Exact Algorithms for Network Design Problems using Graph Orientations

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Maria Kandyba-Chimani

Dortmund
2011

Tag der mündlichen Prüfung:
30.03.2011

Dekan:
Prof. Dr. Peter Buchholz

Gutachter:
Prof. Dr. Petra Mutzel, Fakultät für Informatik, TU Dortmund
Prof. Dr. Christoph Buchheim, Fakultät für Mathematik, TU Dortmund

Abstract

The subject of this thesis are exact solution strategies for topological network design problems. These combinatorial optimization problems arise in various real-world scenarios, as, e.g., in the telecommunication and energy industries. The prime task thereby is to plan or extend networks, physically connecting customers. In general we can describe such problems graph-theoretically as follows: Given a set of nodes (customers, street crossings, routers, etc.), a set of edges (potential connections, e.g., cables), and a cost function on the edges and/or nodes. We ask for a subset of nodes and edges, such that the sum of the costs of the selected elements is minimized while satisfying side-conditions as, e.g., connectivity, reliability, or cardinality. In this thesis we concentrate on two special classes of topological network design problems: the k-cardinality tree problem (KCT) and the $\{0,1,2\}$-survivable network design problem ($\{0,1,2\}$-SND) with node-connectivity constraints. These problems are in general NP-hard, i.e., according to the current knowledge, it is very unlikely that optimal solutions can be found efficiently (i.e., in polynomial time) for all possible problem instances.

The above problems can be formulated as integer linear programs (ILPs), i.e., as systems of linear inequalities, integral variables, and a linear objective function. Such models can be solved using methods of mathematical programming. Generally, the corresponding solutions methods can be very time-consuming. This was often used as an argument for developing (meta-)heuristics to obtain solutions fast, although at the cost of their optimality. However, in this thesis we show that, exploiting certain graph-theoretic properties of the feasible solutions, we are able to solve large real-world problem instances to provable optimality efficiently in practice. Based on orientation properties of optimal solutions we formulate new, provably stronger ILPs and solve them via specially tailored branch-and-cut algorithms. Our extensive polyhedral analyses show that these models give tighter descriptions of the solution spaces and also offer certain algorithmic advantages in practice. In the context of $\{0,1,2\}$-SND we are able to present the first orientation property of 2-node-connected graphs which leads to a provably stronger ILP formulation, thereby answering a long standing open research question. Until recently, both our problem classes allowed optimal solutions only for instances with roughly up to 200 nodes. Our experimental results show that our new approaches allow instances with thousands of nodes. Especially for the KCT problem, our exact method is often even faster than state-of-the-art metaheuristics, which usually do not find optimal solutions.

Zusammenfassung

Gegenstand dieser Dissertation sind exakte Lösungsverfahren für topologische Netzwerkdesignprobleme. Diese kombinatorischen Optimierungsprobleme tauchen in unterschiedlichen realen Anwendungen auf, wie z.B. in der Telekommunikation und der Energiewirtschaft. Die grundlegende Problemstellung dabei ist die Planung bzw. der Ausbau von Netzwerken, die Kunden durch physikalische Leitungen miteinander verbinden. Im Allgemeinen lassen sich solche Probleme graphentheoretisch wie folgt beschreiben: Gegeben eine Menge von Knoten (Kunden, Straßenkreuzungen, Router u.s.w.), eine Menge von Kanten (potenzielle Verbindungsmöglichkeiten) und eine Kostenfunktion auf den Kanten und/oder Knoten. Zu bestimmen ist eine Teilmenge von Knoten und Kanten, so dass die Kostensumme der gewählten Elemente minimiert wird und dabei Nebenbedingungen wie Zusammenhang, Ausfallsicherheit, Kardinalität o.ä. erfüllt werden. In dieser Dissertation behandeln wir zwei spezielle Klassen von topologischen Netzwerkdesignproblemen, nämlich das k-Cardinality Tree Problem (KCT) und das {0,1,2}-Survivable Netzwerkdesignproblem ({0,1,2}-SND) mit Knotenzusammenhang. Diese Probleme sind im Allgemeinen NP-schwer, d.h. nach derzeitigem Stand der Forschung kann es für solche Probleme keine Algorithmen geben die eine optimale Lösung berechnen und dabei für jede mögliche Instanz eine effiziente (d.h. polynomielle) Laufzeit garantieren.

Die oben genannten Probleme lassen sich als ganzzahlige lineare Programme (ILPs) formulieren, d.h. als Systeme aus linearen Ungleichungen, ganzzahligen Variablen und einer linearen Zielfunktion. Solche Modelle lassen sich mit Methoden der sogenannten mathematischen Programmierung lösen. Das die entsprechenden Lösungsverfahren im Allgemeinen sehr zeitaufwendig sein können, war ein oft genutzes Argument für die Entwicklung von (Meta-)Heuristiken um schnell eine Lösung zu erhalten, wenn auch auf Kosten der Optimalität. In dieser Dissertation zeigen wir, dass es, unter Ausnutzung gewisser graphentheoretischer Eigenschaften der zulässigen Lösungen, durchaus möglich ist große anwendungsnahe Probleminstanzen der von uns betrachteten Probleme beweisbar optimal und praktisch-effizient zu lösen. Basierend auf Orientierungseigenschaften der optimalen Lösungen, formulieren wir neue, beweisbar stärkere ILPs und lösen diese anschließend mit Hilfe maßgeschneiderter Branch-and-Cut Algorithmen. Durch umfangreiche polyedrische Analysen können wir beweisen, dass diese Modelle einerseits formal stärkere Beschreibungen der Lösungsräume liefern als bisher bekannte Modelle und andererseits für Branch-and-Cut Verfahren viele praktische Vorteile besitzen. Im Kontext des {0,1,2}-SND geben wir zum ersten Mal eine Orientierungseigenschaft zweiknotenzusammenhängender Graphen an, die zu einer beweisbar stärkeren ILP-Formulierung führt und lösen damit ein in der Literatur seit langem offenes Problem. Unsere experimentellen Ergebnisse für beide Problemklassen zeigen, dass—während noch vor kurzem nur Instanzen mit weniger als 200 Knoten in annehmbarer Zeit berechnet werden konnten—unsere Algorithmen das optimale Lösen von Instanzen mit mehreren tausend Knoten erlauben. Insbesondere für das KCT Problem ist unser exaktes Verfahren oft sogar schneller als moderne Metaheuristiken, die i.d.R. keine optimale Lösungen finden.

Acknowledgements

First of all, I want to express my deep gratitude to my advisor Prof. Dr. Petra Mutzel for opening me the way into the scientific world, for leading me into the field of network design, guiding me through my PhD years, and introducing me to Ivana Ljubic and Mechthild Stoer, whose PhD work was the gateway into my research topics. Also due to Petra, I had the freedom to find my favorite research direction and to change the originally proposed thesis topic. From the very beginning, Petra not only shared her insight in the fields of graph theory, algorithm engineering and mathematical programming, but encouraged and supported me to visit helpful international workshops and conferences. The latter taught me various subject-specific skills and allowed me to meet researchers from all over the world and establish fruitful cooperations. Also, thank you, Petra, for supervising me over long distances, thus allowing me to be flexible enough to combine research and family and to live and work in Brno, Vienna, and Jena.

My special thanks are directed to my frequent co-authors and friends Ivana Ljubic and Markus Chimani for being personally involved in my PhD work and for innumerable useful advices and discussions. Thank you for the fruitful cooperations, for being a pleasant company at conferences and workshops, and also for proofreading my thesis. You both taught me that research may cause a lot of joy. Thank you, Ivana, for providing me with the initial implementation of your branch-and-cut algorithm for the prize-collecting Steiner tree problem. Moreover, I want to thank S. Raghavan and Mechthild Stoer for very useful discussions on the topic of survivable network design.

I want to thank the German Research Foundation and the Deutsche Telekom Stiftung for the financial support of my PhD studies. In particular, many thanks to the Deutsche Telekom Stiftung for prolonging my grant and for giving me more freedom due to the birth of my son Jonathan.

I want to thank my colleagues from LS XI for having a good and productive time at TU Dortmund, for the relaxing coffee breaks, and for an overall pleasant working environment. Many thanks to all my friends for providing me comfortable leisure time, especially Assja, Xenia, Elena and Galina for supporting me in all life situations.

Many special thanks go to my family: I would like to thank my parents, whom I owe all my ambitions and thirst for knowledge, for their absolute support, and for offering me innumerable life possibilities. I furthermore want to express my gratitude to my parents in law for their unconditional involvement and help with Jonathan which allowed me to overcome my permanent lack of sleep and allowing me to have many precious working hours. Last but not least, I want to thank Markus, my beloved husband and co-author, for not ceasing to motivate me, cheering me up, for his support, inspiration, and endless fruitful discussions. You know, my gratitude cannot be expressed in simple words!

Contents

Part I

Prelude

Chapter 1

Introduction

In this thesis we discuss *topological network design* problems and present *provably optimal* algorithms for them that are based on certain *orientation properties* of the corresponding feasible solutions. In the next sections we will introduce these three main ingredients and close with a brief overview on this thesis.

1.1 Topological Network Design

Providing communication services like telephone and internet connections requires a huge planning effort where minimizing the arising costs is one of the biggest issues. The main cable material used for internet and telephone connections in the last century was copper. However, using modern fiber-optic technology allows much higher and faster data throughput. Indeed, it is essential for delivering high-speed internet to customers. Therefore, large European telecom companies decided to use fiber-optic cables not only for their backbone networks, but also for new customer connections and in fact to completely replace the old copper networks over time. Since the process of cable laying is very costly, deciding where to lay them justifies a thorough optimization.

The above setting is a prominent motivation for *topological network design* which constitutes the main subject of this thesis: Consider a set of potential customers, facilities and further connection points, and a set of possible route segments (e.g., streets) together with the, e.g., cable laying costs for each of them. In order to design a network, a subset of the route segments has to be chosen connecting the given customers with each other or to some service facilities. Thereby we want to minimize the overall cost. As we will discuss in the following, various problem scenarios may occur depending on certain economical or even political side-conditions. Apart from natural *connectivity constraints*, further restrictions may be imposed on the required network. In this thesis, we in particular deal with *survivability*, *cardinality* and *prize-collecting* constraints. Figures 1.1 and 1.2 illustrate examples of different topological network design settings and their possible feasible solutions, see also Section 1.4 for details.

One possible aspect of topological network design is *survivability*: Some customers may be more profitable or important than others, and require certain ser-

vice guarantees. In the case of cable damages or failures of some service facilities, a number of alternative connections has to be provided. Indeed, guaranteeing such redundancy becomes particularly crucial when designing fiber-optic networks: Not incorporating survivability in the corresponding design model results in networks that contain a substantially smaller number of connection links than current copper networks. This is due to the higher capacity of fiber optic cables. Together with the fact that each fiber link carries a larger amount of the traffic than its copper counterpart, the mentioned sparsity of a fiber-optic network may cause more severe service interruptions (and losses of revenue) even when only a single connection point or link is damaged. One of the most prominent examples is a severe undersea cable damage through a ship anchor that happened 2008 near the Egyptian coast. Thereby more than 75% of traffic between the Middle East, Europe, and America has been disrupted[1]. The following revealing comment on this incident underlining the importance of a thorough cost optimization was given by Todd Underwood, vice-president of the information analysis company Renesys, to the online magazine Wired[2]:

> "Given the desire by telecoms and broadband customers to keep costs low, situations like the current cuts will continue to happen. [...] This is all about money – how much money do we want to pay to make sure the network doesn't go down?"

Our second important set of requirements for topological network design are *cardinality constraints*, i.e., constraints on the number of the route segments or connection points that are contained in the solution network. The following real-world problem occurred in the 1990s in the Norwegian oil-industry: The government leased oil-fields to companies allowing them to explore these fields. After six years, each company had to return a connected part of its territory which had to contain at least half of the original oil-field. Naturally, the essential task for a company was to find the most unprofitable such part. As we discuss later in this thesis, this problem can be seen as a topological network design problem with cardinality constraints. Such constraints, e.g., also arise when a telecommunication company is permitted to service only a certain number of possible customers due to monopoly regulations.

Furthermore, a telecommunication company is often allowed to decide whether to include a customer into a network or not. This decision has to be made by comparing the estimated profit of a customer to the cost of connecting him/her to the network. On the other hand, there may also be customers (e.g., some governmental institutions), which have to be provided with service at any cost. When the customer's profit is taken into account (at least for some customers), we are dealing with *prize-collecting* topological network design problems.

All above mentioned problems can be mathematically formulated by modeling them on graphs: customers, facilities and further connection points are represented

[1] http://www.computerweekly.com/Articles/2009/01/06/234016/Cable-damage-disrupts-internet-services.htm

[2] http://www.wired.com/threatlevel/2008/01/fiber-optic-cab/

as nodes and the route segments are represented as edges. The resulting graph problems belong to the class of combinatorial optimization problems, as the necessary decisions are of discrete nature. Prominent representatives of topological network design problems in the literature are, e.g., the well-known minimum spanning tree (MST), the Steiner tree and Steiner network problems as well as their extensions. Whereas MST is polynomially solvable, already the Steiner tree problem is *NP-hard*, i.e., it is assumed that there is no algorithm that can solve such a problem in polynomial worst-case time.

In this thesis, we focus on two different classes of NP-hard topological network design problems: *k-cardinality tree* and *{0,1,2}-survivable network design* (also incorporating the prize-collecting aspect), see also Section 1.4 for a more detailed description.

1.2 Exact Algorithms

The main goal of this thesis is to develop practically relevant exact algorithms, i.e., algorithms computing provably optimal solutions for large real-world instances of the above mentioned problems. Therefore, we model the given problems as (mixed) integer linear programs (ILPs) and solve these using methods of mathematical programming.

Usually, algorithms for solving ILPs have a worst-case running time that is exponential in the size of the input. Hence, naive ILP algorithms often fail in practice when applied to real-world scenarios. For a long time, this has been a main argument by different research communities to waive provable optimality and use promising heuristic or meta-heuristic algorithms instead.

However, exploiting the structure of the given problem in order to get provably strong ILP models and using sophisticated algorithms tailored to these and to the typical special properties of real-world instances, can be surprisingly successful and lead to quick optimal solutions. Several recent results [PD01c,LWP$^+$06] confirm this and show that strong ILP models, when used in conjunction with recent advances in CPU power and ILP solvers, can allow to solve large real-world instances for prominent network design problems, such as the Steiner tree and prize-collecting Steiner tree, to provable optimality within reasonable time bounds. Hence, our first step is to design such strong ILP models for the above mentioned problem classes. Fortunately, it turns out that these problems have an important common property: the corresponding optimal solutions can be characterized by the means of *graph orientations*. We therefore use such characterizations to derive ILPs that are stronger then the previously known models.

When designing our ILP models and solution methods, we furthermore follow the principles of Algorithm Engineering, e.g., by exploiting the knowledge about the given underlying graphs. They may have special properties like low or high density, specially structured cost functions (e.g., satisfying the triangle inequalities) or may be similar to grids. We can design special purpose improvements tuned for such instances to further enhance the performance of our exact algorithms, making them highly competitive in practice.

1.3 The Role of Graph Orientations

All our problems and most of the corresponding known ILP models are based on undirected graphs. Usually, similar problems that are defined on arbitrary directed graphs tend to be harder than their undirected counterparts. Yet, for network design problems such as, e.g., the Steiner tree problem, it can be advantageous to transform the undirected problem into a directed one on a related bidirected graph, as the LP relaxation of the resulting ILP may deliver stronger bounds than their undirected counterpart.

Unfortunately, not all undirected topological network design problems have an equivalent directed counterpart. This is only possible for problems where all optimal solutions have certain common orientation properties that can be exploited. For our problems, we can show that this is indeed the case. Thus the orientability issue becomes a central focus of this thesis.

Whereas the orientation property of trees that we use to solve the k-cardinality tree problem is well-known and has already been used in order to solve similar problems (where feasible solutions represent trees), the necessary orientation properties of solutions to the considered survivable network design problems were a long-time open question in the literature. We answer this question by describing such orientation properties and proving their feasibility.

