008, pp. 112–124.

[Arm+08b] M. Armbruster, C. Helmberg, M. Fügenschuh, and A. Martin. "On the graph bisection cut polytope". In: *SIAM Journal on Discrete Mathematics* 22.3 (2008), pp. 1073–1098.

[BCR97] L. Brunetta, M. Conforti, and G. Rinaldi. "A branch-and-cut algorithm for the equicut problem". In: *Mathematical Programming* 78.2 (1997), pp. 243–263.

[BEL03] B. Bozkaya, E. Erkut, and G. Laporte. "A tabu search heuristic and adaptive memory procedure for political districting". In: *European Journal of Operational Research* 144.1 (2003), pp. 12–26.

[BES19] R. Borndörfer, Z. Elijazyfer, and S. Schwartz. *Approximating Balanced Graph Partitions*. Tech. rep. 19-25. Zuse Institute Berlin, 2019.

[BH03] G. D. Bader and C. W. Hogue. "An automated method for finding molecular complexes in large protein interaction networks". In: *BMC bioinformatics* 4.1 (2003), p. 2.

[BJ17] F. Botler and A. Jiménez. "On path decompositions of $2k$-regular graphs". In: *Discrete Mathematics* 340.6 (2017), pp. 1405–1411.

[BJ92] T. N. Bui and C. Jones. "Finding good approximate vertex and edge partitions is NP-hard". In: *Information Processing Letters* 42.3 (1992), pp. 153–159.

[BJJ83] J. C. Bermond, B. Jackson, and F. Jaeger. "Shortest coverings of graphs with cycles". In: *Journal of Combinatorial Theory, Series B* 35.3 (1983), pp. 297–308.

[BKN15] M. D. Biha, H. L. Kerivin, and P. H. Ng. "Polyhedral study of the connected subgraph problem". In: *Discrete Mathematics* 338.1 (2015), pp. 80–92.

[BKV12] M. Blanchette, E. Kim, and A. Vetta. "Clique cover on sparse networks". In: *2012 Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments*. SIAM. 2012, pp. 93–102.

[BL84] S. N. Bhatt and F. T. Leighton. "A framework for solving VLSI graph layout problems". In: *Journal of Computer and System Sciences* 28.2 (1984), pp. 300–343.

[BLP05] F. Bacao, V. Lobo, and M. Painho. "Applying genetic algorithms to zone design". In: *Soft Computing* 9.5 (2005), pp. 341–348.

[BPS80] R. I. Becker, Y. Perl, and S. R. Schach. "A shifting algorithm for min-max tree partitioning". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 1980, pp. 64–75.

[BRVN16] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse. "The vehicle routing problem: State of the art classification and review". In: *Computers & Industrial Engineering* 99 (2016), pp. 300–313.

[BS01] M. Bläser and B. Siebert. "Computing cycle covers without short cycles". In: *European Symposium on Algorithms*. Springer. 2001, pp. 368–379.

[BS05] E. Balas and C. C. de Souza. "The vertex separator problem: A polyhedral investigation". In: *Mathematical Programming* 103.3 (2005), pp. 583–608.

[BS11] C.-E. Bichot and P. Siarry. *Graph partitioning*. John Wiley & Sons, 2011.

[BSS22] R. Borndörfer, S. Schwartz, and W. Surau. "Rooted maximum weight connected subgraphs with balancing and capacity constraints". In: *Proceedings of the 10th International Network Optimization Conference (INOC), Aachen, Germany*. Springer. 2022, pp. 63–68.

[BSS23] R. Borndörfer, S. Schwartz, and W. Surau. "Vertex covering with capacitated trees". In: *Networks* 81.2 (2023), pp. 253–277. DOI: 10.1002/net.22130.

[BT06] M. Behrisch and A. Taraz. "Efficiently covering complex networks with cliques of similar vertices". In: *Theoretical Computer Science* 355.1 (2006), pp. 37–47.

[BW00] R. Borndörfer and R. Weismantel. "Set packing relaxations of some integer programs". In: *Mathematical Programming* 88.3 (2000), pp. 425–450.

[BW17] S. Brandt and R. Wattenhofer. "Approximating small balanced vertex separators in almost linear time". In: *Workshop on Algorithms and Data Structures*. Springer. 2017, pp. 229–240.

[Bac+12] C. Backes, A. Rurainski, G. W. 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". In: *Nucleic acids research* 40.6 (2012), e43.

[Bag] *Mautdaten des Bundesamt für Güterverkehr*. https://www.bag.bund.de/DE/Service/Open-Data/Mautdaten-Tabellenwerk/Tabellenwerk_inhalt.html. Accessed: 2022-12-09.

[Bea84] J. E. Beasley. "An algorithm for the Steiner problem in graphs". In: *Networks* 14.1 (1984), pp. 147–159.

[Bec+01] R Becker, I. Lari, M. Lucertini, and B. Simeone. "A polynomial-time algorithm for max-min partitioning of ladders". In: *Theory of Computing Systems* 34.4 (2001), pp. 353–374.

[Bec+98] R. Becker, I. Lari, M. Lucertini, and B. Simeone. "Max-min partitioning of grid graphs into connected components". In: *Networks* 32.2 (1998), pp. 115–125.

[Bei70] L. W. Beineke. "Characterizations of derived graphs". In: *Journal of Combinatorial Theory* 9.2 (1970), pp. 129 –135. ISSN: 0021-9800.

[Ber78] J.-C. Bermond. *Couverture des arêtes d'un graphe par des graphes bipartis complets*. Tech. rep. 10. LRI - CNRS, University Paris-Sud, 1978, pp. 1–3.

[Bez+08] S. Bezrukov, D. Fronček, S. J. Rosenberg, and P. Kovář. "On biclique coverings". In: *Discrete Mathematics* 308.2-3 (2008), pp. 319–323.

[Bho+19a] S. Bhore, S. Chakraborty, S. Jana, J. S. Mitchell, S. Pandit, and S. Roy. "The balanced connected subgraph problem". In: *Conference on Algorithms and Discrete Applied Mathematics*. Springer. 2019, pp. 201–215.

[Bho+19b] S. Bhore, S. Jana, S. Pandit, and S. Roy. "Balanced connected subgraph problem in geometric intersection graphs". In: *International Conference on Combinatorial Optimization and Applications*. Springer. 2019, pp. 56–68.

[Bic11] C.-E. Bichot. "A partitioning requiring rapidity and quality: The multilevel method and partitions refinement algorithms". In: *Graph Partitioning*. John Wiley & Sons, 2011, pp. 29–63.

[Bmd] *Mautdaten des Bundesministerium für Digitales und Verkehr*. `https://bmdv.bund.de/SharedDocs/DE/Artikel/StV/Strassenverkehr/lkw-maut.html`. Accessed: 2022-12-09.

[Bon90] J. Bondy. "Small cycle double covers of graphs". In: *Cycles and Rays*. Springer, 1990, pp. 21–40.

[Bor+21] R. Borndörfer, K. Casel, D. Issac, A. Niklanovits, S. Schwartz, and Z. Zeif. "Connected k-partition of k-connected graphs and c-claw-free graphs". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Ed. by M. Wootters and L. Sanità. Vol. 207. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 27:1–27:14.

[Bra+07] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner. "On modularity clustering". In: *IEEE Transactions on Knowledge and Data Engineering* 20.2 (2007), pp. 172–188.

[Bul+16] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. "Recent advances in graph partitioning". In: *Algorithm Engineering*. Springer, 2016, pp. 117–158.

[Bus94] M. Bussieck. *The minimal cut cover of a graph*. Tech. rep. TR-94-02, Pennsylvania State University, 1994.

[CCL06] A. Costa, J.-F. Cordeau, and G. Laporte. "Steiner tree problems with profits". In: *INFOR: Information Systems and Operational Research* 44.2 (2006), pp. 99–115.

[CCY19] H. Chen, T. Cheng, and X. Ye. "Designing efficient and balanced police patrol districts on an urban street network". In: *International Journal of Geographical Information Science* 33.2 (2019), pp. 269–290.

[CES83] F. Chung, P Erdős, and J. Spencer. "On the decomposition of graphs into complete bipartite subgraphs". In: *Studies in Pure Mathematics*. Springer, 1983, pp. 95–101.

[CF06] D. Cornaz and J. Fonlupt. "Chromatic characterization of biclique covers". In: *Discrete Mathematics* 306.5 (2006), pp. 495–507.

[CFS14] D. Conlon, J. Fox, and B. Sudakov. "Short proofs of some extremal results". In: *Combinatorics, Probability and Computing* 23.1 (2014), pp. 8–28.