1.4 Thesis Overview

This thesis consists of four parts. Part I is a basic introduction to our topics, graph theory, and mathematical programming. Since this thesis deals with two different types of topological network design problems, their discussion is divided into two separate parts: Part II considers the k-cardinality tree problems whereas Part III covers our results for $\{0,1,2\}$-survivable network design. We then conclude with Part IV summarizing our results and giving an outlook on possible future research on topological network design.

As the name already suggests, the k-cardinality tree problems consider the cardinality aspect of topological network optimization. The task is to find a cost-minimum connected, cycle-free subgraph of a given graph that contains exactly k edges. Figure 1.1 shows a small example of this problem in the context of oil-platforms.

The $\{0,1,2\}$-survivable network design problems focus on incorporating survivability and prize-collecting constraints into the network optimization. The task is to find a minimum cost connected subgraph of a given graph such that for each node the subgraph satisfies a given connectivity requirement modeled by integer numbers 0, 1, 2: We call nodes with requirement 2 the *important customers*, the ones with requirement 1 the *regular customers*. All customers of these two types have to lie in a connected solution network. For additional solution properties, we distinguish between different interpretations of connectivity requirement "2": A *survivable network w.r.t. edge-connectivity* requires each pair of important customers to be connected to each other via two edge-disjoint paths, i.e., paths that do not have any connection link in common. A survivable network w.r.t. *root-node-*

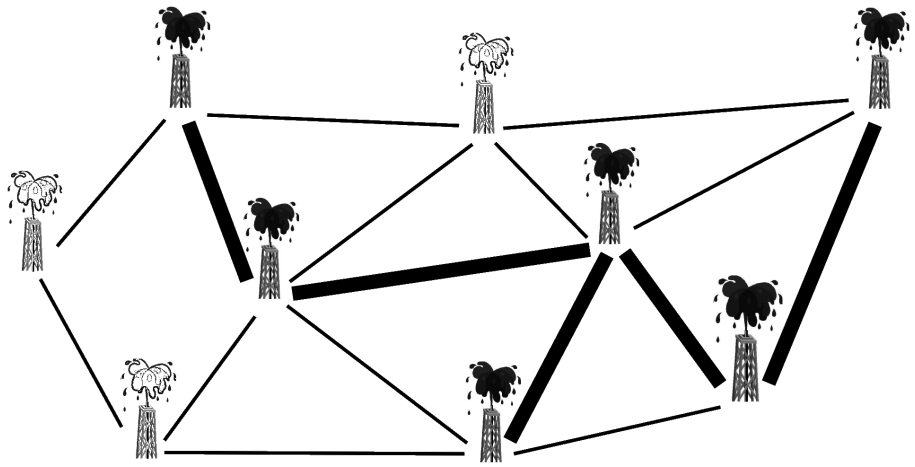


Figure 1.1: Example of a k-cardinality tree problem. Bold edges denote a solution for $k = 5$.

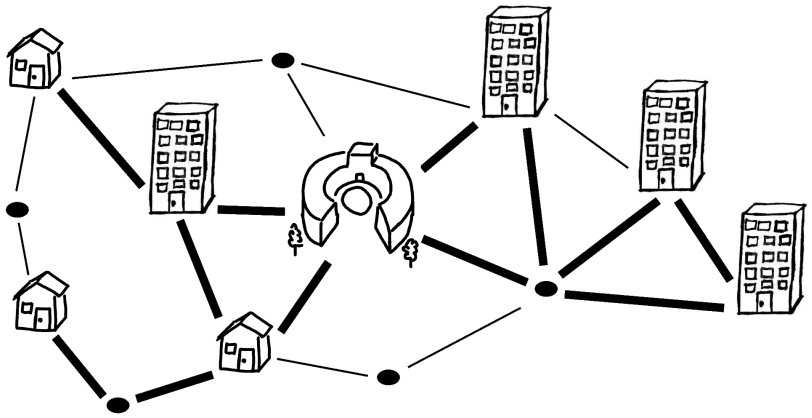
connectivity connects each important customer to a predefined hub (called *root*) via two node-disjoint paths, i.e., paths that do not share any connection point. This root usually represents a preexisting infrastructure, a company headquarter, etc. Finally, a survivable network w.r.t. *node-connectivity* contains two node-disjoint paths for each pair of important customers. Figure 1.2 shows examples of survivable networks that are feasible w.r.t. the above interpretations. In this thesis we will in particular deal with the latter two node-disjoint variants. Furthermore, we will also consider the above problem types in their prize-collecting variant.

As both the k-cardinality tree and the survivable network design problems are well-known in the literature, we provide an extensive overview on the previously published results. We thereby consider complexity results as well as known heuristics, approximations, and exact algorithms. For each problem, we in particular list all previously known ILP models. As the practical relevance of our algorithms is a central aspect of this thesis, we give an overview on all previously published computational studies and the thereby used benchmark instances. In particular on the vast field of survivable network design, multiple surveys have been published before. However, since various new results appeared only recently, most of these surveys are not complete. To our knowledge, the survey given in this thesis, is the currently most complete one including results of the most current research regarding heuristic, approximation, and exact algorithms.

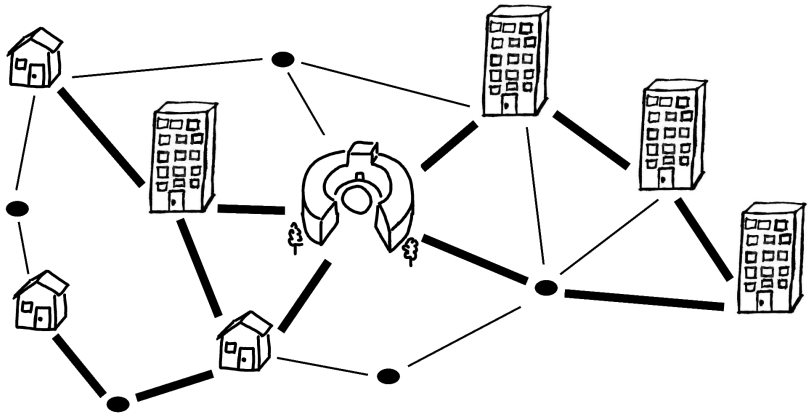
For both problem classes, we develop fast, practically relevant exact algorithms based on a branch-and-cut scheme. Thereby we use the following strategy:

(a) We analyze the structure of the optimal solutions and provide graph-theoretical foundations in order to construct an equivalent problem based on directed graphs.

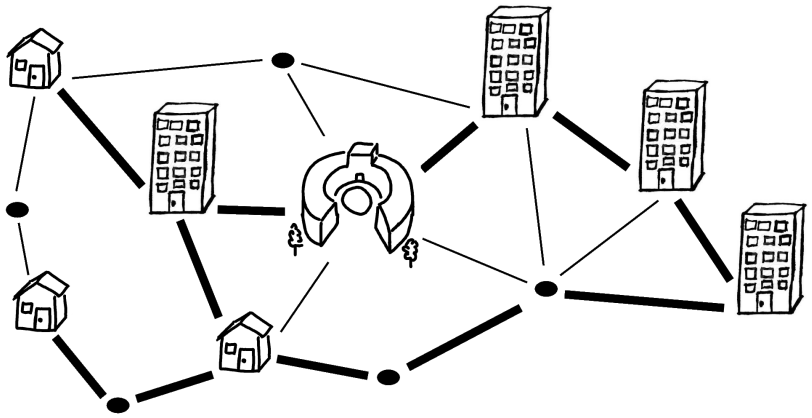
(b) We formulate different ILPs for the directed problem and choose the adequate model for our algorithm. This choice is based on extensive polyhedral



(a) Survivable network with respect to edge-connectivity



(b) Survivable network with respect to root-node-connectivity



(c) Survivable network with respect to node-connectivity

Figure 1.2: Examples of our {0,1,2}-survivable network design problems. Small houses and high-rise buildings denote regular and important customers, respectively. The architecturally sophisticated round building represents a central hub that, depending on the problem type, serves either as the root or plays the role of a high-rise building. Black dots denote nodes with connectivity requirement 0 (e.g., street crossings). Bold edges denote a possible solution.

comparisons of our ILPs with the ones previously used in the literature.

(e) We design a special purpose exact algorithm using a branch-and-cut framework and provide a corresponding computational analysis on both the known and on new benchmark instances.

Although the theory and methods we apply in order to solve the k-cardinality problem have already been successfully applied to other problems like the *Steiner tree* and *prize-collecting Steiner tree* problems, this is the first exact algorithm for the k-cardinality tree problem that is able to deal with large-scale real-world instances (with up to roughly 2500 nodes) and random instances with up to 5000 nodes. We extensively tested it on the known benchmark library KCTLIB that was used to test the previous, mainly metaheuristic, approaches for this problem. Our approach solves these instances to *provable* optimality not only in reasonable time bounds, but (for groups with up to 1000 nodes) is also faster than the best previously known metaheuristics. Our experiments therefore serve as a counterexample to the previously mentioned argument for the necessity of metaheuristics purely based on the fact that a problem is NP-hard. We show that this argument is deceptive on this and related problems at least for the problems of the known benchmark instances.

For our {0,1,2}-survivable network design problems with the node-disjoint character, we derive novel orientation properties and use them to derive strong ILP models and design competitive branch-and-cut algorithms based on them. Although most of our survivability problems have been already treated in the literature by other communities, until now there was no common challenging benchmark set that could have been used for comparative computational studies. We therefore propose such a benchmark set and use it for our experiments. We give an extensive experimental study on the performance of our algorithms thus showing their effectiveness in practice.

This thesis is largely based on the following original papers:

- Obtaining Optimal k-Cardinality Trees Fast; at ALENEX'08 [CKLM08a]
- Obtaining Optimal k-Cardinality Trees Fast; in Journal on Experimental Algorithms [CKLM09]
- A New ILP Formulation for 2-Root-Connected Prize-Collecting Steiner Networks; at ESA'07 [CKM07]
- Strong Formulations for 2-Node-Connected Steiner Network Problems; at COCOA'08 [CKLM08b]
- Orientation-based Models for {0,1,2}-Survivable Network Design: Theory and Practice; in Mathematical Programming B [CKLM10]

Note that the experimental studies for both problem classes were largely extended (and recomputed) compared to the previously published results.

Although not included in this thesis, ideas presented herein formed the basis of further research by the author:

- Hybrid Numerical Optimization for Combinatorial Network Problems; at HM'07 [CKP07]
- 2-InterConnected Facility Location: Specifications, Complexity Results, and Exact Solutions; technical report, submitted to a journal [CKM09]

The first publication describes a hybrid version of the heuristic algorithm for {0,1,2}-survivable network design presented in this thesis. The second paper demonstrates how the results of this thesis can be successfully applied to facility location problems.

Chapter 2

Preliminaries

2.1 Graphs

The problems considered in this thesis are defined on graphs. In this chapter we will provide basic definitions for graph structures and emphasize their main properties that will be used later. This section is based on [Har69]. Omitted proofs can be found therein.

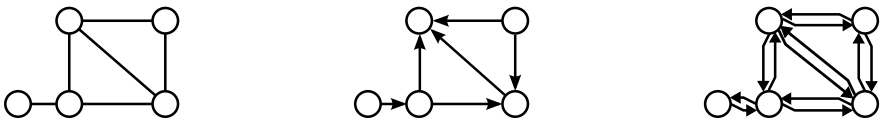
Graphs. An *undirected graph* G is a tuple (V, E), where V is a set of *nodes* and $E \subseteq \binom{V}{2} := \{\{u, v\} \mid u \neq v \in V\}$ is a set of edges, i.e., unordered pairs of nodes. Given an arbitrary graph G', we may use the functions $V(G')$ and $E(G')$ to obtain its nodes and edges, respectively. For an edge $e = \{u, v\}$, u and v are called *end nodes* of e. Furthermore, e is said to be *incident* with u and v, nodes u and v are said to be *adjacent* to each other. The number of edges incident to v is the *degree* of v. Edges sharing a common end node are *adjacent edges*.

Sometimes it can be useful to consider graphs with multiple edges between two nodes and to allow loops, i.e., edges with non-distinct end nodes. In this context our definition describes *simple* graphs. Unless mentioned otherwise our graphs will always be simple.

A *directed graph* $D = (V, A)$ features *arcs*, i.e., ordered node pairs $A \subseteq V \times V$, instead of edges. For an arc (u, v), u is called its *start node* and v its *end node*. The number of arcs with u as their start node (end node) is the *out-degree (in-degree)* of u.

A graph in which all nodes are adjacent is *complete*. A complete graph on three nodes is called a *triangle (graph)*. An $n \times m$*-grid* (also known as *lattice*) is a graph whose nodes correspond to the points (x, y) in the plane with integer coordinates $x = 1, \ldots, n$ and $y = 1, \ldots, m$ and two nodes are connected whenever the corresponding points have distance 1. An undirected graph is k*-regular* if every node has degree k.

In the following we will always assume $G = (V, E)$ and $D = (V, A)$ to be an undirected and a directed graph, respectively.



(a) G (b) $\hat{G}$ (c) $\bar{G}$

Figure 2.1: Example for an undirected graph G, one of its possible orientations $\hat{G}$ and its bidirection $\bar{G}$. G is also a shadow of $\hat{G}$

.

Cuts and Flows. Let $S \subseteq V$ be a subset of nodes. We denote by $\delta_G(S) := \{\{u, v\} \in E \mid u \in S, v \notin S\}$ the (*undirected*) *cut* or *cut edges* of S, i.e., the set of edges that separate S from the rest of the graph. We call S the *cut set* of $\delta_G(S)$. Furthermore, $\delta(S)$ is an s,t-*cut* if it separates nodes $s \in S$ and $t \in V \setminus S$. Observe that the degree of a node v can be written as the cardinality of the cut $\delta_G(v) := \delta_G(\{v\})$.

When we are dealing with a directed graph $D = (V, A)$ and $S \subseteq V$, the *directed cuts* $\delta^+_D(S) = \{(u, v) \in A \mid u \in S, v \notin S\}$ and $\delta^-_D(S) = \{(u, v) \in A \mid u \notin S, v \in V\}$ denote the set of out-going and in-coming arcs in S, respectively. Again, $|\delta^+_D(v)|$ and $|\delta^-_D(v)|$ denotes the *out-* and the *in-degree* of v. In the following, we may omit the subscripts specifying the graph, if it is clear from the context.

A function $f : A \to \mathbb{R}^+$ is an s,t-*flow* for some nodes $s \neq t \in V$ if the capacity constraints $f(a) \leq c(a)$ are satisfied for all arcs $a \in A$, and for all nodes $v \in V$ the *flow-conservation constraints*

$$\sum_{a \in \delta^-(v)} f(a) - \sum_{a \in \delta^+(v)} f(a) = \begin{cases} -b < 0 & \text{if } v = s, \\ b > 0 & \text{if } v = t, \\ 0 & \text{else} \end{cases}$$

hold. The nodes s and t are the *source* and *sink* of this flow, respectively. We denote the *amount* of flow by $|f| = b$. The maximum amount over all s,t-flows is the *maximum flow* in D. Flows in undirected graphs are defined analogously.

Orientations. A directed graph is called *oriented* if there are no symmetric pairs of arcs, i.e., it does not contain pairs of arcs (u, v) and (v, u). An *orientation* of an undirected graph $G = (V, E)$ is a directed graph $\hat{G} = (V, A)$ obtained by assigning a unique direction to each edge of G, so that $\hat{G}$ is oriented. Formally, we have $|A \cap \{(u, v), (v, u)\}| = 1$ for all $\{u, v\} \in E$. Inversely, a *shadow* (or *underlying undirected graph*) of a directed graph $D = (V, A)$ is the undirected graph $G_D = (V, E)$ where $\{u, v\} \in E$ if and only if (u, v) or (v, u) is in A. During the thesis we will often use a related concept: A *bidirection* of an undirected graph $G = (V, E)$ is the graph $\bar{G} = (V, A)$ where each edge is replaced by its two possible arcs, i.e., $A = \{(u, v), (v, u) \mid \{u, v\} \in E\}$. See Figures 2.1(a) and 2.1(b), 2.1(c) for examples of these concepts.

The following definitions are given for undirected graphs. If not specified otherwise they can be extended to directed graphs straightforwardly.

Graph structures. A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$, if $V' \subseteq V$ and $E' \subseteq E$. A subgraph G' is *induced* by V' if $E' = E \cap \binom{V'}{2}$. In the latter case we may use the notations $G[V']$ and $E(V')$ for G' and E', respectively.

A *walk* $P = [v_0 \to v_\ell]$ of length $\ell \geq 0$ in an undirected graph G is a subgraph $P = (V_P, E_P)$ of G with $V_P = \{v_i \in V \mid i \in \{0, ..., \ell\}\}$ and $E_P = \{\{v_i, v_{i+1}\} \in E \mid i \in \{0, \dots, \ell - 1\}\}$. We call v_0 and v_ℓ the *end nodes* and v_i ($i \in 1, \dots, \ell - 1$) the *inner nodes* of the walk. In the context of directed graphs we analogously talk about directed walks $P = (v_0 \to v_\ell)$ with arcs $A_P = \{(v_i, v_i + 1) \in A \mid i \in \{0, \dots, \ell - 1\}\}$.

A walk is a *path* if $v_i \neq v_j$ for $i \neq j$. If all inner nodes of a path have degree 2, we call it *line*. A walk is *closed* if $v_0 = v_\ell$. A *cycle* is a closed walk where all nodes are pairwise disjoint except for the end nodes.

Let $u \neq v$ be two distinct nodes in a graph G. A collection of ℓ paths $P_i = [u \to v]$ with $\ell \geq 2$ in G are *edge-disjoint* if they do not share any edge. They are *node-disjoint* if all their inner nodes are pairwise disjoint. Clearly, if all paths are node-disjoint, they are also edge-disjoint but not vice versa.

Connectivity and trees. A graph $G = (V, E)$ is *connected* if there exists a path $[u \to v]$ for every pair of nodes $u, v \in V$. A *connected component* of a given graph is a maximal connected subgraph. In general, a graph may consist of several (disjoint) connected components. For directed graphs $D = (V, A)$ we can distinguish two kinds of connectivity: D is *weakly connected* if its shadow is connected. It is *strongly connected* if there exists a directed path from u to v for all pairs of nodes $u, v \in V$. A *strongly connected component* is a maximal strongly connected subgraph of D.

A *tree* $T = (V, E)$ is a connected graph that does not contain any cycle. A subtree of a graph G is a subgraph of G that forms a tree. A *leaf* is a node of a tree with degree 1. If all connected components of a graph G form trees, G is a *forest*. A directed graph $D = (V, A)$ is called *arborescence*, if its shadow is a tree and there exists a special *root node* $r \in V$ such that there exists a directed path $(r \to u)$ for every node $u \in V$.

2-Connectedness. A *bridge* in G is an edge $e \in E$ such that the removal of e disconnects G. More precisely, the graph $G \setminus \{e\} := (V, E \setminus \{e\})$ has more connected components than G. A connected graph on at least 2 nodes is *2-edge-connected* if it does not contain any bridge.

Analogously, a *cut node* (or *articulation node*) in G is a node $v \in V$ such that the removal of v and all its incident edges disconnects G. A connected graph on at least 3 nodes is *2-node-connected* if it does not contain any cut node, i.e., the removal of any node would still leave a connected graph. Note that if a given graph is 2-node-connected, it is also 2-edge-connected but not vice versa.

Analogously to connected components we can define *2-edge-connected* and *2-node-connected components* of a given graph, i.e., its maximal 2-edge-connected or 2-node-connected subgraphs, respectively. In slight contrast to a 2-node-connected

component, a *block* of G is any maximal subgraph without any cut node, i.e., a block is either a 2-node-connected component or a bridge including its endpoints (i.e., an edge connecting two cut nodes).

The following theorems characterize bridges, cut nodes and 2-node-connected graphs, respectively, and give a deeper intuition of these graph structures.

Theorem 2.1. *Let $e \in E$ be an edge of a graph $G = (V, E)$. The following statements are equivalent:*

1. *e is a bridge in G.*
2. *e is not on any cycle of G.*
3. *There exist nodes $u, v \in V$ such that e is on every $[u \to v]$-path in G.*
4. *There exists a bipartition of V into subsets U and W such that for any nodes $u \in U$ and $w \in W$, e is on every $[u \to w]$-path in G.*

Theorem 2.2. *Let v be a node of a connected graph G. The following statements are equivalent:*

1. *v is a cut node of G.*
2. *There exist nodes $u \neq w \in V$, $u, w \neq v$, such that v is on every $[u \to w]$ path.*
3. *There exists a bipartition of $V \setminus \{v\}$ into subsets U and W such that for any nodes $u \in U$ and $w \in W$, the node v is on every $[u \to w]$ path.*

Theorem 2.3. *Let $G = (V, E)$ be a connected graph with $|V| \geq 3$. The following statements are equivalent:*

1. *G is 2-node-connected.*
2. *Every pair of nodes $u, v \in V$ lie on a common cycle.*
3. *Let $u \in V$ and $e \in E$ be any node and edge in G. Then u and e lie on a common cycle.*

Let a graph H and a walk $P = [u \to v]$ be subgraphs of G such that $V(H) \cap V(P) = \{u, v\}$, whereby u and v are the end nodes of P. If P is a path or a cycle, it is called an *ear*. If P is a path it is called an *open ear*. We say a graph G has an *(open) ear decomposition* with a start edge $\{s, t\}$ if it can be constructed from $\{s, t\}$ by successively adding (open) ears. Thereby, the first added ear has to be an open ear $[s \to t]$. The start edge together with an open ear $[s \to t]$ is the smallest (open) ear decomposable graph. The following theorem characterizes 2-edge- and 2-node-connected graphs by the means of ear decompositions:

Theorem 2.4. *A graph G is 2-edge-connected if and only if it has an ear decomposition. G is 2-node-connected if and only if it has an open ear decomposition.*

k-Connectedness. The notion of 2-connectedness can be naturally extended to more general k-connectedness.

The *node-connectivity number* (also known as *connectivity number*) $\kappa(G)$ of a graph G is the minimum number of nodes whose removal results in a disconnected graph G or a trivial graph consisting of one node. Analogously, the *edge-connectivity number* $\alpha(G)$ is the minimum number of edges whose removal results in a disconnected graph. Clearly we have $\kappa(G) \leq \alpha(G)$. A graph G is *k-node-connected* if $k \leq \kappa(G)$ and it is k-edge-connected if $k \leq \alpha(G)$.

The node aspect of the following theorem was originally proposed by Menger in 1927 [Men27] and was later extended by its edge variant, e.g., in [FF56, EFS56].

Theorem 2.5 (Menger's Theorem)**.** *The minimum number of edges (nodes) separating two nonadjacent nodes s and t in G is the maximum number of edge-disjoint (node-disjoint, respectively) $[s \to t]$-paths in G.*

The generalization of the above theorem is the well-known max-flow min-cut theorem:

Theorem 2.6 (max-flow min-cut)**.** *Given a graph $G = (V, E)$, two nonadjacent nodes s and t and a capacity vector $c \in \mathbb{R}_+^{|E|}$. The maximum amount over all s,t-flows equals the minimum over all* cut capacities $c(\delta(S)) := \sum_{e \in \delta(S)} c(e)$ *with $S \subset V$, $s \in S$, and $t \in V \setminus S$.*

2.2 Orientation Theorems

A large part of this thesis relies on characterizations of 2-edge- and 2-node-connected graphs by means of feasible orientations. In 1939, Robbins showed the following theorem:

Theorem 2.7 ([Rob39])**.** *A graph $G = (V, E)$ is 2-edge-connected if and only if there exists an orientation $\hat{G}$ of G such that $\hat{G}$ is strongly connected, i.e., for every pair of nodes $u, v \in V$ there exist a directed $(u \to v)$-path in $\hat{G}$.*

Proof. **$\Rightarrow$:** Let $\hat{G}$ be a strongly connected orientation of G. For any subset $S \subset V$ and nodes $u \in S, v \in V \setminus S$, $\hat{G}$ contains at least one $(u \to v)$- and at least one $(v \to u)$-path. This implies $|\delta^+_{\hat{G}}(S)| \geq 1$ and $|\delta^-_{\hat{G}}(S)| \geq 1$, leading to $\delta_G(S) \geq 2$.

$\Leftarrow$: Let G be 2-edge-connected. Hence it has an ear decomposition (cf. Theorem 2.4). We use induction on the number of ears. As a base case, assume G is a cycle. Orienting this cycle such that we obtain a directed cycle gives us a strong connected orientation of G. As the induction step consider the decomposition of G into a subgraph G' with a strongly connected orientation $\hat{G}'$ and an ear $P = [a \to b]$. We obtain an orientation $\hat{P}$ by orienting P as a directed $(a \to b)$-walk. We now show that both orientations $\hat{G}'$ and $\hat{P}$ build a strongly connected orientation $\hat{G}$ of G.

Let $u \neq v \in V$ be some nodes in G. If $u, v \in V(G')$, there is a $(u \to v)$-path in the strongly connected $\hat{G}'$ by induction, and therefore also in $\hat{G}$. If $u \in V(P)$

and $v \in V(G')$, concatenating the $(u \to b)$-path in $\hat{P}$ and a $(b \to v)$-path in $\hat{G}'$ gives us a $(u \to v)$-path in $\hat{G}$. Analogously, we can concatenate a path $(u \to a)$ in $\hat{G}'$ with $(a \to v)$ in $\hat{P}$ if $u \in V(G')$ and $v \in V(P)$. If $u, v \in V(P)$, there is either a $(u \to v)$-path in $\hat{P}$ or we can concatenate $(u \to b)$ in $\hat{P}$, a $(b \to a)$-path in $\hat{G}'$, and $(a \to v)$ in $\hat{P}$ to a $(u \to v)$-path in $\hat{G}$. □

In general, it is not possible to straightforwardly generalize this proof to 2-node-connectivity. When simply replacing 2-edge-connectivity with 2-node-connectivity in the above theorem, the first part of the proof fails as the edges $\delta_G(S)$ may all be incident to a common node. Extending the orientability to additionally requiring node-disjointness for all pairs of paths $(u \to v)$ and $(v \to u)$ fails in the proof's second part: the concatenation with paths in $\hat{G}'$ does not allow to deduce node-disjointness, cf. Section 9.4.2 for a counterexample.

A feasible characterization of 2-node-connected graphs can be done by using the concepts of s,t-ordering and bipolar orientations. These concepts were first introduced by [LEC67]. Let $G = (V, E)$ be a 2-node-connected graph and $s \neq t \in V$. An ordering $s = v_1, v_2, \ldots, v_n = t$ on the nodes of G is called an *s,t-ordering*, if for all nodes v_j, $1 \leq j \leq n$, there exist $1 \leq i \leq j \leq k \leq n$ such that $\{v_i, v_j\}, \{v_j, v_k\} \in E$. An *$s,t$-orientation* (or *bipolar orientation*) of a graph G is an orientation such that the resulting directed graph is acyclic, and s and t are the only source and sink nodes, respectively.

Lemma 2.8. *A graph $G = (V, E)$ has an s,t-orientation if and only if it has an s,t-ordering. These can be transformed into each other in linear time.*

Proof. An s,t-ordering can be obtained from an s,t-orientation by topological ordering. An s,t-orientation can be obtained from an s,t-ordering by orienting the edges from smaller to higher ordering index. □

Theorem 2.9. *A graph $G = (V, E)$ is 2-node-connected if and only if it has an s,t-orientation w.r.t. any edge $\{s,t\} \in E$.*

Proof. Let $\{s,t\}$ be any edge in G. Since G is 2-node-connected, G has an open ear decomposition $P_0, P_1, \ldots, P_r$ starting with $\{s,t\}$. We use a labeling function $L : V \to [0,1] \cup \{\infty\}$ and call a node $v \in V$ labeled if $L(v) < \infty$. Initially we set $L(s) := 0$ and $L(t) := 1$ and $L(v) := \infty$ for all $v \in V \setminus \{s,t\}$. We first orient the edge $\{s,t\}$ from s to t. We than successively orient P_i from the lower-numbered to the higher-numbered end nodes and thereby label the inner nodes with unique fractional number in an increasing manner. Since the resulting orientation $\hat{G}$ of G contains only arcs (v, w) with $L(v) < L(w)$, $\hat{G}$ is acyclic and s and t are the only source and sink nodes, respectively. Hence, $\hat{G}$ is an s,t-orientation of G.

On the other hand, let $\hat{G}$ be an s,t-orientation of G w.r.t. an edge $\{s,t\} \in E$ and let the function L represent the corresponding s,t-ordering. We prove that $G \setminus \{v\} := (V \setminus \{v\}, E \setminus \delta(v))$ is connected for any $v \in V$. We show that any node $w \in V \setminus \{s,t,v\}$ is connected to s or to t in $G \setminus \{v\}$: Assume $L(v) < L(w)$. Due to the s,t-ordering there is the unique sink t in $\hat{G}$ and hence at least one oriented path from w to t using only nodes labeled at least with $L(w)$. By deleting v from $\hat{G}$, there may arise additional sinks; these will have labels strictly smaller than $L(v)$.

Hence, the oriented path(s) starting from w with $L(w) > L(v)$ to t remain after this deletion. Analogously, if $L(v) > L(w)$, there remains an oriented path from s to w in $\hat{G} \setminus \{v\}$.

As s and t are adjacent, every node w is connected to s in the undirected graph $G \setminus \{v\}$. Hence, $G \setminus \{v\}$ is connected. Similarly, if we delete the node s (or t), each node w is connected to t (or s, respectively). □

Using the above theorem and Lemma 2.8, we have an equivalent characterization of 2-node-connected graphs:

Corollary 2.10. *A graph $G = (V, E)$ is 2-node-connected if and only if it has an s, t-numbering w.r.t. any edge $\{s, t\} \in E$.*

Computing s, t-numbering and bipolar orientations has been subject of intensive research, for linear time algorithms see [ET76b, ET77, Ebe83, Bra02]. A survey on bipolar orientations can be also found in [dFdMR95].

2.3 Mathematical Programming for Network Design

2.3.1 Network Design Problems

A *combinatorial optimization problem* is to find the best (optimal) element from a given discrete set. Formally we can consider:

Problem 2.11 (Subset selection problem)**.** Given a finite set F, a set $\mathcal{I} \subseteq 2^F$ of subsets of F (the feasible solutions) and a function $c : F \to \mathbb{R}$. For each set $I \subseteq F$, let $c(I) = \sum_{f \in F} c(f)$. The *subset selection problem* $(F, \mathcal{I}, c)$ is to find a subset $I^* \subseteq F$ with

$$c(I^*) = \min\{c(I) \mid I \in \mathcal{I}\}.$$

In this thesis we deal with *(topological) network design problems*. In various industry applications, a network has to be designed that connects different objects (say customers) and satisfies a set of constraints. We will model such problems as a *subgraph selection problems*, which clearly belong to the class of subset selection problems. As input of these problems we have an undirected graph $G = (V, E)$, an edge weight function $w_E : E \to \mathbb{R}$ and/or a node weight function $w_V : V \to \mathbb{R}$. A subgraph selection problems is to find a subgraph C of G that satisfies a set of constraints and has minimum total weight.

The following problems are prominent examples of the network design problems modeled on graphs.

Problem 2.12 (MST)**.** The *minimum spanning tree problem* on an instance (G, w_E) is to find a subgraph $T = (V, E' \subseteq E)$ of G such that T is a tree with the minimum sum of the edge weights.

Problem 2.13 (STP)**.** The *Steiner tree problem* on an instance $(G, w_E, N \subseteq V)$ is to find a subgraph $T = (V', E')$ of G such that T is a tree with $N \subseteq V'$ and the minimum sum of edge weights. We call N the *terminal (or customer) nodes*.

Problem 2.14 (PCST)**.** The *prize-collecting Steiner tree problem* on an instance $(G, w_E, N \subseteq V, w_N : N \to \mathbb{R})$ is to find a subgraph $T = (V', E')$ of G such that T is a tree and the sum $\sum_{e \in E'} w_E(e) - \sum_{v \in N \cap V'} w_N(v)$ is minimized. I.e., not all terminal nodes have to be contained in the solution tree, but each terminal node has a *prize* (or *profit*) which is subtracted from the tree's cost if the node is selected.