[CG12] C.-Y. Chen and K. Grauman. "Efficient activity detection with max-subgraph search". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 1274–1281.

[CG95] B. V. Cherkassky and A. V. Goldberg. "On implementing push-relabel method for the maximum flow problem". In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 1995, pp. 157–171.

[CGM83] P. M. Camerini, G. Galbiati, and F. Maffioli. "On the complexity of finding multi-constrained spanning trees". In: *Discrete Applied Mathematics* 5.1 (1983), pp. 39–50.

[CGP19] F. Clautiaux, J. Guillot, and P. Pesneau. "Exact approaches for solving a covering problem with capacitated subtrees". In: *Computers & Operations Research* 105 (2019), pp. 85–101.

[CGP85] D de Caen, D. A. Gregory, and N. Pullman. "Clique coverings of complements of paths and cycles". In: *North-Holland Mathematics Studies*. Vol. 115. Elsevier, 1985, pp. 257–267.

[CGR92] S. Chopra, E. R. Gorres, and M. Rao. "Solving the Steiner tree problem on a graph using branch and cut". In: *ORSA Journal on Computing* 4.3 (1992), pp. 320–335.

[CHQ10] K. M. Curtin, K. Hayslett-McCall, and F. Qiu. "Determining optimal police patrol areas with maximal covering and backup covering location models". In: *Networks and Spatial Economics* 10.1 (2010), pp. 125–145.

[CM01] M.-S. Chang and H. Müller. "On the tree-degree of graphs". In: *International Workshop on Graph-Theoretic Concepts in Computer Science*. Springer. 2001, pp. 44–54.

[CP83] L Caccetta and N. Pullman. "On clique covering numbers of regular graphs". In: *Ars Combinatoria* 15 (1983), pp. 201–230.

[CPP16] M. Cygan, M. Pilipczuk, and M. Pilipczuk. "Known algorithms for edge clique cover are probably optimal". In: *SIAM Journal on Computing* 45.1 (2016), pp. 67–83.

[CR93] S. Chopra and M. R. Rao. "The partition problem". In: *Mathematical Programming* 59.1-3 (1993), pp. 87–115.

[CR94] S. Chopra and M. R. Rao. "The Steiner tree problem I: Formulations, compositions and extension of facets". In: *Mathematical Programming* 64.1-3 (1994), pp. 209–229.

[CR95] S. Chopra and M. Rao. "Facets of the k-partition polytope". In: *Discrete Applied Mathematics* 61.1 (1995), pp. 27–48.

[CRS90] M. Conforti, M. Rao, and A. Sassano. "The equipartition polytope I, II". In: *Mathematical Programming* 49 (1990), pp. 49–70, 71–90.

[CSW07] F. Chataigner, L. R. Salgado, and Y. Wakabayashi. "Approximation and inapproximability results on balanced connected partitions of graphs". In: *Discrete Mathematics and Theoretical Computer Science* 9.1 (2007), pp. 177–192.

[CV08] M. Cavers and J. Verstraëte. "Clique partitions of complements of forests and bounded degree graphs". In: *Discrete Mathematics* 308.10 (2008), pp. 2011–2017.

[CWC13] A.-C. Chu, B. Y. Wu, and K.-M. Chao. "A linear-time algorithm for finding an edge-partition with max-min ratio at most two". In: *Discrete Applied Mathematics* 161.7-8 (2013), pp. 932–943.

[Cac+85] L. Caccetta, P. Erdős, E. T. Ordman, and N. J. Pullman. "The difference between the clique numbers of a graph". In: *Ars Combinatoria A* 19 (1985), pp. 97–106.

[Car+13] R. Carvajal, M. Constantino, M. Goycoolea, J. P. Vielma, and A. Weintraub. "Imposing connectivity constraints in forest planning models". In: *Operations Research* 61.4 (2013), pp. 824–836.

[Cas+21] K. Casel, T. Friedrich, D. Issac, A. Niklanovits, and Z. Zeif. "Balanced crown decomposition for connectivity constraints". In: *29th Annual European Symposium on Algorithms (ESA 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2021, 26:1–26:15.

[Cav05] M. S. Cavers. "Clique partitions and coverings of graphs". MA thesis. 2005.

[Cha09] M. Chan. "A survey of the cycle double cover conjecture". In: *Princeton University* (2009).

[Cha+17] T. Chakraborty, A. Dalmia, A. Mukherjee, and N. Ganguly. "Metrics for community analysis: A survey". In: *ACM Computing Surveys (CSUR)* 50.4 (2017), pp. 1–37.

[Cha+21] P. Charbit, G. Hahn, M. Kamiński, M. Lafond, N. Lichiardopol, R. Naserasr, B. Seamone, and R. Sherkati. "Edge clique covers in graphs with independence number two". In: *Journal of Graph Theory* 97.2 (2021), pp. 324–339.

[Che+00] G. Chen, M. S. Jacobson, A. E. Kézdy, J. Lehel, E. R. Scheinerman, and C. Wang. "Clique covering the edges of a locally cobipartite graph". In: *Discrete Mathematics* 219.1-3 (2000), pp. 17–26.

[Che+20] G. Chen, Y. Chen, Z.-Z. Chen, G. Lin, T. Liu, and A. Zhang. "Approximation algorithms for the maximally balanced connected graph tripartition problem". In: *Journal of Combinatorial Optimization* (2020), pp. 1–21.

[Che+21] Y. Chen, Z.-Z. Chen, G. Lin, Y. Xu, and A. Zhang. "Approximation algorithms for maximally balanced connected graph partition". In: *Algorithmica* 83.12 (2021), pp. 3715–3740.

[Chl96] J. Chlebíková. "Approximating the maximally balanced connected partition problem in graphs". In: *Information Processing Letters* 60.5 (1996), pp. 225–230.

[Cho94] S. Chopra. "The graph partitioning polytope on series-parallel and 4-wheel free graphs". In: *SIAM Journal on Discrete Mathematics* 7.1 (1994), pp. 16–31.

[Chu+10] A.-C. Chu, B. Y. Wu, H.-L. Wang, and K.-M. Chao. "A tight bound on the min-ratio edge-partitioning problem of a tree". In: *Discrete Applied Mathematics* 158.14 (2010), pp. 1471–1478.

[Chu78] F. Chung. "On partitions of graphs into trees". In: *Discrete Mathematics* 23.1 (1978), pp. 23–30.

[Chu80] F. Chung. "On the coverings of graphs". In: *Discrete Mathematics* 30.2 (1980), pp. 89–93.

[Con+07] J. Conrad, C. P. Gomes, W.-J. Van Hoeve, A. Sabharwal, and J. Suter. "Connections in networks: Hardness of feasibility versus optimality". In: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer. 2007, pp. 16–28.

[Cun19] A. S. da Cunha. "Formulation and Branch-and-cut algorithm for the Minimum Cardinality Balanced and Connected Clustering Problem". In: *9th International Network Optimization Conference Avignon, France, June 12–14, 2019*. 2019, pp. 25–30.

[DAR12] J. C. Duque, L. Anselin, and S. J. Rey. "The max-p-regions problem". In: *Journal of Regional Science* 52.3 (2012), pp. 397–419.

[DC+86] D De Caen, P Erdős, N. J. Pullmann, and N. C. Wormald. "Extremal clique coverings of complementary graphs". In: *Combinatorica* 6.4 (1986), pp. 309–314.

[DF85] M. E. Dyer and A. M. Frieze. "On the complexity of partitioning graphs into connected subgraphs". In: *Discrete Applied Mathematics* 10.2 (1985), pp. 139–153.

[DG10] B. Dilkina and C. P. Gomes. "Solving connected subgraph problems in wildlife conservation". In: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer. 2010, pp. 102–116.

[DGL92] M. Deza, M. Grötschel, and M. Laurent. "Clique-web facets for multicut polytopes". In: *Mathematics of Operations Research* 17.4 (1992), pp. 981–1000.

[DH72] W. E. Donath and A. J. Hoffman. "Algorithms for partitioning of graphs and computer logic based on eigenvectors of connection matrices". In: *IBM Technical Disclosure Bulletin* 15.3 (1972), pp. 938–944.

[DJK20] S. Das, L. Jain, and N. Kumar. "A Constant Factor Approximation for Capacitated Min-Max Tree Cover". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Ed. by J. Byrka and R. Meka. Vol. 176. Leibniz International Proceedings in Informatics. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 55:1–55:13.

[DL07] J. Dong and Y. Liu. "On the decomposition of graphs into complete bipartite graphs". In: *Graphs and Combinatorics* 23.3 (2007), pp. 255–262.