Problem 2.15 (TSP)**.** The (symmetric) *traveling salesman problem* on an instance (G, w_E) is to find a subgraph $S = (V, E')$ such that S is a cycle spanning all nodes with the minimum sum of edge weights.

Whereas MST is polynomially solvable, the STP, PCST, TSP, and many other subgraph selection problems are NP-hard, i.e., computing an optimal solution will in general lead to a running time that is exponential in the size of the given instance (unless $P = NP$). This is the reason why *heuristics*, i.e., (fast) algorithms that compute a feasible solution but do not guarantee optimality, are often used to solve such problems.

Let I^* be an optimal solution of a combinatorial optimization problem. A heuristic that guarantees a solution I with $\frac{c(I)}{c(I^*)} \leq \alpha$ is called an *approximation algorithm* with *approximation factor* α.

This thesis, however, deals with *exact algorithms*, i.e., algorithms that output optimal solutions. A useful tool to solve various optimization problems exactly is mathematical programming. Generally, all subset selection problems can be formulated problems as *integer linear programs*, although with varying degree of practical efficiency. Modeling and solving network design problems as integer linear programs is the central topic of this thesis. A detailed overview over integer linear programming can be found, e.g., in [NW99, Sch98, Wol98]. In the next section, we give a brief introduction into this field based upon [Chi08, Lju04].

2.3.2 Linear Programing

A *linear program* (LP) $\min\{c^{\mathrm{T}}x \mid Ax \geq b, x \in (\mathbb{R}_0^+)^n\}$ is a system of linear inequalities and a linear objective function with a constraint matrix $A \in \mathbb{R}^{(m,n)}$, a cost vector $c \in \mathbb{R}^n$, and a right-hand side vector $b \in \mathbb{R}^m$. A variable vector $\bar{x} \in (\mathbb{R}_0^+)^n$ satisfying all the constraints $a_i^{\mathrm{T}}\bar{x} \geq b_i$, $i = 1, \ldots, m$, is called a *feasible solution*. We say a feasible solution x^* is *optimal* if it minimizes $c^{\mathrm{T}}x$.[1]

An LP can be solved using, e.g., the ellipsoid method, interior point methods or the Simplex algorithm. We can solve a (polynomially-sized) LP in polynomial time using the first two methods. The Simplex algorithm has an exponential worst-case running time. In practice, however, this algorithm is the simplest and most of the times fastest algorithm to solve linear programs.

Considering the above LP and restricting the vector x to be integer, we obtain an integer linear program (ILP). Furthermore, we deal with a *binary ILP* if we require x to be binary, i.e., $x \in \{0, 1\}^n$. All network design problems considered in this thesis can be formulated as a binary ILP. Yet, different formulations can be used for one and the same problem. Solving general ILPs and binary ILPs is

[1] Alternatively, LPs can also be defined to maximize the objective function. As this can be modeled by minimizing $-c^{\mathrm{T}}x$, we will only consider minimizations in the following.

NP-hard. However, various problem instances arising in industry applications can be solved efficiently by choosing an appropriate strong model and a sophisticated algorithm to solve it.

All ILPs described in this thesis are binary ILPs. Nevertheless, we will always use the term ILP for better readability.

2.3.3 Measuring Formulation Strength

In order to be able to compare different ILP models we need some insight in polyhedral theory. Let the matrix A and the vector b be defined as above. A *polyhedron* is a set that can be described in the form $\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \geq b\}$. A polyhedron is bounded—and therefore called *polytope*—if there exists a $w \in \mathbb{R}$ such that $\mathcal{P} \subset \{x \in \mathbb{R}^n \mid -w \leq x_i \leq w, \forall i = 1, \ldots, n\}$. In the following, we always deal with polytopes.

Consider a subset selection problem $(F, \mathcal{I}, c)$ with associated linear objective function c. We represent every feasible subset $I \subseteq F$ of a subset selection problem by the means of the *incidence vector* $h_I \in \{0,1\}^F$ that is given by $h_I(f) = 1$ if $f \in I$ and $h_I(f) = 0$, otherwise. It is known that minimizing an objective function over $\mathcal{I}$ is equivalent to minimize the same objective function over the convex hull of $\mathcal{I}$. Therefore, we can consider the polytope

$$\mathcal{P}_\mathcal{I} = \text{conv}\{h_I \mid I \in \mathcal{I}\},$$

i.e., the convex hull of the incidence vectors of all feasible sets $I \in \mathcal{I}$. Any such polytope can be represented by using its *linear description*, i.e., a finite set of inequalities

$$\mathcal{P}_\mathcal{I} = \{x \in \mathbb{R}^{|F|} \mid Ax \geq b\}.$$

Generally, it is not possible to give a polynomially-sized description of this convex hull. Hence, it is common to use polyhedra that are larger than $\mathcal{P}_\mathcal{I}$. One possibility is to use the polyhedron corresponding to the *LP relaxation* of the given ILP, i.e., the LP resulting from replacing the integrality constraints $x \in \{0,1\}^n$ by $x \in [0,1]^n$. Clearly, each feasible solution of the original ILP is also feasible for its LP relaxation. On the other hand, there are no additional integer solutions. Thus, a (fractional) solution of such an LP relaxation is a lower bound to the optimal ILP solution. Note that if this solution is integer, we found an optimal solution to our initial ILP. Both the branch-and-bound and branch-and-cut algorithms described in the next section use such LP-based lower bounds. Hence, given two different ILP models for one and the same problem, we can compare them by comparing the corresponding LP relaxations.

Definition 2.16 (Strength of LP relaxations, cf. [PD01c])**.** We say a relaxation $\mathcal{R}_1$ is *weakly stronger* than a relaxation $\mathcal{R}_2$ if the optimal value of $\mathcal{R}_1$ is no less than that of $\mathcal{R}_2$ for all instances of the problem. If $\mathcal{R}_2$ is also weakly stronger than $\mathcal{R}_1$, we call them *equivalent*, otherwise we say that $\mathcal{R}_1$ is *strictly stronger* than $\mathcal{R}_2$. If neither is stronger than the other, they are incomparable.

Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be the polytopes associated with the feasible solutions of the LP relaxations $\mathcal{R}_1$ and $\mathcal{R}_2$ for a given instance of a subset selection problem. Note that these polytopes do not depend on the given cost function. Optimizing a cost function over these polytopes gives us a lower bound on the optimal solution. In order to compare these polytopes, we have to project them into a suitable common variable space, say x'. As both contain the set of integer feasible solutions, we have $\text{proj}_{x'}(\mathcal{P}_1) \cap \text{proj}_{x'}(\mathcal{P}_2) \neq \emptyset$. Throughout this thesis we use the following observation:

Observation 2.17. *Let $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{P}_1$ and $\mathcal{P}_2$ be defined as above.*

- *The relaxation $\mathcal{R}_1$ is weakly stronger than $\mathcal{R}_2$ for all cost functions, if for all problem instances we have*

$$\text{proj}_{x'}(\mathcal{P}_1) \subseteq \text{proj}_{x'}(\mathcal{P}_2).$$

- *The relaxations $\mathcal{R}_1$ and $\mathcal{R}_2$ are equivalent for all cost functions, i.e., the lower bounds of both relaxations are equally strong, if and only if $\text{proj}_{x'}(\mathcal{P}_1) = \text{proj}_{x'}(\mathcal{P}_2)$ for all instances of the problem.*

- *For a given cost function $\bar{c}$ the relaxation $\mathcal{R}_1$ is strictly stronger than $\mathcal{R}_2$, if $\mathcal{R}_1$ is weakly stronger than $\mathcal{R}_2$ and there exists a problem instance for which optimizing over $\mathcal{P}_1$ leads to stronger lower bounds as over $\mathcal{P}_2$, i.e.,*

$$\min\{\bar{c}(x') \mid x' \in \text{proj}_{x'}(\mathcal{P}_2)\} \lneq \min\{\bar{c}(x') \mid x' \in \text{proj}_{x'}(\mathcal{P}_1)\}.$$

Definition 2.18 (Strength of ILPs)**.** For a given subset selection problem, let $\mathcal{F}_1$ and $\mathcal{F}_2$ be different ILPs with their corresponding LP relaxations $\mathcal{R}_1$ and $\mathcal{R}_2$. We say $\mathcal{F}_1$ is weakly stronger (strictly stronger, equivalent) than $\mathcal{F}_2$ if and only if $\mathcal{R}_1$ is weakly stronger (strictly stronger, equivalent, resp.) than $\mathcal{R}_2$.

2.3.4 Solving Binary ILPs

Consider an ILP and the polytope associated with its feasible solutions. As described above, it is in general not possible to find a polynomially-sized linear description of this polytope. Therefore, we first optimize over a larger polytope, e.g., corresponding to its LP relaxation, obtaining a fractional solution that gives us a lower bound on the optimal solution of the original ILP.

Cutting planes. If the solution to the LP relaxation is not integer, we may try to identify *cutting planes*, also known as *integer-valid cuts*, i.e., constraints that that are satisfied by all feasible integer points but that are violated by the current fractional solution. The integer-valid cuts are then added to the current model and the problem is resolved iteratively. When no integer-valid cuts can be found, we obtain a (hopefully strong) lower bound for the original ILP.

Recall that polynomially-sized LPs are solvable in polynomial time. However, in the context of combinatorial optimization problems we often deal with an ILP that requires an exponential number of constraints. In order to solve its LP relaxation

(efficiently), we choose a subset of all ILP constraints and solve the corresponding LP relaxation. We then solve the *separation problem*, i.e., we check if the current solution violates any original ILP constraint and identify such constraints. We iterate this process, adding such constraints and resolving the resulting LP, until no more violated constraints can be found.

A particularly interesting theorem central to LP theory by Grötschel, Lovász, and Schrijver (cf., e.g, [Wol98]) shows the equivalence of optimization and separation, i.e., if we can solve the separation problem in polynomial time, we can also solve the underlying full LP in polynomial time. Hence, given an ILP with exponentially many constraints we can obtain the optimal fractional solution of the corresponding LP relaxation in polynomial time if the corresponding separation problem is polynomially solvable.

Branch-and-bound. In general, the cutting plane algorithm will not yield an integer solution. The most widely used approach to solve ILPs to provable optimality is to use a branch-and-bound strategy, i.e., the original problem is solved by recursively subdividing it into several (usually two) subproblems. A common branch-and-bound strategy for solving a given binary ILP is the following:

First, a lower bound is computed by solving the LP relaxation of the original problem. If this solution is integer, it is already optimal and the algorithm stops. Otherwise, we have to resort to *branching*, i.e., we generate two disjoint subproblems, e.g., by fixing a variable to 0 or 1. We then solve the corresponding LPs and use their solutions as local lower bounds. If such a solution is integer we can also use it as a global upper bound. Further upper bounds can be obtained by using *primal heuristics* that compute heuristic solutions to the original problem. Thereby, such a solution can be either computed in advance (*initial heuristic*) or by incorporating the current fractional solutions (*LP-based heuristic*). These bounds are then used to prune irrelevant subproblems or to prove the optimality of the current upper bound. The branching process can be represented by a *branch-and-bound tree*, where each node corresponds to a subproblem and its children to its further variable fixings.

Branch-and-cut. A branch-and-cut algorithm [PR91] is a combination of the cutting plane approach and the branch-and-bound strategy. Thereby, cutting planes are generated in every node of the branch-and-bound tree. Branch-and-cut turned to be very effective in practice, and lead to the most effective optimal algorithms in many applications.

Part II

k-Cardinality Tree Problems

Chapter 3

Considered Problems

Consider a real-world scenario as it arises in a telecommunication industry: We have n cities and a set of connections between them (e.g., streets). Furthermore, there are costs for using a connection, e.g., rental costs or working costs for cable laying. We assume that a telecommunication company is permitted to service only $k+1$ of the customers and thus wants to find the $k+1$ cities such that the cost of connecting them is minimal. This problem can be easily modeled on a graph, considering the cities as nodes and the connections between them as edges with corresponding edge-weights. The solution of our problem is a minimum-weight subtree of this graph containing exactly k edges.

Another example of our problem originates from the oil-industry [FHJM94]: In the 1990s, Norwegian government leased oil-fields to companies allowing them to explore these fields. After six years a company had to return a connected part of the territory which had to contain at least a half of the original oil-field. Naturally, the essential task for a company was to find the most unprofitable such part of the field. By dividing the original field into subsquares and using the six years of rent to explore the corresponding profits of these, this task can be modeled as a k-cardinality tree problem as follows: Each subsquare is represented by a node. We have an edge between two nodes if the corresponding subsquares are neighbors. To each node of a graph we assign a weight which is the estimated value of the corresponding subsquare. A connected subgraph of this graph containing exactly half of the graph nodes with the minimum sum of node-weights constitutes an optimal solution to the original problem.

Both problems above, as well as other applications, e.g., in quorumcast routing [CK94], telecommunications [GH97], open pit mining [PW97] and facility layout [FH92] can be modeled as a *k-cardinality tree problem*:

Input. An undirected graph $G = (V, E)$, a positive integer number k, and—as suitable—an edge weight function $w_E : E \to \mathbb{R}$ and/or a node weight function $w_N : V \to \mathbb{R}$.

Problem 3.1 (EKCT)**.** The *edge-weighted k-cardinality tree problem* on an instance (G, w_E, k) is to find a subgraph T of G which is a tree with exactly k edges and the minimum sum of the edge weights.

The EKCT problem is also known as the *k-minimum spanning tree problem* (k-MST), whereby the parameter k usually specifies the number of selected nodes, which in a tree is exactly one more than the number of its edges. However, in our notations k only specifies the number of edges in a solution. In the literature, the edge weights are often restricted to be non-negative; our approach, presented in this thesis, does not require this assumption.

Problem 3.2 (NKCT)**.** The *node-weighted k-cardinality tree problem* on an instance (G, w_N, k) is to find a subgraph T which is a tree with exactly k edges and the minimum sum of the node weights.

Although not prominent in the literature, we can straightforwardly generalize both above problems to:

Problem 3.3 (AKCT)**.** The general *all-weighted k-cardinality tree problem* on an instance (G, w_E, w_N, k) is to find a subgraph T which is a tree with exactly k edges and the minimum sum of edge- and node-weights.

In some applications it may be useful to require a solution to include some given root node. Hence, we classify all above problems as *unrooted KCT* and define their rooted counterparts:

Problem 3.4 (Rooted KCT)**.** Given an input of an unrooted KCT problem and a specified root node $r' \in V$, find an optimal solution to this problem containing r'. We may specify a corresponding instance by $(G, r', [w_E], [w_N], k)$, giving the weight functions as necessary.

We summarize all above problem variants under the general abbreviation *KCT*. The following two problems are closely related to KCT:

Problem 3.5 (KST)**.** Given a subset $C \subseteq V$ of terminals, the *k-Steiner tree problem* asks for the edge-weight-minimum tree spanning k terminal nodes. Thereby the solution may contain some arbitrarily many non-terminal, a.k.a. Steiner, nodes.

Problem 3.6 (KPCST)**.** Analogously to the KST, we can consider the *k-cardinality prize-collecting Steiner tree problem*, which is to find a prize-collecting Steiner tree with the side-condition that k terminal nodes have to be selected. The KST problem is then the special case of the KPCST where all prizes of the terminal nodes are 0. Note that the general KPCST does not require the edge weights nor the node prizes to be non-negative.

In this part of the thesis we develop a new exact algorithm for KCT problems that will also be able to solve K(PC)ST problems: In Chapter 4, we give a literature overview on these and related problems, paying special attention to ILP-based algorithms. Most of these were based on undirected graphs. In Chapter 5, we discuss an orientation property of trees that gives us a valid transformation of our KCT problems into a k-cardinality arborescence problem (KCA). We then give a novel ILP formulation for KCA and use it to solve all k-cardinality problems defined above. In order to show the advantages of this formulation, we compare it to the previously known formulations from a polyhedral point of view. Our branch-and-cut algorithm is given in Chapter 6. Its performance is experimentally analyzed and compared with the state-of-the-art algorithms for these problems in Chapter 7.