[DN15] T. Duncan and D. Nowitzki. no communication, just inspiration. Apr. 25, 2015.

[Dar+19] B. Darties, R. Giroudeau, K. Jean-Claude, and V. Pollet. "The Balanced Connected Subgraph Problem: Complexity Results in Bounded-Degree and Bounded-Diameter Graphs". In: *International Conference on Combinatorial Optimization and Applications*. Springer. 2019, pp. 449–460.

[Der+14] D. Deritei, Z. I. Lazar, I. Papp, F. Jarai-Szabo, R. Sumi, L. Varga, E. R. Regan, and M. Ercsey-Ravasz. "Community detection by graph Voronoi diagrams". In: *New Journal of Physics* 16.6 (2014), p. 063007.

[Die16] R. Diestel. *Graph theory*. 5th ed. Vol. 173. Graduate Texts in Mathematics. Springer, 2016.

[Dit+08] M. T. Dittrich, G. W. Klau, A. Rosenwald, T. Dandekar, and T. Müller. "Identifying functional modules in protein–protein interaction networks: An integrated exact approach". In: *Bioinformatics* 24.13 (2008), pp. 223–231.

[Dji00] H. N. Djidjev. "Partitioning planar graphs with vertex costs: Algorithms and applications". In: *Algorithmica* 28.1 (2000), pp. 51–75.

[Don80] A. Donald. "An upper bound for the path number of a graph". In: *Journal of Graph Theory* 4.2 (1980), pp. 189–201.

[Duh+08] C. Duhamel, L. Gouveia, P. Moura, and M. Souza. "Models and heuristics for a minimum arborescence problem". In: *Networks* 51.1 (2008), pp. 34–47.

[EFO88] P. Erdős, R. Faudree, and E. T. Ordman. "Clique partitions and clique coverings". In: *Discrete Mathematics* 72 (1988), pp. 93–101.

[EGP66] P. Erdős, A. W. Goodman, and L. Pósa. "The representation of a graph by set intersections". In: *Canadian Journal of Mathematics* 18 (1966), pp. 106–112.

[EJ73] J. Edmonds and E. L. Johnson. "Matching, Euler tours and the Chinese postman". In: *Mathematical Programming* 5.1 (1973), pp. 88–124.

[EKK14] M. El-Kebir and G. W. Klau. *Solving the maximum-weight connected subgraph problem to optimality*. 11th DIMACS Implementation Challenge Workshop. 2014.

[EP97] P Erdős and L. Pyber. "Covering a graph by complete bipartite graphs". In: *Discrete mathematics* 170.1-3 (1997), pp. 249–251.

[ERD14] M. G. Elizondo-Amaya, R. Z. Ríos-Mercado, and J. A. Díaz. "A dual bounding scheme for a territory design problem". In: *Computers & Operations Research* 44 (2014), pp. 193–205.

[Eli18] Z. Elijazyfer. "Längenbeschränkte Teilgraphenbildung zur Maut-Kontrollstreckenoptimierung". Master's thesis. Freie Universität Berlin, 2018.

[Eve+04] G. Even, N. Garg, J. Könemann, R. Ravi, and A. Sinha. "Min–max tree covers of graphs". In: *Operations Research Letters* 32.4 (2004), pp. 309–315.

[FB07] S. Fortunato and M. Barthélemy. "Resolution limit in community detection". In: *Proceedings of the National Academy of Sciences* 104.1 (2007), pp. 36–41.

[FF08] A. Fügenschuh and M. Fügenschuh. "Integer linear programming models for topology optimization in sheet metal design". In: *Mathematical Methods of Operations Research* 68.2 (2008), pp. 313–331.

[FF56] L. R. Ford and D. R. Fulkerson. "Maximal Flow Through a Network". In: *Canadian Journal of Mathematics* 8 (1956), 399–404.

[FG90] J. A. Ferland and G. Guénette. "Decision support system for the school districting problem". In: *Operations Research* 38.1 (1990), pp. 15–21.

[FH15] T. Friedrich and C. Hercher. "On the kernel size of clique cover reductions for random intersection graphs". In: *Journal of Discrete Algorithms* 34 (2015), pp. 128–136.

[FH16] S. Fortunato and D. Hric. "Community detection in networks: A user guide". In: *Physics Reports* 659 (2016), pp. 1–44.

[FH96] P. C. Fishburn and P. L. Hammer. "Bipartite dimensions and bipartite degrees of graphs". In: *Discrete Mathematics* 160.1-3 (1996), pp. 127–148.

[FHK76] G. N. Frederickson, M. S. Hecht, and C. E. Kim. "Approximation algorithms for some routing problems". In: *17th annual symposium on foundations of computer science (sfcs 1976)*. IEEE. 1976, pp. 216–227.

[FHL08] U. Feige, M. Hajiaghayi, and J. R. Lee. "Improved approximation algorithms for minimum weight vertex separators". In: *SIAM Journal on Computing* 38.2 (2008), pp. 629–657.

[FK01] Z. Füredi and A. Kündgen. "Covering a graph with cuts of minimum total size". In: *Discrete Mathematics* 237.1-3 (2001), pp. 129–148.

[FK09] Y. Faenza and V. Kaibel. "Extended formulations for packing and partitioning orbitopes". In: *Mathematics of Operations Research* 34.3 (2009), pp. 686–697.

[FL15] B. Farbstein and A. Levin. "Min–max cover of a graph with a small number of parts". In: *Discrete Optimization* 16 (2015), pp. 51–61.

[FM82] C. M. Fiduccia and R. M. Mattheyses. "A linear-time heuristic for improving network partitions". In: *19th Design Automation Conference*. IEEE. 1982, pp. 175–181.

[FP88] B. Fleischmann and J. N. Paraschis. "Solving a large scale districting problem: A case report". In: *Computers & Operations Research* 15.6 (1988), pp. 521–533.

[FR92] M. Fürer and B. Raghavachari. "Approximating the minimum degree spanning tree to within one from the optimal degree". In: *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics. 1992, pp. 317–324.

[FSS87] U. Faigle, R. Schrader, and R. Suletzki. "A cutting plane algorithm for optimal graph partitioning". In: *Methods of Operations Research* 57 (1987), pp. 109–116.

[Fan02] G. Fan. "Subgraph coverings and edge switchings". In: *Journal of Combinatorial Theory, Series B* 84.1 (2002), pp. 54–83.

[Fan05] G. Fan. "Path decompositions and Gallai's conjecture". In: *Journal of Combinatorial Theory, Series B* 93.2 (2005), pp. 117–125.

[Fan97] G. Fan. "Minimum cycle covers of graphs". In: *Journal of Graph Theory* 25.3 (1997), pp. 229–242.

[Fan98] G. Fan. "Proofs of two minimum circuit cover conjectures". In: *Journal of Combinatorial Theory, Series B* 74.2 (1998), pp. 353–367.

[Fer+96] C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey. "Formulations and valid inequalities for the node capacitated graph partitioning problem". In: *Mathematical Programming* 74.3 (1996), pp. 247–266.

[Fer+98] C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey. "The node capacitated graph partitioning problem: A computational study". In: *Mathematical Programming* 81.2 (1998), pp. 229–256.

[Fie75] M. Fiedler. "A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory". In: *Czechoslovak Mathematical Journal* 25.4 (1975), pp. 619–633.

[Fis+17] 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". In: *Mathematical Programming Computation* 9.2 (2017), pp. 203–229.

[Fjä98] P.-O. Fjällström. *Algorithms for graph partitioning: A survey.* Vol. 3. Linköping University Electronic Press Linköping, 1998.

[Fra85] P. Fraisse. "Cycle covering in bridgeless graphs". In: *Journal of Combinatorial Theory, Series B* 39.2 (1985), pp. 146–152.

[Fre91] G. N. Frederickson. "Optimal algorithms for tree partitioning". In: *SODA*. Vol. 91. 1991, pp. 168–177.

[GG82] B. Gavish and S. Graves. *Scheduling and routing in transportation and distribution systems: formulations and new relaxations.* Tech. rep. Graduate Schhol of Management, University of Rochester, 1982.

[GH94] O. Goldschmidt and D. S. Hochbaum. "A polynomial algorithm for the k-cut problem for fixed k". In: *Mathematics of Operations Research* 19.1 (1994), pp. 24–37.

[GM93] M. X. Goemans and Y.-S. Myung. "A catalog of Steiner tree formulations". In: *Networks* 23.1 (1993), pp. 19–28.

[GMT98] J. R. Gilbert, G. L. Miller, and S.-H. Teng. "Geometric mesh partitioning: Implementation and experiments". In: *SIAM Journal on Scientific Computing* 19.6 (1998), pp. 2091–2110.

[GN02] M. Girvan and M. E. Newman. "Community structure in social and biological networks". In: *Proceedings of the National Academy of Sciences* 99.12 (2002), pp. 7821–7826.

[GN70] R. S. Garfinkel and G. L. Nemhauser. "Optimal political districting by implicit enumeration techniques". In: *Management Science* 16.8 (1970), B–495.

[GP72] R. L. Graham and H. O. Pollak. "On embedding graphs in squashed cubes". In: *Graph theory and applications.* Springer, 1972, pp. 99–110.

[GP82] D. A. Gregory and N. J. Pullman. "On a clique covering problem of Orlin". In: *Discrete Mathematics* 41.1 (1982), pp. 97–99.

[GT87] E Győri and Z. Tuza. "Decompositions of graphs into complete subgraphs of given order". In: *Studia Sci. Math. Hung* 22.315-320 (1987), p. 24.

[GVHS08] C. P. Gomes, W.-J. Van Hoeve, and A. Sabharwal. "Connections in networks: A hybrid approach". In: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming.* Springer. 2008, pp. 303–307.

[GW18] S. Goderbauer and J. Winandy. *Political Districting Problem: Literature Review and Discussion with regard to Federal Elections in Germany.* Tech. rep. RWTH Aachen, 2018.

[GW89] M. Grötschel and Y. Wakabayashi. "A cutting plane algorithm for a clustering problem". In: *Mathematical Programming* 45.1-3 (1989), pp. 59–96.

[GW90] M. Grötschel and Y. Wakabayashi. "Facets of the clique partitioning polytope". In: *Mathematical Programming* 47.1-3 (1990), pp. 367–387.

[GWS99] D. A. Gregory, V. L. Watts, and B. L. Shader. "Biclique decompositions and Hermitian rank". In: *Linear Algebra and its Applications* 292.1-3 (1999), pp. 267–280.

[Gir+21] A. Girão, B. Granet, D. Kühn, and D. Osthus. "Path and cycle decompositions of dense graphs". In: *Journal of the London Mathematical Society* 104.3 (2021), pp. 1085–1134.

[Goe94a] M. X. Goemans. "Arborescence polytopes for series-parallel graphs". In: *Discrete Applied Mathematics* 51.3 (1994), pp. 277–289.

[Goe94b] M. X. Goemans. "The Steiner tree polytope and related polyhedra". In: *Mathematical Programming* 63.1-3 (1994), pp. 157–182.

[Gor96] A. Gordon. "A survey of constrained classification". In: *Computational Statistics & Data Analysis* 21.1 (1996), pp. 17–29.

[Gra+07] J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, H.-P. Piepho, and R. Schmid. "Algorithms for compact letter displays: Comparison and evaluation". In: *Computational Statistics & Data Analysis* 52.2 (2007), pp. 725–736.

[Gra+09] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. "Data reduction and exact algorithms for clique cover". In: *Journal of Experimental Algorithmics)* 13 (2009), pp. 2–2.

[Gün07] O. Günlük. "A new min-cut max-flow ratio for multicommodity flows". In: *SIAM Journal on Discrete Mathematics* 21.1 (2007), pp. 1–15.

[Gyá90] A. Gyárfás. "A simple lower bound on edge coverings by cliques". In: *Discrete Mathematics* 85.1 (1990), pp. 103–104.

[HFV99] S. Hanafi, A. Freville, and P. Vaca. "Municipal solid waste collection: An effective data structure for solving the sectorization problem with local search methods". In: *INFOR: Information Systems and Operational Research* 37.3 (1999), pp. 236–254.

[HHM77] F. Harary, D. Hsu, and Z. Miller. "The biparticity of a graph". In: *Journal of Graph Theory* 1.2 (1977), pp. 131–133.

[HJ97] P. Hansen and B. Jaumard. "Cluster analysis and Mathematical Programming". In: *Mathematical Programming* 79.1-3 (1997), pp. 191–215.

[HJM90] P. Hansen, B. Jaumard, and K Musitu. "Weight constrained maximum split clustering". In: *Journal of Classification* 7.2 (1990), pp. 217–240.

[HKT95] T. C. Hu, A. B. Kahng, and C.-W. A. Tsao. "Old bachelor acceptance: A new class of non-monotone threshold accepting methods". In: *ORSA Journal on Computing* 7.4 (1995), pp. 417–425.

[HM03] P. Hansen and N. Mladenović. "Variable neighborhood search". In: *Handbook of metaheuristics*. Springer, 2003, pp. 145–184.

[HO01] D. S. Hochbaum and E. V. Olinick. "The bounded cycle-cover problem". In: *INFORMS Journal on Computing* 13.2 (2001), pp. 104–119.

[HP94] D. S. Hochbaum and A. Pathria. *Node-optimal connected k-subgraphs*. manuscript, UC Berkeley, 1994.

[HPK11] J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[HR95] M. T. Heath and P. Raghavan. "A Cartesian parallel nested dissection algorithm". In: *SIAM Journal on Matrix Analysis and Applications* 16.1 (1995), pp. 235–253.

[HS71] S. W. Hess and S. A. Samuels. "Experiences with a sales districting model: criteria and implementation". In: *Management Science* 18.4 (1971), pp. 41–54.

[HT73] J. Hopcroft and R. Tarjan. "Efficient Algorithms for Graph Manipulation". In: *Communications of the ACM* 16.6 (1973), pp. 372–378.

[Hal70] K. M. Hall. "An r-dimensional quadratic placement algorithm". In: *Management science* 17.3 (1970), pp. 219–229.

[Han+03] P. Hansen, B. Jaumard, C. Meyer, B. Simeone, and V. Doring. "Maximum split clustering under connectivity constraints". In: *Journal of Classification* 20.2 (2003), pp. 143–180.

[Hes+65] S. W. Hess, J. Weaver, H. Siegfeldt, J. Whelan, and P. Zitlau. "Nonpartisan political redistricting by computer". In: *Operations Research* 13.6 (1965), pp. 998–1006.

[Hoj+21] C. Hojny, I. Joormann, H. Lüthen, and M. Schmidt. "Mixed-integer programming techniques for the connected max-k-cut problem". In: *Mathematical Programming Computation* 13.1 (2021), pp. 75–132.

[Hol+10] M. Holzer, F. Schulz, D. Wagner, G. Prasinos, and C. Zaroliagis. "Engineering planar separator algorithms". In: *Journal of Experimental Algorithmics* 14 (2010), pp. 1–31.

[Hoo92] D. N. Hoover. "Complexity of graph covering problems for graphs of low degree". In: *Journal of Combinatorial Mathematics and Combinatorial Computing* 11.187-208 (1992), p. 7.

[IMM05] N. Immorlica, M. Mahdian, and V. S. Mirrokni. "Cycle cover with short cycles". In: *Annual Symposium on Theoretical Aspects of Computer Science*. Springer. 2005, pp. 641–653.

[IZN06] T. Ito, X. Zhou, and T. Nishizeki. "Partitioning a graph of bounded tree-width to connected subgraphs of almost uniform size". In: *Journal of Discrete Algorithms* 4.1 (2006), pp. 142–154.

[Ide+02] T. Ideker, O. Ozier, B. Schwikowski, and A. F. Siegel. "Discovering regulatory and signalling circuits in molecular interaction networks". In: *Bioinformatics* 18 (2002), pp. 233–240.

[Ita+81] A. Itai, R. J. Lipton, C. H. Papadimitriou, and M. Rodeh. "Covering graphs by simple circuits". In: *SIAM Journal on Computing* 10.4 (1981), pp. 746–750.

[Ito+12] T. Ito, T. Nishizeki, M. Schröder, T. Uno, and X. Zhou. "Partitioning a weighted tree into subtrees with weights in a given range". In: *Algorithmica* 62.3-4 (2012), pp. 823–841.

[JH19] R. Javadi and S. Hajebi. "Edge clique cover of claw-free graphs". In: *Journal of Graph Theory* 90.3 (2019), pp. 311–405.

[JK09] S. Jukna and A. S. Kulikov. "On covering graphs by complete bipartite subgraphs". In: *Discrete Mathematics* 309.10 (2009), pp. 3399–3403.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. "Data clustering: A review". In: *ACM computing surveys (CSUR)* 31.3 (1999), pp. 264–323.

[JMN93] E. L. Johnson, A. Mehrotra, and G. L. Nemhauser. "Min-cut clustering". In: *Mathematical Programming* 62.1-3 (1993), pp. 133–151.

[JMO16] R. Javadi, Z. Maleki, and B. Omoomi. "Local clique covering of claw-free graphs". In: *Journal of Graph Theory* 81.1 (2016), pp. 92–104.