Chapter 4

Literature Overview

KCT problems were originally considered in [HJM91]. Its complexity and polyhedral structure are analyzed in [FHJM94]. Since then, algorithmic solutions for these problems have been subject of intensive research for various scientific communities. In the following we discuss the complexity issues and give an overview over existing algorithmic solutions for KCT, K(PC)STP, and other closely related problems.

4.1 Related Problems

The following problem is a special case of the EKCT problem, as it is defined on the complete graph K_n with corresponding Euclidean distances as edge weights:

Problem 4.1 (PEKCT)**.** Given n points in the Euclidean *plane*, the *Euclidean k-cardinality tree problem* is to find the minimum-weight tree containing exactly $k+1 < n$ points. Thereby we consider the Euclidean distance between two points as the weight of a connection between them.

A prominent relaxation of the EKCT problem is:

Problem 4.2 (KCS)**.** Consider the input of EKCT. The *k-cardinality subgraph problem* is to find a minimum edge-weight connected subgraph with k edges.

4.2 Complexity

4.2.1 Relation Between KCT Problems

EKCT and NKCT. Given an NKCT instance (G, w_N, k), we can obtain an EKCT instance $(G', w_E, 2k+1)$ in polynomial time as follows [EFHM97]: We construct G' from G by adding a dummy node v' for each node $v \in V$ and connecting them via edges $e' = \{v, v'\}$ with $w_E(e') := w_N(v)$. For each original edge $e \in E$ we set $w_E(e) := (|E|+1)\max_{v \in V} |w_N(v)|$. This high weight ensures that in the corresponding optimal EKCT solution as few of these edges are selected as possible. An optimal NKCT solution of (G, w_N, k) can be easily deduced from an optimal EKCT solution of $(G', w_E, 2k+1)$ by deleting the dummy edges.

Hence, the NP-hardness of general NKCT implies the NP-hardness of general EKCT. However, historically the complexity of both problems was proven independently, see below. In [BE03] it is mentioned that both problems are equivalent if the input graph itself is a tree, see also Section 4.2.3.

Rooted and unrooted KCT. In [ABV95] it is shown that the EKCT problem and its rooted version are essentially equivalent. Most of the known approximation algorithms originally solve the rooted EKCT. The corresponding unrooted problem is then solved by considering the rooted version $|V|$ times, once for each possible choice of the root node.

4.2.2 NP-Hardness

EKCT, NKCT and KCS problems (and therefore also the AKCT problem) are in general strongly NP-hard as it has been shown in [FHJM94]. For the EKCT problem, the NP-hardness was also indepedently proved in [RSM$^+$96, ZL93]. The proof is based on a reduction from the prominent Steiner tree problem to the EKCT problem.

The following NP-hardness results are known for some restricted cases:

- The EKCT problem is NP-hard even if the input graph is planar or the edge-weights are restricted to $w_E(e) \in \{1, 2, 3\}$ for all edges $e \in E$ [RSM$^+$96].
- The PEKCT problem is also NP-hard [RSM$^+$96].
- The NKCT problem remains NP-hard on grid graphs, arbitrary planar graphs, split graphs, bipartite graphs, chordal graphs, and perfect graphs [Woe92].

Furthermore, in [Woe92] it is observed that the NKCT and the Steiner tree problems are closely related in the sence that their complexity classes for the investigated graph types are identical.

4.2.3 Polynomially Solvable Cases

The following restricted cases of the KCT problems are polynomially solvable:

- If $k = |V| - 1$, the EKCT problem becomes the classical minimum spanning tree problem, which is polynomially solvable, e.g., by the algorithms of Kruskal or Prim (cf., e.g., [CLRS01]). Furthermore, the optimal value of the corresponding NKCT problem is simply the sum of all given node-weights.
- If k is fixed, the EKCT problem is polynomially solvable by enumeration: for each possible set of $k + 1$ nodes compute the minimum spanning tree and afterwards choose the one of those with miminum weight. Analogously, the same holds if $k' = |V| - k$ is fixed by choosing complementary sets.
- If G is complete, the NKCT problem is trivially solved by selecting $k+1$ nodes with smallest weights.

- If G is complete and w_E has only two different values, EKCT is also polynomially solvable [RSM+96]: first, a not necessarily connected subgraph containing all edges of the lower weight is constructed. We then choose the minimum number of its connected components (say ℓ) containing $k+1$ nodes in total (dropping nodes if necessary) and construct a minimum spanning tree for each such component. These trees are then connected by $\ell - 1$ edges of the larger weight.

- In [Maf91] it was shown that if G is a tree, there is a dynamic programming algorithm that runs in $O(k^2|V|)$ time and solves EKCT exactly. This algorithm is also sketched in [FHJM94]. Blum [Blu07] extended this algorithm such that it solves both EKCT and NKCT problems and outputs not only the value of the optimal solution, but also the corresponding tree. For the NKCT problem on trees, a polynomial-time algorithm was also given in [Woe92].

- Ravi et al. [RSM+96] presented a dynamic programming algorithm with the same running time as for trees for the more general class of graphs with bounded tree-width [BLW87, ACPS93]. This class of graphs contains, e.g., trees, series-parallel, and bounded-bandwidth graphs.

- The NKCT problem is polynomially solvable for interval graphs and cographs [Woe92]. The corresponding dynamic programming algorithms have worst-case running times of $O(k^2|V|^2)$ and $O(k^2|V|)$, respectively, and require $O(k|V|)$ space.

- The NKCT problem is polynomially solvable by a greedy algorithm on the class of graphs containing exactly one trough [BE03]: For $S \subset V$ let $G[S]$ be the thereby induced subgraph. $G[S]$ is a *trough* if all nodes S have equal weight and for all $(u, v) \in E$ with $u \in S$ and $v \in V \setminus S$ we have $w_N(u) \leq w_N(v)$.

- The PEKCT problem is solvable in $O(k^2n^7)$ for n points in the Euclidean plane that lie on the boundary of a convex region. If the given n points lie on a circle, there is an $O(k^2n)$ algorithm [RSM+96].

4.3 Preprocessing

In [RSM+96, BRV96] it is mentioned that if the input edge-weights of the EKCT problem are positive, we can assume that they satisfy the triangle-inequality. Although never explicitly mentioned in the literature, this gives the following preprocessing routine:

Lemma 4.3. *Let $(G, w_E : E \to \mathbb{R}^+, k)$ be the input of the EKCT problem. If for an edge $e = \{v, w\}$ we have $w_E(P) \leq w_E(e)$, where $P = [v \to w]$ is the shortest path from v to w in $G - e$, then e will not be contained in any optimal solution of the EKCT problem.*

Proof. Let T be an optimal k-cardinality tree with $e \in E(T)$.

$|P| \geq k$**:** Let $P' \subseteq P$ with $|P'| = k$ be a subpath of P of minimum length, then $w_E(P') < w_E(T)$ (even if $w_E(P') = w_E(e)$) which is a contradiction to the optimality of T.

$|P| < k$**:** We construct a tree T' with $w_E(T') \leq w_E(T)$ from T by deleting e and inserting all edges from P that are not already part of T. We thereby have $w_E(T') \leq w_E(T)$.

- If we insert only one edge $e' \in P$, i.e., we have T' is a k-cardinality tree with $w_E(T') < w_E(T)$ as $w_E(e') < w_E(e)$. This is a contradiction to the optimality of T.
- If we have to insert more than one edge, we can reduce T' to a k-cardinality tree T'' by pruning T'. Thereby we have $w_E(T'') < w_E(T') \leq w_E(T)$ which is a contradiction to optimality of T.

□

The above lemma can be used in order to preprocess and reduce our instances: using a (polynomial) all-pair shortest path algorithm we identify edges with the above property and delete them from the input graph.

4.4 Heuristic Algorithms

4.4.1 Heuristics

A collection of heuristics for EKCT and NKCT including an experimental comparison, is presented in [EFHM97].

EKCT. In [EFHM97], three classes of heuristics were presented for the EKCT problem:

- Algorithms based on the spanning tree algorithms (k-CardPrim).
- Algorithms based on shortest paths methods.
- Algorithms based on the dynamic programming algorithm of [Maf91].

Later, improved heuristics based on the dynamic programming algorithm for trees were presented in [Blu07].

NKCT. All known EKCT heuristics can be applied to the NKCT problem using the polynomial-time transformation of [EFHM97], also described in Section 4.2. Although not considered in [EFHM97], note that for transforming an EKCT solution T' back to its NKCT counterpart an additional pruning step may be necessary, as T' (being only a heuristic solution) may contain more than k edges of the original NKCT instance.

4.4.2 Metaheuristics

EKCT. A large amount of research was devoted to the development of metaheuristic methods for EKCT such as tabu search [JL97, BB05b], ant colonies [BS04, BB05a], evolutionary algorithms [Blu06, BB05b], as well as variable neighborhood search and variable neighborhood decomposition search [UBM04].

NKCT. A local search routine as well as genetic and tabu search algorithms for the NKCT problem were presented in [BE03]. Further known NKCT metaheuristics are different variable neighborhood search heuristics [BUM06] and a combination of ant colony optimization with dynamic programming [BB05a].

4.4.3 Approximation Algorithms.

The EKCT problem has received a lot of attention in the approximation algorithm community. The first approximation algorithm was given in [RSM$^+$96] offering an approximation factor of $2\sqrt{k}$. This factor was then improved in [ABV95] to $\log^2 k$. Both algorithms are constructive and use the ideas of Kruskal's algorithm for minimum spanning trees and Dijkstra's shortest path algorithm.

The first approximation algorithm for EKCT with a constant factor was proposed in [BRV96] and featured factor 17. This factor has been reduced to 3 in [Gar96], then to $2+\epsilon$ in [AK00]. Finally, the approximation factor 2 was achieved in [Gar05]. All above algorithms with a constant approximation factor are based on the primal-dual scheme, originally proposed by Goemans and Williamson [GW95] for the prize-collecting Steiner tree problem which can also be interpreted as a constructive algorithm using components of Kruskal's and Dijkstra's algorithms.

Note that most of the above algorithms solve the rooted EKCT problem and should be run for every node of the input graph in order to solve the unrooted EKCT problem.

For the NKCT problem on a grid and with $k = \ell^2$ for some positiv integer ℓ an $O(\sqrt{k})$ approximation algorithm was suggested in [Woe92]. The idea of this algorithm is quite simple: Find a $\sqrt{k} \times \sqrt{k}$-subgrid of minimum value and compute its spanning tree.

In [CRW01], the authors present an ILP for KSTP based on undirected graphs and use it to derive a 5-approximation. It was shown in [BRV96] that an α-approximation algorithm for the EKCT yields a 2α approximation for the KSTP. As the best known approximation factor for the EKCT is 2, this yields an approximation factor of 4 for the KSTP. In [BRV96], the authors sketch ideas for 3- and even $(2 + \epsilon)$-approximations.

For the PEKCT problem, after multiple approximation algorithms with approximation factors depending on k [RSM$^+$96, GH97, Epp97] or being constant [BCV95, MBCV99], two independent polynomial approximation schemes where proposed in [AK00, Mit99].

4.4.4 Experimental Studies

To our knowledge, no experimental studies exist with regard to approximation algorithms for KCT problems.

In [EFHM97], the behaviour of different heurisic algorithms for the EKCT and NKCT problems were experimentally compared on randomly generated connected graphs and grid graphs. Both graph classes contained up to 30 nodes. Approaches based on dynamic programming (*DynamicTree*) were the best in terms of solution quality. The same instances were also used to compare metaheuristic algorithms [BE03].

A study on the DynamicTree heuristic for both EKCT and NKCT on the more up-to-date benchmark set *KCTLIB* [BB03] is presented in [Blu07].

An extensvise study [Blu06] on the KCTLIB benchmark set shows that the best metaheuristics both in terms of solution quality and running time are: an evolutionary algorithm that uses exact dynamic programming algorithm for trees as a subroutine [Blu06], an ant-colony optimization algorithm [BS04] and, for some instances also the variable neighborhood search algorithm of [UBM04].

For the NKCT problem, it is shown in [BUM06] that their variable neighborhood decomposition search algorithm outperforms metaheuristic algorithms of [BE03] on the large node-weighted instances of the KCTLIB. Later, the authors of [BB05a] claim that their ACO algorithm outperforms [BUM06] in terms of solution quality on NKCT problems. However, results are only given for the node-weighted grid graphs on 900, 1600 and 2500 nodes and not for the random node-weighted graphs of the same instance library. Indeed, when grid instances grow, the ACO algorithm tends to obtain better solutions for small k.

The KSTP is only considered from a theoretical point of view. To our knowledge, there are no established benchmark instances nor any experimental studies.

4.5 ILP-based Exact Algorithms

4.5.1 General Subtour Elimination Constraints

The first exact approach for the (unrooted) KCT problems has been presented in [FHJM94], by formulating an integer linear program based on *generalized subtour elimination constraints* (GSECs). This formulation is based on the given undirected graph. We rephrase it here using our notations.

We have a binary variable $z_e \in \{0, 1\}$ for each edge $e \in E$, which is set to 1 if e is in the solution (i.e., $e \in E(T)$), and 0 otherwise. Furthermore there is a binary variable y_v for each node $v \in V$ which indicates whether v is in the solution ($y_v = 1$) or not ($y_v = 0$). The following constraint system describes all k-cardinality trees in G. Generally, given a variable vector ξ and a set of indices $\mathcal{J}$, we will use the shorthand $\xi(\mathcal{J}) := \sum_{j \in \mathcal{J}} \xi_j$ in the remainder of this thesis.

$$\text{KCT-GSEC} := \{(4.1)\text{–}(4.4)\}$$

with

$$z(E) = k \tag{4.1}$$
$$y(V) = k + 1 \tag{4.2}$$
$$z(E(S)) \leq y(S \setminus \{t\}) \qquad \forall S \subseteq V, |S| \geq 2, \forall t \in S \tag{4.3}$$
$$z_e, y_v \in \{0, 1\} \qquad \forall e \in E, v \in V. \tag{4.4}$$

The inequality (4.1) ensures that there are exactly k edges in the solution. The inequalities (4.3) are the generalized subtour elimination constraints. As their name already suggests, this constraint class is a strengthened generalization of the *subtour elimination constraints (SECs)* which only use edge variables:

$$z(E(S)) \leq |S| - 1.$$

Both constraint classes ensure that the solution does not contain cycles. With (4.2) we furthermore guarantee that the solution contains $k + 1$ nodes, i.e., the resulting solution is a tree.

Hence, minimizing weights of the selected edges over KCT-GSEC gives us an optimal solution of the EKCT problem, i.e., we can consider the ILP

$$\min \left\{ \sum_{e \in E} w_E(e) \cdot z_e : (z, y) \text{ satisfies KCT-GSEC} \right\}. \tag{4.5}$$

Analogously, we obtain an ILP for the NKCT problem:

$$\min \left\{ \sum_{e \in E} w_N(e) \cdot z_e : (z, y) \text{ satisfies KCT-GSEC} \right\}. \tag{4.6}$$

We may summarize both ILPs simply as KCT-GSEC ILPs. The above formulations contain an exponential number of constraints. The authors suggest a branch-and-cut algorithm, using an $O(|V|^4)$ separation routine for the inequalities of type (4.3). Furthermore, the paper contains an extensive polyhedral analysis of the above EKCT model.

4.5.2 Undirected Cuts

In [Gar96], Garg considers approximation algorithms for the EKCT problem. To this ends, he presents an ILP based on undirected cuts for the rooted EKCT problem, whose LP relaxation is subsequently used for lower bounds. The formulation uses a root node r' that is selected among the original nodes. To solve the general EKCT problem, he considers $|V|$ many ILPs, one for each possible choice of node r', since it cannot be guaranteed that r' is contained in the optimal solution. We rephrase the original formulation, adapted to our notation. The y and z variables are used analogously to the GSEC-based formulation.

$$\min \sum_{e \in E} w_E(e) \cdot z_e$$

$$y(V) = k + 1 \tag{4.7}$$

$$z(\delta(S)) \geq y_v \qquad \forall S \subseteq V \setminus \{r'\}, \forall v \in S \tag{4.8}$$

$$z_e, y_v \in \{0, 1\} \qquad \forall e \in E, \forall v \in V \tag{4.9}$$

We call the inequalities (4.7) and (4.8) *node cardinality* and *undirected cut constraints*. As Garg assumes the edge weights to be positive, he does not have to restrict the number of selected edges. To allow general edge weights we can add the constraint:

$$z(E) = k \tag{4.10}$$

In fact, adding this constraint strengthens the corresponding LP relaxation even for positive edge weights, cf. Section 5.4.2.