[JN83] D. S. Johnson and K. Niemi. "On knapsacks, partitions, and a new dynamic programming technique for trees". In: *Mathematics of Operations Research* 8.1 (1983), pp. 1–14.

[JRP85] M. Jünger, G. Reinelt, and W. R. Pulleyblank. "On partitioning the edges of graphs into connected subgraphs". In: *Journal of Graph Theory* 9.4 (1985), pp. 539–549.

[Jac93] B. Jackson. "On circuit covers, circuit decompositions and Euler tours of graphs". In: *Surveys in Combinatorics*. Vol. 187. Cambridge Univ. Press Cambridge, 1993, pp. 191–210.

[Jae85] F. Jaeger. "A survey of the cycle double cover conjecture". In: *North-Holland Mathematics Studies*. Vol. 115. Elsevier, 1985, pp. 1–12.

[Jav+18] M. A. Javed, M. S. Younis, S. Latif, J. Qadir, and A. Baig. "Community detection in networks: A multidisciplinary review". In: *Journal of Network and Computer Applications* 108 (2018), pp. 87–111.

[Jor13] A. Jorati. "Approximation algorithms for some min-max vehicle routing problems". MA thesis. University of Alberta, 2013.

[KK98] G. Karypis and V. Kumar. "Multilevelk-way partitioning scheme for irregular graphs". In: *Journal of Parallel and Distributed computing* 48.1 (1998), pp. 96–129.

[KL70] B. W. Kernighan and S. Lin. "An efficient heuristic procedure for partitioning graphs". In: *The Bell system technical journal* 49.2 (1970), pp. 291–307.

[KLS91] B. Korte, L. Lovász, and R. Schrader. *Greedoids*. Vol. 4. Algorithms and Combinatorics. Springer-Verlag, 1991.

[KLT15] T.-W. Kuo, K. C.-J. Lin, and M.-J. Tsai. "Maximizing submodular set function with connectivity constraint: Theory and application to networks". In: *IEEE/ACM Transactions on Networking* 23.2 (2015), pp. 533–546.

[KM77] S. Kundu and J. Misra. "A linear tree partitioning algorithm". In: *SIAM Journal on Computing* 6.1 (1977), pp. 151–154.

[KM98] T. Koch and A. Martin. "Solving Steiner tree problems in graphs to optimality". In: *Networks* 32.3 (1998), pp. 207–232.

[KP08] V. Kaibel and M. Pfetsch. "Packing and partitioning orbitopes". In: *Mathematical Programming* 114.1 (2008), pp. 1–36.

[KPH93] B. N. Khoury, P. M. Pardalos, and D. W. Hearn. "Equivalent Formulations for the Steiner Problem in Graphs". In: *Network Optimization Problems: Algorithms, Applications And Complexity*. World Scientific, 1993, pp. 111–123.

[KPJ04] A. D. King, N. Pržulj, and I. Jurisica. "Protein complex prediction via cost-based clustering". In: *Bioinformatics* 20.17 (2004), pp. 3013–3020.

[KPP11] V. Kaibel, M. Peinhardt, and M. E. Pfetsch. "Orbitopal fixing". In: *Discrete Optimization* 8.4 (2011), pp. 595–610.

[KR19] J. Kalcsics and R. Z. Ríos-Mercado. "Districting problems". In: *Location Science*. Springer, 2019, pp. 705–743.

[KS05] A. Kündgen and M. Spangler. "A bound on the total size of a cut cover". In: *Discrete mathematics* 296.1 (2005), pp. 121–128.

[KS14] M. R. Khani and M. R. Salavatipour. "Improved approximation algorithms for the min-max tree cover and bounded tree cover problems". In: *Algorithmica* 69.2 (2014), pp. 443–460.

[KSW78] L. T. Kou, L. J. Stockmeyer, and C.-K. Wong. "Covering edges by cliques with regard to keyword conflicts and intersection graphs". In: *Communications of the ACM* 21.2 (1978), pp. 135–139.

[KU16] K. Knauer and T. Ueckerdt. "Three ways to cover a graph". In: *Discrete Mathematics* 339.2 (2016), pp. 745–758.

[KZ14] H. Kerivin and J. Zhao. "Polyhedral study for the maximum bounded r-tree problem". In: *2014 International Conference on Control, Decision and Information Technologies (CoDIT)*. IEEE. 2014, pp. 140–145.

[Kel73] E. Kellerman. "Determination of keyword conflict". In: *IBM Technical Disclosure Bulletin* 16.2 (1973), pp. 544–546.

[Kim+11] J. Kim, I. Hwang, Y.-H. Kim, and B.-R. Moon. "Genetic approaches for graph partitioning: A survey". In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. 2011, pp. 473–480.

[Kob+19] Y. Kobayashi, K. Kojima, N. Matsubara, T. Sone, and A. Yamamoto. "Algorithms and Hardness Results for the Maximum Balanced Connected Subgraph Problem". In: *International Conference on Combinatorial Optimization and Applications*. Springer. 2019, pp. 303–315.

[Kor09] R. E. Korf. "Multi-way number partitioning". In: *Proceedings of the 21st international Joint Conference on Artifical Intelligence*. 2009, pp. 538–543.

[Krá+19] D. Král, B. Lidický, T. L. Martins, and Y. Pehova. "Decomposing graphs into edges and triangles". In: *Combinatorics, Probability and Computing* 28.3 (2019), pp. 465–472.

[LD96] H. F. Lee and D. R. Dooly. "Algorithms for the constrained maximum-weight connected graph problem". In: *Naval Research Logistics (NRL)* 43.7 (1996), pp. 985–1008.

[LD98] H. F. Lee and D. R. Dooly. "Decomposition algorithms for the maximum-weight connected graph problem". In: *Naval Research Logistics (NRL)* 45.8 (1998), pp. 817–837.

[LK13] D. LaSalle and G. Karypis. "Multi-threaded graph partitioning". In: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE. 2013, pp. 225–236.

[LL20] M. Lin and R. J. La. "Miniature Robot Path Planning for Bridge Inspection: Min-Max Cycle Cover-Based Approach". In: *arXiv preprint arXiv:2003.12134* (2020).

[LÖ10] M. Labbé and F. A. Özsoy. "Size-constrained graph partitioning polytopes". In: *Discrete Mathematics* 310.24 (2010), pp. 3473–3493.

[LPS93] M. Lucertini, Y. Perl, and B. Simeone. "Most uniform path partitioning and its use in image processing". In: *Discrete Applied Mathematics* 42.2-3 (1993), pp. 227–256.

[LR03] A. Lisser and F. Rendl. "Graph partitioning using linear and semidefinite programming". In: *Mathematical Programming* 95.1 (2003), pp. 91–101.

[LT80] R. J. Lipton and R. E. Tarjan. "Applications of a planar separator theorem". In: *SIAM Journal on Computing* 9.3 (1980), pp. 615–627.

[LTVM15] D. Liu, S. Trajanovski, and P. Van Mieghem. "ILIGRA: An efficient inverse line graph algorithm". In: *Journal of Mathematical Modelling and Algorithms in Operations Research* 14.1 (2015), pp. 13–33.

[LY94] C. Lund and M. Yannakakis. "On the hardness of approximating minimization problems". In: *Journal of the ACM* 41.5 (1994), pp. 960–981.

[LZ21] J. Li and P. Zhang. "New approximation algorithms for the rooted budgeted cycle cover problem". In: *International Conference on Combinatorial Optimization and Applications*. Springer. 2021, pp. 167–179.

[Lap92] G. Laporte. "The vehicle routing problem: An overview of exact and approximate algorithms". In: *European Journal of Operational Research* 59.3 (1992), pp. 345–358.

[Lar+16] I. Lari, F. Ricca, J. Puerto, and A. Scozzari. "Partitioning a graph into connected components with fixed centers and optimizing cost-based objective functions or equipartition criteria". In: *Networks* 67.1 (2016), pp. 69–81.

[Lei+18] M. Leitner, I. Ljubić, M. Luipersbeck, and M. Sinnl. "A dual ascent-based branch-and-bound framework for the prize-collecting steiner tree and related problems". In: *INFORMS Journal on Computing* 30.2 (2018), pp. 402–420.

[Lju+06] I. Ljubić, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti. "An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem". In: *Mathematical Programming* 105.2-3 (2006), pp. 427–449.

[Lju20] I. Ljubić. "Solving Steiner trees: Recent advances, challenges, and perspectives". In: *Networks* (2020), pp. 177–204.

[Lov68] L. Lovász. "On covering of graphs". In: *Theory of Graphs (Proc. Colloq., Tihany, 1966)*. Academic Press New York. 1968, pp. 231–236.

[Luk74] J. A. Lukes. "Efficient algorithm for the partitioning of trees". In: *IBM Journal of Research and Development* 18.3 (1974), pp. 217–224.

[Lüt18] H. Lüthen. "Partitioning into Isomorphic or Connected Subgraphs". PhD thesis. Technische Universität, 2018.

[MDZ00] I. Méndez Díaz and P. Zabala. "A polyhedral approach for graph coloring". In: *Electronic Notes in Discrete Mathematics* 7 (2000), pp. 178–181.

[MJN98] A. Mehrotra, E. L. Johnson, and G. L. Nemhauser. "An optimization based heuristic for political districting". In: *Management Science* 44.8 (1998), pp. 1100–1114.

[MLT95] Y.-S. Myung, C.-H. Lee, and D.-W. Tcha. "On the generalized minimum spanning tree problem". In: *Networks* 26.4 (1995), pp. 231–241.

[MOW22] P. F. Moura, M. J. Ota, and Y. Wakabayashi. "Approximation and Parameterized Algorithms for Balanced Connected Partition Problems". In: *Conference on Algorithms and Discrete Applied Mathematics*. Springer. 2022, pp. 211–223.

[MPR95] S. D. Monson, N. J. Pullman, and R. Rees. "A survey of clique and biclique coverings and factorizations of (0, 1)-matrices". In: *Bull. Inst. Combin. Appl* 14 (1995), pp. 17–86.

[MR05] T. L. Magnanti and S Raghavan. "Strong formulations for network design problems with connectivity requirements". In: *Networks* 45.2 (2005), pp. 61–79.

[MS95] M. Maravalle and B. Simeone. "A spanning tree heuristic for regional clustering". In: *Communications in Statistics - Theory and Methods* 24.3 (1995), pp. 625–639.

[MSN97] M. Maravalle, B. Simeone, and R. Naldini. "Clustering on trees". In: *Computational Statistics & Data Analysis* 24.2 (1997), pp. 217–234.

[MT98] A. Mehrotra and M. A. Trick. "Cliques and clustering: A combinatorial approach". In: *Operations Research Letters* 22.1 (1998), pp. 1–12.

[MW95] T. L. Magnanti and L. A. Wolsey. "Optimal trees". In: *Handbooks in Operations Research and Management Science* 7 (1995), pp. 503–615.

[MWW89] S. Ma, W. D. Wallis, and J. Wu. "Clique covering of chordal graphs". In: *Utilitas Mathematica* 36 (1989), pp. 151–152.

[Mac87] N. Maculan. "The Steiner problem in graphs". In: *North-Holland Mathematics Studies*. Vol. 132. Elsevier, 1987, pp. 185–211.

[Mar10] F. Margot. "Symmetry in integer linear programming". In: *50 Years of Integer Programming 1958-2008* (2010), pp. 647–686.

[Mar+21] T Martinod, V Pollet, B Darties, R Giroudeau, and J.-C. König. "Complexity and inapproximability results for balanced connected subgraph problem". In: *Theoretical Computer Science* (2021).

[Mát06] T. Mátrai. "Covering the edges of a graph by three odd subgraphs". In: *Journal of Graph Theory* 53.1 (2006), pp. 77–82.

[Mat14] D. Matić. "A mixed integer linear programming model and variable neighborhood search for maximally balanced connected partition problem". In: *Applied Mathematics and Computation* 237 (2014), pp. 85–97.

[Miy+20] F. K. Miyazawa, P. F. Moura, M. J. Ota, and Y. Wakabayashi. "Cut and Flow Formulations for the Balanced Connected k-Partition Problem". In: *International Symposium on Combinatorial Optimization*. Springer. 2020, pp. 128–139.

[Miy+21] F. K. Miyazawa, P. F. Moura, M. J. Ota, and Y. Wakabayashi. "Partitioning a graph into balanced connected classes: Formulations, separation and experiments". In: *European Journal of Operational Research* 293.3 (2021), pp. 826–836.

[Mül96] H. Müller. "On edge perfectness and classes of bipartite graphs". In: *Discrete Mathematics* 149.1-3 (1996), pp. 159–187.

[Mur85] F. Murtagh. "A survey of algorithms for contiguity-constrained clustering and related problems". In: *The computer journal* 28.1 (1985), pp. 82–88.

[N64] C. Nash-Williams. "Decomposition of finite graphs into forests". In: *Journal of the London Mathematical Society* 1.1 (1964), pp. 12–12.

[NG04] M. E. Newman and M. Girvan. "Finding and evaluating community structure in networks". In: *Physical review E* 69.2 (2004), p. 026113.

[NO03] H. Nagamochi and K. Okada. "Polynomial time 2-approximation algorithms for the minmax subtree cover problem". In: *International Symposium on Algorithms and Computation*. Springer. 2003, pp. 138–147.

[NO04] H. Nagamochi and K. Okada. "A faster 2-approximation algorithm for the minmax p-traveling salesmen problem on a tree". In: *Discrete Applied Mathematics* 140.1-3 (2004), pp. 103–114.

[Nag05] H. Nagamochi. "Approximating the minmax rooted-subtree cover problem". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 88.5 (2005), pp. 1335–1338.

[Ngu+17] D. P. Nguyen, M. Minoux, V. H. Nguyen, T. H. Nguyen, and R. Sirdey. "Improved compact formulations for a wide class of graph partitioning problems in sparse graphs". In: *Discrete Optimization* 25 (2017), pp. 175–188.

[Nyg88] B. Nygreen. "European assembly constituencies for wales-comparing of methods for solving a political districting problem". In: *Mathematical Programming* 42.1-3 (1988), pp. 159–169.

[ORS01] M. Oosten, J. H. Rutten, and F. C. Spieksma. "The clique partitioning problem: facets and patching facets". In: *Networks* 38.4 (2001), pp. 209–226.

[Orl77] J. Orlin. "Contentment in graph theory: covering graphs with cliques". In: *Indagationes Mathematicae (Proceedings)*. Vol. 80. 5. Elsevier. 1977, pp. 406–424.

[Ost+11] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio. "Orbital branching". In: *Mathematical Programming* 126.1 (2011), pp. 147–178.

[PC80] N. J. Pullman and D. de Caen. "Clique coverings of graphs III: clique coverings of regular graphs". In: *Congressus Numerantium* 29 (1980), pp. 795–808.

[PC81] N. J. Pullman and D. de Caen. "Clique coverings of graphs I: clique partitions of regular graphs". In: *Utilitas Mathematica* 19 (1981), pp. 177–205.

[PD01] T. Polzin and S. V. Daneshmand. "A comparison of Steiner tree relaxations". In: *Discrete Applied Mathematics* 112.1-3 (2001), pp. 241–261.

[PD81] N. J. Pullman and A. Donald. "Clique coverings of graphs II: complements of cliques". In: *Utilitas Mathematica* 19 (1981), pp. 207–213.

[PR19] M. E. Pfetsch and T. Rehn. "A computational comparison of symmetry handling methods for mixed integer programs". In: *Mathematical Programming Computation* 11.1 (2019), pp. 37–93.

[PŠ19] M. Petruševski and R. Škrekovski. "Coverability of graph by three odd subgraphs". In: *Journal of Graph Theory* 92.3 (2019), pp. 304–321.

[PS81] Y. Perl and S. R. Schach. "Max-Min Tree Partitioning". In: *J. ACM* 28.1 (1981), pp. 5–15.

[PSW82] N. J. Pullman, H. Shank, and W. Wallis. "Clique coverings of graphs V: Maximal-clique partitions". In: *Bulletin of the Australian Mathematical Society* 25.3 (1982), pp. 337–356.

[PW21] D. V. Papazaharias and J. L. Walteros. *Solving graph partitioning on sparse graphs: Cuts, projections, and extended formulations*. 2021. URL: www.optimization-online.org/DB_FILE/2021/08/8535.pdf.

[Pel11] F. Pellegrini. "Parallelization of graph partitioning". In: *Graph Partitioning*. John Wiley & Sons, 2011, pp. 81–114.

[Pin14] T. Pinto. "Biclique covers and partitions". In: *The Electronic Journal of Combinatorics* 21.1 (2014), P1–19.

[Pop09] P. C. Pop. "A survey of different integer programming formulations of the generalized minimum spanning tree problem". In: *Carpathian Journal of Mathematics* (2009), pp. 104–118.

[Pop20] P. C. Pop. "The generalized minimum spanning tree problem: An overview of formulations, solution procedures and latest advances". In: *European Journal of Operational Research* 283.1 (2020), pp. 1–15.