Although not mentioned in [Gar96], we obtain an ILP for the rooted NKCT problem by suitably replacing the above objective function and using (4.10). Hence, the feasible solutions to the rooted KCT problems can be described by the constraint system

$$\text{rKct-UCut} := \{(4.7)\text{–}(4.10)\}.$$

We will show in Section 5.4.2 that rKct-UCut can be modified to directly describe the solutions to the unrooted KCT problems.

4.5.3 Miller-Tucker-Zemlin SECs and Multicommodity Flows

At the same time as the conference version of our results [CKLM08a] was published, Quintão et al. [QCM08] presented two alternative orientation-based ILPs for KCT. They transform a given KCT instance in an instance of a *constrained minimum spanning arborescence problem (MSA′)* (cf. Section 5.4.4). Although somewhat more complicated, this transformation is similar to our transformation strategy described in Section 5.2. The feasible solutions of the MSA′ problem are then described by two different sets of inequalities. The first such set is based on multi-commodity flows, whereas the second uses *Miller-Tucker-Zemlin subtour elimination constraints (MTZ SECs)*. We call these formulations Msa′-DFlow and Msa′-MTZ, respectively, and describe them in Section 5.4.4 in more detail.

4.5.4 Experimental Studies

For a long time, the only known practical implementation of the above ILP-based exact algorithms was a branch-and-cut algorithm presented in [EF96]. It was based on the GSEC model as suggested in [FHJM94]. This algorithm was only able to solve unrooted EKCT and NKCT problems on graphs with up to 30 nodes, which may be mainly due to the comparably weak computers in 1996.

Very recently, Quintão et al. [QCML10] presented computational results based on their orientation-based ILPs. Thereby they used some of the (edge-weighted) KCTLIB instances [BB03] and node-weighted grid instances, generated as suggested in [BUM06]. This experimental study shows that the multi-commodity flow based formulation delivers stronger lower bounds for all considered instances as the MTZ SECs based formulation. Indeed, although not shown in [QCML10], the Msa′-DFlow ILP is strictly stronger than the Msa′-MTZ ILP, as it was already the

case for the related Steiner tree problem. Since solving the LP relaxation of MSA′-DFLOW is very time consuming, a branch-and-bound algorithm based on MTZ SECs and a Lagrangian heuristic based on MSA′-DFLOW ILP were developed and experimentally evaluated.

4.6 Our Contribution

In this thesis, we show how to model all the KCT problems defined above using a simple orientation property of trees. Based on this modeling we formulate an orientation-based ILP that can be used to solve all the KCT problems defined in Section 3, both in their rooted and unrooted variants. Furthermore, it can be easily adapted to K(PC)ST problems. Within our ILP we express the connectivity requirements by the means of *directed cuts*. Our model can be alternatively realized by the means of a multi-commodity flow based ILP, which we show to be equivalent with our directed cut based formulation. We furthermore conduct a polyhedral comparison of our formulations to all other ILPs mentioned in this chapter.

Based on our directed cut based formulation, we develop a branch-and-cut algorithm which is the first algorithm that solves all known KCT benchmark instances to provable optimality. Being able to optimally solve both edge- and node-weighted instances with up to 2500 nodes, it also clearly outperforms all other (meanwhile) published exact algorithms. Using comparable computers, our algorithm computes optimal solutions even faster (by orders of magnitude) than the LP-based Lagrangian *heuristic* of [QCML10] that does not guarantee optimality of its solution. Interestingly, although we informed one of the authors on our results long time before their final revision of [QCML10] was submitted, Quintão et al. do not even mention our results in their publication.

Moreover, for the EKCT problem our approach even outperforms the state-of-the-art metaheuristics in terms of both running time and, evidently, solution quality for graphs with roughly up to 1000 nodes. An often used argument in particular for the metaheuristic approaches is that exact methods for this NP-hard problem would require too much computation time and could only be applied to very small graphs [BE03, BUM06]. Hence, with our algorithm we show that this traditional argument for metaheuristics over exact algorithms is deceptive on KCT-type problems.

Chapter 5

Orientation-based Modeling

5.1 Feasible Orientations of Trees

All previously known ILP models for KCT were based on undirected graphs. However, for related network design problems, e.g., the Steiner tree problem or its prize-collecting variant, using the following simple orientation property of a tree leads to stronger ILP formulations that are also beneficial in practice [PD01c, LWP+06].

Lemma 5.1 (Orientation of trees)**.** *Let $T = (V, E)$ be a tree. For any choice of a root node $r \in V$, we can find a unique orientation $\hat{T}$ such that for every node $v \in V \setminus \{r\}$ there exists a directed $(r \to v)$-path. Note that $\hat{T}$ is then an arborescence.*

Observation 5.2. *The in-degree of the root node of an arborescence is always 0.*

The following theorem is quite intuitive:

Theorem 5.3. *An undirected graph T is a k-cardinality tree if (and only if) for any choice of a root node $r \in V(T)$ there exists a unique orientation $\hat{T}$ which is a* k-cardinality arborescence rooted at r*, i.e., $\hat{T}$ satisfies the following properties:*

(P1) $\hat{T}$ contains exactly k arcs,

(P2) $\hat{T}$ contains exactly $k + 1$ nodes, and

(P3) for all $v \in V(\hat{T}) \setminus \{r\}$, there exists a directed path $(r \to v)$ in $\hat{T}$.

5.2 Transformation into the KCA Problem

Using Theorem 5.3, we give a transformation of a KCT instance to an equivalent instance of the following problem:

Problem 5.4 (KCA)**.** Let $D = (V_D, A_D)$ be a directed graph with a specified root node $r_D \in V_D$ and arc costs $c(a)$ for all arcs $a \in A_D$. The *k-cardinality arborescence problem* on the instance (D, r_D, c, k) is to find a minimum weight rooted subtree $T_D \subseteq D$ with k arcs which is directed from the root outwards. More formally, T_D has to be a k-cardinality arborescence.

EKCT. Given an instance (G, r', w_E, k) of a rooted EKCT problem (i.e., $r' \in V$), we solve it by solving the KCA problem on $(\bar{G}, r', c, k)$, where $\bar{G}$ is the bidirection of G and the arc weights are $c((i,j)) = c((j,i)) := w_E(\{i,j\})$ for every edge $\{i,j\} \in E$.

For the unrooted EKCT, we do not know which node will certainly be contained in an optimal solution and therefore we would have to solve $|V|$ many KCA problems, one for each possible choice of the root node. We can avoid this by extending $\bar{G}$ via an artificial root node $r \notin V$ and connecting r to every node in $v \in V$ with arcs (r, v), cf. Figure 5.1. Hence we obtain a digraph $\bar{G}_r = (V \cup \{r\}, A \cup A_r)$ with $A = \{(i,j), (j,i) \mid \{i,j\} \in E\}$ and $A_r = \{(r,j) \mid j \in V\}$. The weight for an arc (i,j) in $\bar{G}_r$ is $c((i,j)) := 0$ if $i = r$, and $c((i,j)) = c((j,i)) := w_E(\{i,j\})$ otherwise. To be able to interpret any feasible solution $T_{\bar{G}_r}$ of the resulting KCA instance $(\bar{G}_r, r, c, k+1)$ as a solution to the original KCT instance, we have to impose the additional constraint

(P4) $T_{\bar{G}_r}$ contains only a single arc of A_r.

If this property is satisfied, it is easy to see that a feasible KCT solution with the same objective value can be obtained by removing r from $T_{\bar{G}_r}$ and interpreting the directed arcs as undirected edges. On the other hand, for each feasible solution to the KCT problem there exists a feasible solution to the KCA problem in the correspondingly transformed instance.

NKCT and AKCT. As [Seg87] did for Steiner trees, we can use the following observation in order to transform the NKCT and AKCT problems into a KCA problem.

Observation 5.5 ([Seg87])**.** *Any subgraph T_D of a directed graph D satisfying (P1)–(P3) also satisfies: For all $v \in V(T_D) \setminus \{r\}$, v has in-degree 1 in T_D.*

This allows us to establish a one-to-one correspondence between each selected arc and its target node (disregarding the root node). Whenever we select an arc, we know that we have to consider the cost of its target node v, and we will never consider this node weight multiple times since we will only select a single arc having v as its target node. This allows us to shift the node weights of the NKCT and AKCT problems into the arc weights of the KCA problem.

If we deal with a rooted NKCT (AKCT) problem instance, we can hence solve it by solving the KCA problem on $(\bar{G}, r', c, k)$ with arc weights $c((i,j)) := w_N(j)$ ($c((i,j)) := w_E(\{i,j\}) + w_N(j)$, respectively). Note that the value of the optimal solution for the rooted NKCT or AKCT problem is the value of the corresponding optimal KCA solution plus $w_N(r)$.

By generalizing the transformation for the unrooted EKCT problem we can furthermore transform any given unrooted NKCT instance (G, w_N, k) or AKCT instance (G, w_E, w_N, k) into a corresponding KCA instance $(\bar{G}_r, r, c, k+1)$ as follows:

As before let $\bar{G}_r = (V \cup \{r\}, A \cup A_r)$. For all $(i,j) \in A \cup A_r$ we have $c((i,j)) := w_N(j)$ for the NKCT problem, and $c((i,j)) := w_E(\{i,j\}) + w_N(j)$ (or $c((i,j)) := w_N(j)$ if $i = r$), for the AKCT problem. By this transformation, we implicitly choose $w_N(r) := 0$ which is correct since the root node is artificial and should not be part of the objective function. Again, we have to guarantee that the corresponding

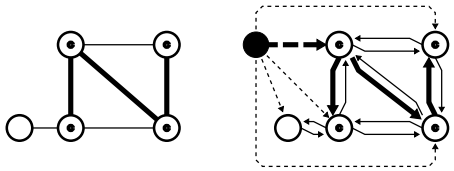


Figure 5.1: An undirected unrooted KCT instance (left), and its directed KCA counterpart (right). A possible solution for $k = 3$ is denoted by bold edges/arcs.

KCA solution has only one outgoing arc from the artificial root node. This is done analogously to the EKCT problem by imposing (P4).

K(PC)ST. The above ideas can be directly used to search feasible orientations of the K(PC)ST solutions in $\bar{G}_r$. We can integrate the node prizes into the arc costs as discussed for the AKCT problem above, drop the cardinality requirement on the arcs, and apply a node cardinality requirement analogous to (P2) only to the terminal nodes. In order to guarantee that the solution forms a tree, we have to require that every node in the solution, except for the root, has exactly one incoming arc (cf. Observation 5.5 and constraint (5.6) below).

5.3 ILP Modeling

We now model the KCA problem as an ILP based on directed cuts. Given an instance $(D = (V_D, A_D), r_D, c, k')$ of the KCA problem, we define $V_D^* = V_D \setminus \{r_D\}$ and $A_D^* = A \setminus \{(i, r_D) \mid i \in V_D^*\}$. We introduce two sets of binary variables $x \in \{0,1\}^{|A_D^*|}$ and $y \in \{0,1\}^{|V_D^*|}$. As usual, the variables are 1, if the corresponding arc or node is in the solution and 0 otherwise. The feasible k'-arborescences can then be described by the following constraint system. Recall that, given a variable vector ξ and a set of indices $\mathcal{J}$, we use the shorthand $\xi(\mathcal{J}) := \sum_{j \in \mathcal{J}} \xi_j$.

$$\text{KCA-DCUT} := \{(5.1)\text{–}(5.4)\}$$

with

$$x(A_D^*) = k' \tag{5.1}$$

$$y(V_D^*) = k' \tag{5.2}$$

$$x(\delta^-(S)) \geq y_v \qquad \forall S \subseteq V_D^*, \forall v \in S \tag{5.3}$$

$$x_a, y_v \in \{0,1\} \qquad \forall a \in A_D^*, v \in V_D^* \tag{5.4}$$

The *dcut-constraints* (5.3) guarantee property (P3) via directed cuts. Furthermore, they ensure that the root node r_D is always part of the solution, as for at least one node $v \in V_D^*$, the arc $(r_D, v) \in A_D^*$ must be chosen. This property and the inequalities (5.1) and (5.2) ensure the requirements (P1) and (P2), respectively.

Hence, the following ILP solves the KCA problem:

$$\min \left\{ \sum_{a \in A_D^*} c(a) \cdot x_a : (x, y) \text{ satisfies KCA-DCUT} \right\}. \tag{5.5}$$

Although this formulation is sound and practically applicable, we will reformulate the ILP based on Observation 5.5: Considering the system KCA-DCUT, we can replace the node cardinality constraint (5.2) by in-degree constraints

$$x(\delta^-(v)) = y_v \qquad \forall v \in V_D^*. \tag{5.6}$$

While the original formulation is more compact, we will see in Chapter 6 that the in-degree constraints have certain advantages in practice. The most important property of this reformulation is that it retains the strength of the original formulation. From the polyhedral point of view, we can measure this strength by considering the LP relaxations of both ILPs, i.e., we replace the integer requirements for the vector (x, y) by

$$0 \leq x_a, y_v \leq 1 \qquad \forall a \in A_D^*, \forall v \in V_D^*, \tag{5.7}$$

and thereby allow fractional solutions, cf. Definition 2.18 in Section 2.3.3. We can show that in this context it is irrelevant whether we model (P2) directly via a cardinality constraint or indirectly via multiple in-degree constraints.

Lemma 5.6. *By replacing the node cardinality constraint* (5.2) *with the in-degree constraints* (5.6) *we obtain an equivalent ILP with an equivalent LP relaxation.*

Proof. We show this equivalence by generating the constraints from each other. We have (5.6)⇒(5.2) as:

$$y(V_D^*) = \sum_{v \in V_D^*} y_v \overset{(5.6)}{=} \sum_{v \in V_D^*} x(\delta^-(v)) = x(A_D^*) \overset{(5.1)}{=} k'.$$

And we show (5.2)⇒(5.6) by:

$$\begin{aligned} k' \overset{(5.2)}{=} y(V_D^*) = \sum_{v \in V_D^*} y_v \overset{(5.3)}{\leq} \sum_{v \in V_D^*} x(\delta^-(v)) = x(A_D^*) \overset{(5.1)}{=} k' \\ \implies y_v = x(\delta^-(v)) \quad \forall v \in V_D^*. \end{aligned}$$

□

The above ILP models the general KCA problem. Using the transformations described in Section 5.2, we can hence solve the rooted KCT problems by applying this ILP. To be able to also consider the unrooted problem variants, it remains to model the requirement (P4). This can be straightforwardly achieved by demanding

$$x(\delta^+(r_D)) = 1. \tag{5.8}$$

As most KCT benchmark instances in the literature are unrooted, we will in the following concentrate on these types of problems. We therefore explicitly define the KCA constraint system with respect to a transformed instance $(\bar{G}_r, r, c, k+1)$ where (P4) is required:

$$\text{KCA}'\text{-DCUT} := \{(5.9)\text{–}(5.12)\},$$

with

$$x(A) = k \tag{5.9}$$
$$y(V) = k + 1 \tag{5.10}$$
$$x(A_r) = 1 \tag{5.11}$$
$$x(\delta^-(S)) \geq y_v \qquad \forall S \subseteq V, \forall v \in S \tag{5.12}$$
$$x_a, y_v \in \{0, 1\} \qquad \forall a \in A \cup A_r, v \in V \tag{5.13}$$

where we can substitute (5.10) by

$$x(\delta^-(v)) = y_v \qquad \forall v \in V \tag{5.14}$$

according to Lemma 5.6.

5.4 Polyhedral Comparisons

We investigate the polyhedral properties of our new formulation by comparing it to the other known ILP formulations for the KCT problems. In particular, we compare the relative strengths of their LP relaxations, cf. Section 2.3.3. In the following, let $\mathcal{P}_D$ be the polyhedron corresponding to the LP relaxation of Kca′-DCut:

$$\mathcal{P}_D := \{(x, y) \in [0, 1]^{|A \cup A_r| + |V|}, \text{ subject to (5.9)–(5.12)}\}$$

Hence, the feasible integer points described by Kca′-DCut are a proper subset of $\mathcal{P}_D$ and optimizing any objective function over $\mathcal{P}_D$ leads to a lower bound for the respective optimal solution of Kca′-DCut.

5.4.1 Generalized Subtour Elimination

We first compare the ILPs that use Kca′-DCut to those that use Kct-Gsec (cf. Section 4.5.1). We show that for all KCT problems both approaches are equivalent from the polyhedral point of view. Therefore, due to Definition 2.18 we compare the strength of the corresponding LP relaxations.

Recall that we use the node selection variables y equivalently in both formulations. Let $\mathcal{P}_G$ be the polyhedron corresponding to the LP relaxation of Kct-Gsec, i.e.,

$$\mathcal{P}_G := \{(z, y) \in [0, 1]^{|E| + |V|}, \text{ subject to (4.1)–(4.3)}\}.$$

To be able to compare both polyhedra we consider the following projection of $\mathcal{P}_D$ into the space of (z, y) variables:

$$\text{proj}_{z,y}(\mathcal{P}_D) := \{(z, y) \mid (x, y) \in \mathcal{P}_D, \forall \{i, j\} \in E : z_{\{i,j\}} = x_{ij} + x_{ji}\}.$$

Theorem 5.7. *For the unrooted KCT problems, the ILPs based on* Kct-Gsec *and* Kca′-DCut *are equivalent for all cost functions.*

Proof. According to Observation 2.17 we have to show that $\mathcal{P}_G = \text{proj}_{z,y}(\mathcal{P}_D)$. We prove this equality by showing mutual inclusion:

$\text{proj}_{z,y}(\mathcal{P}_D) \subseteq \mathcal{P}_G$**:** Any $(\bar{z}, \bar{y}) \in \text{proj}_{z,y}(\mathcal{P}_D)$ satisfies (4.1) by definition, and (4.2) by (5.14) and Lemma 5.6. Let $\bar{x}$ be the vector from which we projected the vector $\bar{z}$, and consider some $S \subseteq V$ with $|S| \geq 2$ and some node $t \in S$. We show that $(\bar{z}, \bar{y})$ also satisfies the corresponding GSEC (4.3):

$$\begin{aligned}\bar{z}(E(S)) = \bar{x}(A(S)) &= \textstyle\sum_{v \in S} \bar{x}(\delta^-(v)) - \bar{x}(\delta^-(S)) \\ &\overset{(5.14)}{=} \bar{y}(S) - \bar{x}(\delta^-(S)) \overset{(5.12)}{\leq} \bar{y}(S) - \bar{y}_t.\end{aligned}$$

$\mathcal{P}_G \subseteq \text{proj}_{z,y}(\mathcal{P}_D)$**:** Consider any $(\bar{z}, \bar{y}) \in \mathcal{P}_G$ and a set

$$\begin{aligned}X := \{ \quad & x \in \mathbb{R}_{\geq 0}^{|A \cup A_r|} \mid x \text{ satisfies (5.11)} \\ & \text{and } x_{ij} + x_{ji} = \bar{z}_{\{ij\}} \ \forall (i,j) \in A \quad \}.\end{aligned}$$

Every such projective vector $\bar{x} \in X$ clearly satisfies (5.9). In order to generate the dcut-inequalities (5.12) for the corresponding $(\bar{x}, \bar{y})$, it is sufficient to show that we can always find an $\hat{x} \in X$, which together with $\bar{y}$ satisfies the in-degree constraints (5.14). Since then, for any $S \subseteq V$ and $t \in S$:

$$\begin{aligned}\hat{x}(\delta^-(S)) = \textstyle\sum_{v \in S} \hat{x}(\delta^-(v)) - \hat{x}(A(S)) \\ \overset{(5.14)}{=} \bar{y}(S) - \bar{z}(E(S)) \overset{(4.3)}{\geq} \bar{y}_t.\end{aligned}$$

We show the existence of such an $\hat{x}$ using a proof technique related to [GM93, proof of Claim 2], where it was used for the Steiner tree problem.

An $\hat{x} \in X$ satisfying (5.14) can be interpreted as the set of feasible flows in a bipartite transportation network (N, L), with $N := (E \cup \{r\}) \cup V$. For each undirected edge $e = (u, w) \in E$ in G, our network contains exactly two outgoing arcs $(e, u), (e, w) \in L$. Furthermore, L contains all arcs of A_r. For all nodes $e \in E$ in N we define a supply $s(e) := \bar{z}_e$; for the root r we set $s(r) := 1$. For all nodes $v \in V$ in N we define a demand $d(v) := \bar{y}_v$.

Finding a feasible flow for this network can be viewed as a capacitated transportation problem on a complete bipartite network with capacities either zero (if the corresponding edge does not exist in L) or infinity. Note that in our network the sum of all supplies is equal to the sum of all demands, due to (4.1) and (4.2). Hence, each feasible flow in such a network will lead to a feasible $\hat{x} \in X$. Such a flow exists if and only if for every set $M \subseteq N$ without arcs leaving M (i.e., $\delta^+_{(N,L)}(M) = \emptyset$) the condition

$$s(M) \leq d(M) \tag{5.15}$$

is satisfied, where $s(M)$ and $d(M)$ are the total supply and the total demand in M, respectively, cf. [Gal57, GM93]. In order to show that this condition holds for (N, L), we distinguish between two cases; let $U := E \cap M$:

$r \in M$: Since r has an outgoing arc for every $v \in V$ and $\delta^+_{(N,L)}(M) = \emptyset$, we have $V \subset M$. Condition (5.15) is satisfied, since $s(r) = 1$ and therefore:

$$\begin{aligned} s(M) &= s(r) + \bar{z}(U) \leq s(r) + \bar{z}(E) \\ &= \bar{z}(E) + 1 \overset{(4.1),(4.2)}{=} \bar{y}(V) = d(M). \end{aligned}$$

$r \notin M$: Let $S := V \cap M$. We then have $U \subseteq E(S)$. If $|S| \leq 1$ we have $U = \emptyset$ and therefore (5.15) is automatically satisfied. For $|S| \geq 2$, the condition is also satisfied, since for every $t \in S$ we have:

$$s(M) = \bar{z}(U) \leq \bar{z}(E(S)) \overset{(4.3)}{\leq} \bar{y}(S) - \bar{y}_t \leq \bar{y}(S) = d(M). \qquad \square$$

Hence, we know that, independent of the specific objective function, the value of the optimal solution in $\mathcal{P}_D$ will be identical to the one in $\mathcal{P}_G$. In particular, this therefore holds for any unrooted KCT problem variant. For a rooted KCT problem, the corresponding Kct-Gsec ILP can be extended by $y_{r'} = 1$ for some root node $r' \in V$. Applying the above proof techniques we can deduce that:

Theorem 5.8. *For the rooted KCT problems, the* Kct-Gsec *ILPs, extended for rooted KCT problems as described above, are equivalent to the corresponding* Kca-DCut *ILPs.*

Even though Kca$^{(\prime)}$-DCut- and Kct-Gsec-based formulations are polytope-wise equivalent, Kca$^{(\prime)}$-DCut offers advantages in practice, as we will discuss in Chapter 6.

5.4.2 Undirected Cuts

In this section, we compare the Kca$'$-DCut ILP to those based on undirected cuts. Recall that the rKct-UCut ILP given in Section 4.5.2 has to be applied $|V|$ times in order to solve unrooted KCT problems. We suggest two further approaches based on undirected cuts for such unrooted variants.

Kct$'$-UCut: In contrast to the formulation by Garg, we can augment G with an artificial root node $r \notin V$ of weight 0 that is connected to all other nodes with edges of weight 0, thus obtaining the graph G_r. We can consider the system rKct-UCut for the instance $(G_r, r, [w_E], [w_N], k+1)$ and additionally ensure that only one of the new edges is selected, analogously to (5.11). Thereby, we obtain a constraint system Kct$'$-UCut that directly describes all KCT solutions via undirected cuts, instead of requiring linearly many ILPs.

Kct-UCut: Alternatively, we can obtain a system Kct-UCut by substituting the cut-inequalities (4.8) in rKct-UCut with

$$z(\delta(S)) \geq y_u + y_v - 1 \qquad \forall S \subset V, \forall u \in S, \forall v \in V \setminus S. \tag{5.16}$$

and applying it to the original instance $(G, [w_E], [w_N], k)$.

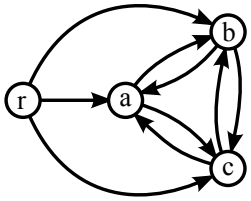


Figure 5.2: Graph $\bar{G}_r$ for the proof of Lemma 5.10.

Observation 5.9. *For the EKCT problem, if the input edge-weights are strictly positive, we can omit the edge-cardinality constraint $z(E) = k$ in all above undirected cut formulations without losing feasibility of their solutions. However, the ILPs containing $z(E) = k$ are strictly stronger than those without this constraint.*

Proof. The first part of the observation is trivial as the number of selected edges is minimized. It remains to show the advantage of the edge-cardinality constraint. Consider a triangle graph with unit edge weights as an input graph and $k = 2$, cf. Figure 5.3(a). An optimal fractional solution is to set all edge variables to 0.5. If the formulation requires an artificial node, we choose any of its adjacent edges to have an associated variable value of 1. Due to the node cardinality constraint all node variables have to be 1. The shown solution is feasible for all above undirected cut formulations if we omit the edge-cardinality constraint. The value of this (optimal) solution is thereby 1,5. However, requiring $z(E) = k$ leads to an optimal solution of value 2, thus strengthening the above formulations. □

For the comparison of our KCA′-DCUT ILP to above undirected cut formulations we will need the following technical lemma:

Lemma 5.10. *Let Δ be a triangle graph on the nodes $\{a, b, c\}$ and $\bar{G}_r$ its corresponding bidirection extended with an artificial root node r. For each variable setting $(\bar{x}, \bar{y}) \in [0,1]^{|A \cup A_r|} \times [0,1]^{|V|}$ in $\bar{G}_r$ with $\bar{y} = (1,1,1)$ that satisfies constraints* (5.11) *and* (5.12)*, we have $\bar{x}(A) \geq 2$.*

Proof. Figure 5.2 illustrates the graph $\bar{G}_r$. Since $\bar{x}(\delta^-(a)) \geq 1$, we have

$$\bar{x}_{(b,a)} + \bar{x}_{(c,a)} \geq 1 - \bar{x}_{(ra)}.$$

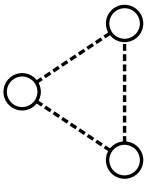
Analogously we have

$$\bar{x}_{(a,b)} + \bar{x}_{(c,b)} \geq 1 - \bar{x}_{(rb)},$$
$$\bar{x}_{(a,c)} + \bar{x}_{(b,c)} \geq 1 - \bar{x}_{(rc)}.$$

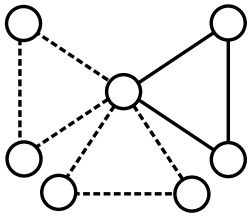
Since $\bar{x}(\delta^+(r)) = 1$, summing up over these three inequalities we obtain

$$\bar{x}(A) \geq 2.$$ □

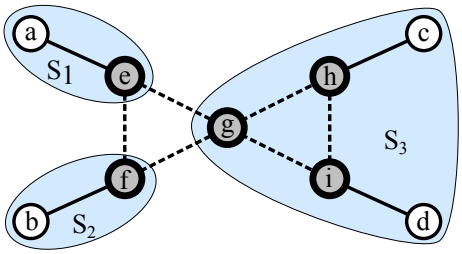
Let $\mathcal{P}_{U'}$ and $\mathcal{P}_U$ be the polyhedra of the LP relaxations of KCT′-UCUT and KCT-UCUT, respectively, and let $\mathcal{P}_{\cup r'}$ be the union of all LP relaxation polyhedra considered by RKCT-UCUT for all choices of the root r'. Reusing the projection $\text{proj}_{z,y}(\mathcal{P}_D)$ defined above (cf. page 41) we can show the following theorem.



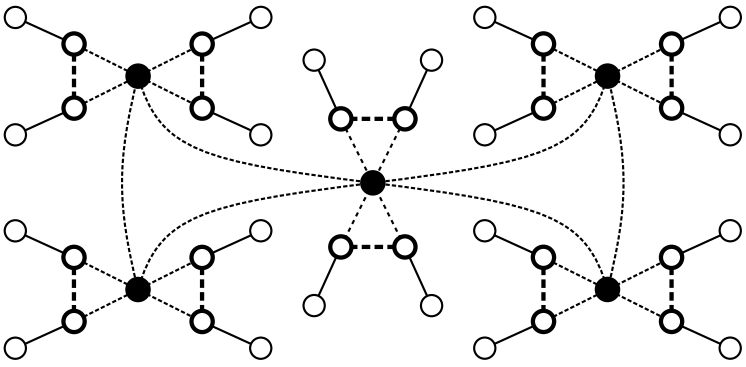
(a) EKCT, $k = 2$



(b) EKCT, $k = 6$



(c) NKCT, $k = 7$



(d) NKCT, $k = 42$

Figure 5.3: Solution examples for the undirected cut formulation used for the proofs of Observation 5.9 and Theorem 5.11. Circles with bold border represent expensive nodes. Solid edges have variable values set to 1, dashed edges to 0.5, bold dashed edges to 9/10. White, grey, black nodes have variable values of 1, 4/5, 3/5, respectively.

Theorem 5.11. *The* KCA′-DCUT *ILPs are strictly stronger than* KCT-UCUT, KCT′-UCUT, *and the union over all suitable* RKCT-UCUT *ILPs for all considered KCT problems, i.e., EKCT, NKCT and AKCT. This holds even when adding the (strengthening)* consistency constraints

$$y_v \geq z_{uv} \qquad \forall \{u, v\} \in E \tag{5.17}$$

to the undirected formulations.

Proof. Due to Definition 2.18 and Observation 2.17, we formally have to show that for any $\mathcal{P} \in \{\mathcal{P}_{U'}, \mathcal{P}_U, \mathcal{P}_{\cup r'}\}$ we have:

(a) $\text{proj}_{z,y}(\mathcal{P}_D) \subseteq \mathcal{P}$, and

(b) let $\text{obj}(z, y)$ be the objective function according to EKCT, NKCT, or AKCT; there are instances where KCA′-DCUT leads to strictly stronger lower bounds, i.e., $\min\{\text{obj}(z, y) \mid (z, y) \in \mathcal{P}\} \lneq \min\{\text{obj}(z, y) \mid (z, y) \in \text{proj}_{z,y}(\mathcal{P}_D)\}$.

Clearly, each feasible point $(\bar{z}, \bar{y}) \in \text{proj}_{z,y}(\mathcal{P}_D)$ is feasible for $\mathcal{P}_{\cup r'}$ and $\mathcal{P}_{U'}$. Furthermore, we show that any $(\bar{z}, \bar{y})$ also satisfies (5.16) and is therefore also feasible for $\mathcal{P}_U$. Let $\bar{x}$ be the vector from which we projected $\bar{z}$, and consider any $S \subset V$ and some nodes $u \in S$ and $v \in V \setminus S$. We then can deduce:

$$\begin{aligned}
\bar{y}_u + \bar{y}_v - 1 &\overset{(5.12)}{\leq} \bar{x}(\delta^-_{\bar{G}_r}(S)) + \bar{x}(\delta^-_{\bar{G}_r}(V \setminus S)) - 1 \\
&= \bar{x}(\delta^-_{\bar{G}}(S)) + \bar{x}(\delta^-_{\bar{G}}(V \setminus S)) + \bar{x}(A_r) - 1 \\
&\overset{(5.11)}{=} \bar{x}(\delta^-_{\bar{G}}(S)) + \bar{x}(\delta^-_{\bar{G}}(V \setminus S)) \overset{\text{proj}_{z,y}}{=} \bar{z}(\delta(S))
\end{aligned}$$

Hence it remains to show part (b), which also induces that the polyhedra are not equal. Generally, a common argument (in the context of STP and PCST) to pinpoint fractional solutions feasible for the LP relaxations of undirected but infeasible for directed formulations is based on a triangle graph with unit edge weights, cf. Figure 5.3(a) and Observation 5.9. When each edge variable is set to 0.5, the undirected cut between any two nodes is equal to 1. While this undirected solution satisfies all undirected cuts, there is no feasible directed solution that projects to some undirected solution of equally low objective value, cf. Lemma 5.10. Even more, the optimal directed fractional solution has the value of twice the (uniform) edge costs.