[Pul83] N. J. Pullman. "Clique coverings of graphs – a survey". In: *Combinatorial Mathematics X*. Springer, 1983, pp. 72–85.

[Pul84] N. J. Pullman. "Clique covering of graphs IV: Algorithms". In: *SIAM Journal on Computing* 13.1 (1984), pp. 57–75.

[Pyb85] L. Pyber. "An Erdős-Gallai conjecture". In: *Combinatorica* 5.1 (1985), pp. 67–79.

[Pyb91] L. Pyber. "Covering the edges of a graph by..." In: *Sets, Graphs and Numbers, Colloquia Mathematica Societatis János Bolyai*. Vol. 60. 1991, pp. 583–610.

[Pyb96] L. Pyber. "Covering the Edges of a Connected Graph by Paths". In: *Journal of Combinatorial Theory, Series B* 66.1 (1996), pp. 152 –159.

[RF09] R. Z. Ríos-Mercado and E. Fernández. "A reactive GRASP for a commercial territory design problem with multiple balancing requirements". In: *Computers & Operations Research* 36.3 (2009), pp. 755–776.

[RFK22] D. Rehfeldt, H. Franz, and T. Koch. "Optimal connected subgraphs: Integer programming formulations and polyhedra". In: *Networks* 80 (2022), pp. 314–332.

[RH01] A. L. Rosenberg and L. S. Heath. *Graph separators, with applications*. Springer Science & Business Media, 2001.

[RK19] D. Rehfeldt and T. Koch. "Combining NP-hard reduction techniques and strong heuristics in an exact algorithm for the maximum-weight connected subgraph problem". In: *SIAM Journal on Optimization* 29.1 (2019), pp. 369–398.

[RKM19] D. Rehfeldt, T. Koch, and S. J. Maher. "Reduction techniques for the prize collecting Steiner tree problem and the maximum-weight connected subgraph problem". In: *Networks* 73.2 (2019), pp. 206–233.

[RS08] F. Ricca and B. Simeone. "Local search algorithms for political districting". In: *European Journal of Operational Research* 189.3 (2008), pp. 1409–1426.

[RS11] R. Z. Ríos-Mercado and J. C. Salazar-Acosta. "A GRASP with strategic oscillation for a commercial territory design problem with a routing budget constraint". In: *Mexican International Conference on Artificial Intelligence*. Springer. 2011, pp. 307–318.

[RSS08] F. Ricca, A. Scozzari, and B. Simeone. "Weighted Voronoi region algorithms for political districting". In: *Mathematical and Computer Modelling* 48.9-10 (2008), pp. 1468–1477.

[RSS13] F. Ricca, A. Scozzari, and B. Simeone. "Political districting: from classical models to recent approaches". In: *Annals of Operations Research* 204.1 (2013), pp. 271–299.

[RUW21] D. Rohatgi, J. C. Urschel, and J. Wellens. "Regarding two questions about clique and biclique partitions". In: *Electronic Journal of Combinatorics* 28.4 (2021).

[Río20] R. Z. Ríos-Mercado. *Optimal Districting and Territory Design*. Vol. 284. Springer, 2020.

[Rob85] F. S. Roberts. "Applications of edge coverings by cliques". In: *Discrete Applied Mathematics* 10.1 (1985), pp. 93–109.

[Rod21] M. O. Rodrigues. "Fast constructive and improvement heuristics for edge clique covering". In: *Discrete Optimization* 39 (2021), p. 100628.

[SB05] C. C. de Souza and E. Balas. "The vertex separator problem: Algorithms and computations". In: *Mathematical Programming* 103.3 (2005), pp. 609–631.

[SBS21] W. Surau, R. Borndörfer, and S. Schwartz. "Finding Minimum Balanced Separators – An Exact Approach". In: *Operations Research Proceedings*. Springer. 2021, pp. 154–159.

[SC11] C. C. de Souza and V. F. Cavalcante. "Exact algorithms for the vertex separator problem in graphs". In: *Networks* 57.3 (2011), pp. 212–230.

[SC98] D. X. Shaw and G. Cho. "The critical-item, upper bounds, and a branch-and-bound algorithm for the tree knapsack problem". In: *Networks* 31.4 (1998), pp. 205–216.

[SKK00] K. Schloegel, G. Karypis, and V. Kumar. *Graph partitioning for high performance scientific simulations*. Army High Performance Computing Research Center, 2000.

[SL95] C. C. de Souza and M. Laurent. "Some new classes of facets for the equicut polytope". In: *Discrete Applied Mathematics* 62.1-3 (1995), pp. 167–191.

[SM00] J. Shi and J. Malik. "Normalized cuts and image segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.8 (2000), pp. 888–905.

[SRC11] M. A. Salazar-Aguilar, R. Z. Ríos-Mercado, and M. Cabrera-Ríos. "New models for commercial territory design". In: *Networks and Spatial Economics* 11.3 (2011), pp. 487–507.

[SS13] P. Sanders and C. Schulz. "Think locally, act globally: Highly balanced graph partitioning". In: *International Symposium on Experimental Algorithms*. Springer. 2013, pp. 164–175.

[Sch07] S. E. Schaeffer. "Graph clustering". In: *Computer science review* 1.1 (2007), pp. 27–64.

[Sch18] V. Schreck. "Algorithmic Analysis of the Graph Segmentation Problem". Master's thesis. Freie Universität Berlin, 2018.

[Sch22] S. Schwartz. "An overview of graph covering and partitioning". In: *Discrete Mathematics* 345.8 (2022).

[Seg+07] J. Segura-Ramiro, R. Z. Ríos-Mercado, A. M. Álvarez-Socarrás, and K. de Alba Romenus. "A location-allocation heuristic for a territory design problem in a beverage distribution firm". In: *Proceedings of the 12th Annual International Conference on Industrial Engineering Theory, Applications, and Practice (IJIE)*. 2007, pp. 428–434.

[Sen01] N. Sensen. "Lower bounds and exact algorithms for the graph partitioning problem using multicommodity flows". In: *European Symposium on Algorithms*. Springer. 2001, pp. 391–403.

[Shi05] T. Shirabe. "A model of contiguity for spatial unit allocation". In: *Geographical Analysis* 37.1 (2005), pp. 2–16.

[Sør04] M. M. Sørensen. "b-tree facets for the simple graph partitioning polytope". In: *Journal of Combinatorial Optimization* 8.2 (2004), pp. 151–170.

[Sør07] M. M. Sørensen. "Facet-defining inequalities for the simple graph partitioning polytope". In: *Discrete Optimization* 4.2 (2007), pp. 221–231.

[Sør17] M. M. Sørensen. "Facets for node-capacitated multicut polytopes from path-block cycles with two common nodes". In: *Discrete Optimization* 25 (2017), pp. 120–140.

[Sur20] W. Surau. "Das homogene längenbeschränkte zusammenhängende Teilgraphenüberdeckungsproblem". Bachelor's thesis. Freie Universität Berlin, 2020.

[TT21] V. Traub and T. Tröbst. "A Fast $(2+\frac{2}{7})$-Approximation Algorithm for Capacitated Cycle Covering". In: *Mathematical Programming* (2021), pp. 1–22.

[TV02] P. Toth and D. Vigo. *The vehicle routing problem*. SIAM, 2002.

[Thi06] S. Thite. "On covering a graph optimally with induced subgraphs". In: *arXiv preprint cs/0604013* (2006).

[Tho97] C. Thomassen. "On the complexity of finding a minimum cycle cover of a graph". In: *SIAM Journal on Computing* 26.3 (1997), pp. 675–677.

[Tuz84] Z. Tuza. "Covering of graphs by complete bipartite subgraphs; complexity of 0–1 matrices". In: *Combinatorica* 4.1 (1984), pp. 111–116.

[Tve82] H. Tverberg. "On the decomposition of Kn into complete bipartite graphs". In: *Journal of Graph Theory* 6.4 (1982), pp. 493–494.

[Uiy03] C. Uiyyasathain. "Maximal-clique partitions". PhD thesis. University of Colorado at Denver, 2003.

[VB22] H. Validi and A. Buchanan. "Political districting to minimize cut edges". In: *Mathematical Programming Computation* 14 (2022), pp. 623–672.

[VBL22] H. Validi, A. Buchanan, and E. Lykhovyd. "Imposing contiguity constraints in political districting models". In: *Operations Research* 70 (2022), pp. 867–892.

[Vid78] K Vidyasankar. "Covering the edge set of a directed graph with trees". In: *Discrete Mathematics* 24.1 (1978), pp. 79–85.

[WBB17] Y. Wang, A. Buchanan, and S. Butenko. "On imposing connectivity constraints in integer programs". In: *Mathematical Programming* 166.1-2 (2017), pp. 241–271.

[WW91] W. Wallis and J. Wu. "On clique partitions of split graphs". In: *Discrete Mathematics* 92.1-3 (1991), pp. 427–429.

[Wal82] W. Wallis. "Asymptotic values of clique partition numbers". In: *Combinatorica* 2.1 (1982), pp. 99–101.

[Wat06] V. L. Watts. "Fractional biclique covers and partitions of graphs". In: *The Electronic Journal of Combinatorics* 13.1 (2006), p. 74.

[Whi32] H. Whitney. "Congruent graphs and the connectivity of graphs". In: *American Journal of Mathematics* 54.1 (1932), pp. 150–168.

[Won84] R. Wong. "Dual ascent approach for Steiner tree problems on a directed graph". In: *Mathematical Programming* 28 (1984), pp. 271–287.

[Wu12] B. Y. Wu. "Fully polynomial-time approximation schemes for the max–min connected partition problem on interval graphs". In: *Discrete Mathematics, Algorithms and Applications* 4.01 (2012).

[XLL13] W. Xu, W. Liang, and X. Lin. "Approximation algorithms for min-max cycle cover problems". In: *IEEE Transactions on Computers* 64.3 (2013), pp. 600–613.

[XOX02] Y. Xu, V. Olman, and D. Xu. "Clustering gene expression data using a graph-theoretic approach: An application of minimum spanning trees". In: *Bioinformatics* 18.4 (2002), pp. 536–545.

[XW10] Z. Xu and Q. Wen. "Approximation hardness of min–max tree covers". In: *Operations Research Letters* 38.3 (2010), pp. 169–173.

[XXZ12] Z. Xu, D. Xu, and W. Zhu. "Approximation results for a min–max location-routing problem". In: *Discrete Applied Mathematics* 160.3 (2012), pp. 306–320.

[YJC13] Z. Yan, N. Jouandeau, and A. A. Cherif. "A survey and analysis of multi-robot coordination". In: *International Journal of Advanced Robotic Systems* 10.12 (2013), p. 399.

[YL19] W. Yu and Z. Liu. "Better approximability results for min–max tree/cycle/path cover problems". In: *Journal of Combinatorial Optimization* 37.2 (2019), pp. 563–578.

[YLB19] W. Yu, Z. Liu, and X. Bao. "New approximation algorithms for the minimum cycle cover problem". In: *Theoretical Computer Science* 793 (2019), pp. 44–58.

[YLB20] W. Yu, Z. Liu, and X. Bao. "New LP relaxations for minimum cycle/path/tree cover problems". In: *Theoretical Computer Science* 803 (2020), pp. 71–81.

[Yam+09] T. Yamamoto, H. Bannai, M. Nagasaki, and S. Miyano. "Better decomposition heuristics for the maximum-weight connected graph problem using betweenness centrality". In: *International Conference on Discovery Science*. Springer. 2009, pp. 465–472.

[Yan98] L. Yan. "On path decompositions of graphs". PhD thesis. Arizona State University, 1998.

[ZL06] M. Zhai and C. Lü. "Path decomposition of graphs with given path length". In: *Acta Mathematicae Applicatae Sinica* 22.4 (2006), pp. 633–638.

[ZS05] A. A. Zoltners and P. Sinha. "Sales territory design: Thirty years of modeling and implementation". In: *Marketing Science* 24.3 (2005), pp. 313–331.

[Zha16] C.-Q. Zhang. "Circuit Double Covers of Graphs". In: *Graph Theory*. Springer, 2016, pp. 273–291.

[Zha97] C.-Q. Zhang. *Integer Flows and Cycle Covers of Graphs*. Vol. 205. CRC Press, 1997.

[Zhe+05] X. Zheng, S. Jain, S. Koenig, and D. Kempe. "Multi-robot forest coverage". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2005, pp. 3852–3857.

[Zho+19] X. Zhou, H. Wang, B. Ding, T. Hu, and S. Shang. "Balanced connected task allocations for multi-robot systems: An exact flow-based integer program and an approximate tree-based genetic algorithm". In: *Expert Systems with Applications* 116 (2019), pp. 10–20.

Zusammenfassung

Das Problem, einen Graphen mit Untergraphen zu überdecken, ist ein grundlegendes Problem der Graphentheorie und kommt in verschiedenen Varianten in einer Reihe von praktischen Optimierungsproblemen vor. Sei es bei der Konzipierung von Vertriebsgebieten eines Unternehmens, von Schulbezirken oder Wahlkreisen, oder sei es der koordinierte Einsatz mehrerer Roboter zum Säubern, Ernten oder zur Sicherheitsüberwachung eines Gebiets: All diese Aufgaben beinhalten im Kern ein Überdeckungsproblem auf einem Graphen. Wie man an den Beispielen bereits erkennen kann, gibt es eine Reihe von weiteren Nebenbedingungen, die erfüllt sein müssen. Besonders hervorzuheben sind hier der (graphentheoretische) Zusammenhang eines Untergraphen und die Einhaltung mehr oder weniger enger Schranken an die "Größe" des Untergraphen, die je nach Anwendung zum Beispiel die Anzahl an Kunden, Schülern oder Wahlberechtigten ist. In vielen Fällen ist es auch vorgeschrieben, dass die Untergraphen disjunkt sind. In diesem Fall spricht man von einem Partitionierungsproblem.

Auch das Ziel der Überdeckung ist abhängig von der Anwendung. Wenn die Anzahl an Untergraphen für die Überdeckung vorgegeben ist, dann kann besipielsweise die Minimierung der durchschnittlichen Wegezeit zur Schule das Ziel sein, und anderenfalls könnte eine Minimierung der Anzahl der Untergraphen angestrebt werden. Dies ist dann im Beispiel von oben die Anzahl der Putzroboter, die benötigt wird, um das gegebene Gebiet in einer bestimmten Zeit zu säubern.

Ein Hauptziel dieser Dissertation ist es, einen Überblick über das gesamte Gebiet der Graphenüberdeckung (und -partitionierung) aus mathematischer Sicht zu geben. Wir behandeln unterschiedliche Varianten der Überdeckung und stellen jeweils die wichtigsten Fragestellungen, Methoden und Ergebnisse vor. An verschiedenen Stellen erweitern wir die bestehende Literatur durch eigene Ergebnisse zu Approximation oder neuen Formulierungen.

Das zweite Hauptziel ist das Lösen eines konkreten Optimierungsproblems aus einem Forschungsprojekt mit dem Bundesamt für Güterverkehr zur optimalen LKW-Mautkontrolle auf deutschen Autobahnen. Dabei muss das Autobahnnetz mit Kontrollgebieten überdeckt werden, auf denen mobile Kontrolleinheiten dann ihre Dienste verrichten. Wir modellieren das Problem als mathematisches Optimierungsproblem und untersuchen verschiedene Lösungsansätze. Besonders erfolgreich ist ein Ansatz, bei dem es eine Variable für jeden möglichen Untergraph gibt. Da dies zuviele Variablen (in Matrixdarstellung: Spalten) sind, verfolgen wir den Ansatz einer Spaltengenerierung (column generation). Der Unteraufgabe, einen weiteren geeigneten Untergraphen zu identifizieren, kommt dabei eine ganz besondere Bedeutung zu. Wir präsentieren verschiedene Algorithmen zur Vereinfachung der Problemstruktur, zur heuristischen Lösung des Problems sowie zum Finden unterschiedlicher Schnittebenen, die das Polyeder der zulässigen LP-Lösungen verkleinern, ohne dabei ganzzahlige Lösungen abzuschneiden. In ausführlichen Testrechnungen auf praxisrelevanten und künstlichen Instanzen zeigen wir, dass unsere Verbesserungen einen enormen Mehrwert erzeugen. Erfreulicherweise beschränkt sich dieser Mehrwert nicht auf das vorliegende Problem der Mautkontrolle, sondern wir können ihn auch auf andere Problemstellungen aus dem *districting*, also den oben beschriebenen Überdeckungsproblemen, übertragen.

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und alle dabei verwendeten Hilfsmittel und Quellen angegeben habe. Geistiges Eigentum anderer Autoren wurde als entsprechend gekennzeichnet. Ebenso versichere ich, dass ich an keiner anderen Stelle ein Prüfungsverfahren beantragt bzw. die Dissertation in dieser oder anderer Form an keiner anderen Fakultät als Dissertation vorgelegt habe.

Berlin, 03. Januar 2023,

Stephan Schwartz