Yet, this simple example cannot be used in our context: As already discussed in the proof for Observation 5.9, the solution of value 1.5 is infeasible even for the undirected formulation, as we require the sum over all edge variables to be k. Nonetheless we will use such triangles as gadgets in our constructions.

EKCT. Consider three triangles with one common node, cf. Figure 5.3(b). The edges of one of the triangles are *cheap* (of weight α, say 0.1), all other edges are *expensive* (of weight β, say 1). Assume $k = 6$, which induces that all nodes variables are set to 1. A possible feasible solution to any above undirected

cut formulation is to set the variables of the expensive edges to 0.5 and those of the cheap edges to 1. If the formulation requires an artificial node, we choose any of its adjacent edges to have an associated variable value of 1. It is easy to see that all undirected cut constraints, as well as the consistency constraints (5.17), are satisfied. Such a solution has the objective value of $3\alpha + 3\beta$.

However, we know from Lemma 5.10 that a triangle does not allow a directed solution of cost less than its two cheapest edges. Hence, since our triangles are attached to each other via a single cut node, we can deduce that each of our triangles requires us to pay for at least two of its edges (again also summing up to $k = 6$). The optimal fractional directed solution will therefore cost $2\alpha + 4\beta \gneq 3\alpha + 3\beta$.

NKCT. Consider the graph in Figure 5.3(c). We have *cheap* nodes a–d (of weight α', say 0.1) and *expensive* nodes e–i (of weight β', say 1). We call the edges that are adjacent with nodes a–d cheap edges. Assume $k = 7$. An optimal solution $(\bar{z}, \bar{y})$ of an undirected cut formulation is to set $\bar{y}_v = 1$ if v is cheap and $\bar{y}_v = \frac{4}{5}$ otherwise. Furthermore, we set $\bar{z}_e = 1$ if e is adjacent to a cheap node, and $\bar{z}_e = 0.5$ otherwise. It is easy to see that $(\bar{z}, \bar{y})$ satisfies all constraints of any above undirected formulation (disregarding the consistency constraints). The cost of this solution is then $4\alpha' + 4\beta'$.

We show that all possible directed solutions for the above example have a strictly larger cost. Assume there would be a directed solution of cost $4\alpha'+4\beta'$, then the node variables of the four cheap nodes a–d have to be set to 1. This would lead to the following contradictory consequences:

Consequence 1: For the cheap node a, the sum of the variables of its incoming arcs $(r, a), (e, a)$ has to be at least 1, i.e., $x_{(e,a)} \geq 1 - x_{(r,a)}$.
Consider the cut constraint for the set S consisting of all nodes except a and r. As S contains cheap nodes, the sum of the variables of its incoming arcs $\delta^-(S) = \delta^+(r) \cup \{(a, e)\} \setminus \{(r, a)\}$ has to be at least 1. Since $x(\delta^+(r)) = 1$, we have $x_{(a,e)} \geq x_{(r,a)}$.
Hence we can deduce that the arc variables of the cheap edge $\{a, e\}$ always sum up to $x_{(a,e)} + x_{(e,a)} \geq x_{(r,a)} + (1 - x_{(r,a)}) = 1$. Therefore, the sum σ of the arc variables of all four cheap edges has to be at least 4.

Consequence 2: Consider the node sets $S_1 = \{a, e\}$, $S_2 = \{b, f\}$, $S_3 = \{c, d, e, h, i\}$. Due to the variable values of nodes a-d, the dcut-constraints ensure that the sum of the arc variables entering S_i $(1 \leq i \leq 3)$ is at least 1. By contracting the three node sets into three (super)nodes, we obtain a triangle situation over the edges $\{e, g\}, \{f, g\}, \{e, f\}$ as discussed above: the sum of the arc variables on these three edges therefore has to be at least 2.
Analogously, we can show that the sum of the arc variables on the edges $\{g, h\}, \{h, i\}, \{i, g, \}$ is also at least 2. Hence, due to the edge-cardinality constraints, the sum σ of the arc variables of all four cheap edges can be at most 3.

Consistency constraints. Furthermore, we can strengthen any undirected cut formulation by adding constraints (5.17). Indeed, the above solution is not feasible for such an extended ILP. We therefore construct a slightly more involved example. Let us call the expensive degree 4 node in the above graph its *center node*. Consider five copies of this graph and connect their center nodes analogous to the connection scheme of the five expensive nodes in the original graph, i.e., the center nodes form two triangles with one common node (cf. Figure 5.3(d)). Let $k = 42$.

Using the directed formulation, we cannot obtain a solution including all cheap nodes: to include all of them, we would also have to select arcs for all cheap edges and (cf. above) select two arcs for each triangle. This would require 44 edges.

In contrast to this, we can give a solution to the undirected formulation which does include all cheap nodes and therefore gives a weaker bound than Kca′-DCut: Set all non-center node variables to 1 and all center node variables to 3/5. Furthermore, set all cheap edges to 1, the edges incident to a center node to 1/2, and the other 10 edges (connecting expensive non-center nodes) to 9/10. We observe that all constraints, including the consistency constraints (5.17) are satisfied.

AKCT. This follows from the fact that EKCT and NKCT are special cases of AKCT. □

Overall, we can deduce that the models based on Kca′-DCut are a better choice for an exact approach than those based on Kct-UCut, since their improved strength will in general lead to tighter bounds and fewer branches in branch-and-cut algorithms.

5.4.3 Multi-Commodity Flow

Another traditional approach for similar network design problems are multi-commodity flow formulations. See, e.g., [PD01a] for the Steiner tree and [LWP+06] for the prize-collecting Steiner tree problems. We use this concept to derive a multi-commodity-based ILP for our KCA problem. We thereby have the same set of binary variables x, y as for Kca-DCut. Additionally, we define a commodity from r to v for every node $v \in V$. For each node v we have to send y_v units of the corresponding commodity from the artificial root to v. Therefore we define continuous variables f_{ij}^v to model the flow of each such commodity on the arc $(i, j) \in A \cup A_r$.

The set of feasible k-arborescences for the KCA problem on $(\bar{G}_r, r, c, k+1)$ with additional condition (P4) is then given by:

$$\text{Kca}'\text{-Mcf} := \{(5.18)\text{—}(5.23)\}$$

with

$$x(A) = k \tag{5.18}$$

$$y(V) = k + 1 \tag{5.19}$$

$$x(A_r) = 1 \tag{5.20}$$

$$\sum_{j:(j,i)\in A\cup A_r} f^v_{ji} - \sum_{j:(i,j)\in A\cup A_r} f^v_{ij} = \begin{cases} y_v, & \text{if } i = v \\ 0, & \text{else} \end{cases} \quad \forall i, v \in V \tag{5.21}$$

$$0 \le f^v_{ij} \le x_{ij} \qquad \forall (i,j) \in A \cup A_r, \forall v \in V \tag{5.22}$$

$$x_{ij}, y_v \in \{0,1\} \qquad \forall (i,j) \in A \cup A_r, \forall v \in V \tag{5.23}$$

Minimizing the objective function $\sum_{a\in A\cup A_r} c(a) \cdot x_a$ over the set Kca′-Mcf gives us the optimal solution of our KCA problem.

Let $\mathcal{P}_\mathcal{F}$ be the polyhedron of the LP relaxation of the above ILP. To be able to compare the LP relaxations of Kca′-Mcf and Kca-DCut we consider the projection of $\mathcal{P}_F$ into the (x, y) variables space:

$$\text{proj}_{(x,y)}(\mathcal{P}_F) := \{(x, y) \mid (x, y, f) \in \mathcal{P}_F\}.$$

Theorem 5.12. *The ILPs based on* Kca′-Mcf *and* Kca′-DCut *are equivalent for all cost functions.*

Proof. Due to Definition 2.18 and Observation 2.17, we have to show

$$\text{proj}_{(x,y)}(\mathcal{P}_F) = \mathcal{P}_D.$$

The proof follows [Lju04], where it was given in the context of PCST:

Given $(\bar{x}, \bar{y}, \bar{f}) \in \mathcal{P}_F$, we show that $(\bar{x}, \bar{y})$ is also in $\mathcal{P}_D$. Therefore, we only have to show that $(\bar{x}, \bar{y})$ satisfies (5.12), since all other constraints in Kca-DCut are trivially satisfied. Assume that there is a subset $S \subseteq V$ and a node $v \in V$ for which the corresponding (5.12) is violated, i.e., $\bar{x}(\delta^-(S)) < \bar{y}_v$. We know from (5.21) that there is a flow of exactly $\bar{y}_v$ from r to v feasible w.r.t. the capacity vector $\bar{x}$. From the max-flow min-cut theorem (cf. Theorem 2.6) we know that the minimum cut separating r and v in the corresponding network is at least y_v. This contradicts our assumption.

On the other hand, let $(\bar{x}, \bar{y}) \in \mathcal{P}_D$. We show that there exists a flow $\bar{f}$ such that $(\bar{x}, \bar{y}, \bar{f}) \in \mathcal{P}_F$. We set $\bar{f}^v_{ij} := 0$ for all $(i, j) \in A \cup A_r$ if $y_v = 0$. Otherwise, we construct a directed graph G'_r from $\bar{G}_r$ by inserting an additional node v' for each $v \in V$ with $y_v \ge 0$ and connecting it to v by an arc (v, v'). The capacity of (v, v') is set to y_v. The capacity of all other arcs $a \in A \cup A_r$ is set to $\bar{x}_a$. We then set $\bar{f}^v_{ij}$ to the values of the corresponding maximum r, v'-flow in the above network. The flow value of f^v is then exactly y_v: Assume that $|f^v| \le y_v$, then we would know from the max-flow min-cut theorem that there is a subset $S \subset V$ with $\bar{x}(\delta^-(S)) < y_v$ thus violating (5.12). On the other hand, $|f^v|$ is restricted by the capacity (v, v') and therefore cannot exceed y_v. □

The Kca′-Mcf formulation requires only a polynomial number of variables and constraints and can be solved directly via branch-and-bound. However, the sheer

number of variables becomes a practical drawback of this approach. In addition to the x and y variables, we require $|V| \cdot (|A| + |A_r|)$ variables to model the flow. As we know from similar problems [LWP+06] and from our results for survivable network design (see next part of this thesis), this leads to poor performance of multi-commodity flows in practice, compared to directed-cut based approaches which allow efficient separation of their exponentially many constraints, cf. Section 6.3.

5.4.4 Alternative Orientation-based Models

Alternative orientation-based models for the unrooted KCT problems are given in [QCML10]: The authors transform a given instance $(G = (V, E), w_E, w_N, k)$ into a problem instance $(\bar{G}^*, r', c^*)$ where the task is to find a minimum spanning arborescence (MSA) rooted at r' and satisfying some additional constraints, see below.

Their transformation is slightly different from our method. Recall that for unrooted KCT problems, we use the bidirection $\bar{G} = (V, A)$ of G extended with one additional node. In [QCML10], $\bar{G}$ is extended with two nodes resulting in the graph $\bar{G}^* = (V^*, A^*)$, $V^* = V \cup \{r'\} \cup \{r\}$ with $r', r \notin V$, and $A^* = A \cup A_{r'} \cup A_r$ with $A_{r'} = \{(r', j) \mid j \in V\} \cup \{(r', r)\}$ and $A_r = \{(r, j) \mid j \in V\}$. The cost function c^* is defined as

$$c^*((i,j)) := \begin{cases} 0 & \text{if } (i,j) \in A_{r'}, \\ w_N(j) & \text{if } (i,j) \in A_r, \\ w_E(\{i,j\}) + w_N(j) & \text{if } (i,j) \in A. \end{cases} \tag{5.24}$$

Note that if we are dealing with an EKCT problem, we can naturally assume $w_N(j) := 0$ for all $j \in V$.

Each spanning arborescence T' in $\bar{G}^*$ that is rooted at r' and satisfies the following constraints can be transformed into a k-cardinality tree T in G of the same cost, and vice versa:

1. T' contains a directed path from r' to each node $v \in V \cup \{r\}$.
2. T' contains only one arc leaving r.
3. The arc (r', r) is the only arc in T' entering r.
4. For each node $v \in V$, the path $(r' \to v)$ either contains the arc (r', r) or consists of the single arc (r', v).
5. Deleting the nodes r', r and their incident arcs from T' leaves an arborescence containing exactly k arcs.

Such constrained spanning arborescences can be described by a set of inequalities in multiple ways. Quintão et al. consider the sets MSA′-DFLOW and MSA′-MTZ that use multi-commodity flows and Miller-Tucker-Zemlin subtour elimination constraints, respectively. Both sets use the network defining binary arc variables x'

which are set to 1 if the corresponding arc is in the solution, and 0 otherwise. Minimizing the objective function $\sum_{a \in A^*} c^*(a) x'_a$ over MSA′-DFLOW or MSA′-MTZ gives two further ILPs that can be used to solve KCT.

Similar to our directed flow formulation, the first formulation sends 1 unit of flow from r' to each other node in $\bar{G}^*$. Therefore, for each $v \in V \cup \{r\}$ and $(i,j) \in A^* \setminus \{(r', w) \mid w \notin v\}$ it uses continuous flow variables h^v_{ij} that give the amount of commodity v that is sent over (i,j) to get from r' to v.

$$\text{MSA′-DFLOW} := \{(5.25)\text{–}(5.31)\}$$

with

$$x'(A) = k \tag{5.25}$$
$$x'(\delta^+(r)) = 1 \tag{5.26}$$
$$x'(\delta^+(r')) = n - k \tag{5.27}$$
$$\sum_{v \in V \cup \{r\}} h^v_{r'r} = k + 2 \tag{5.28}$$
$$\sum_{j:(j,i) \in A^*} h^v_{ji} - \sum_{j:(i,j) \in A^*} h^v_{ij} = \begin{cases} -1, & \text{if } i = r' \\ 1, & \text{if } i = v \\ 0, & \text{else} \end{cases} \qquad \forall v \in V \cup \{r\} \tag{5.29}$$
$$0 \leq h^v_{ij} \leq x'_{ij} \qquad \forall (i,j) \in A^*, \forall v \in V \tag{5.30}$$
$$x'_{ij} \in \{0,1\} \qquad \forall (i,j) \in A^*, \forall v \in V \tag{5.31}$$

The second formulation uses level variables $u_v \in \mathbb{R}^+$ for all nodes $v \in V^*$ to establish the Miller-Tucker-Zemlin constraints (5.37). These constraints guarantee the resulting solutions to be cycle-free. For a $v \in V^*$, u_v indicates the number of arcs on the path $(r' \to v)$ in the corresponding feasible solution.

$$\text{MSA′-MTZ} := \{(5.32)\text{–}(5.41)\}$$

with

$$x'(A) = k \tag{5.32}$$
$$x'(\delta^+(r)) = 1 \tag{5.33}$$
$$x'(\delta^+(r')) = n - k \tag{5.34}$$
$$x'(\delta^-(v)) = 1 \qquad \forall v \in V \cup \{r\} \tag{5.35}$$
$$x'_{r'v} + x'_{vw} \leq 1 \qquad \forall (v,w) \in A \tag{5.36}$$
$$(k+3)x'_{vw} + u_v - u_w + (k+1)x'_{wv} \leq k + 2 \qquad \forall (v,w) \in A \tag{5.37}$$
$$(k+3)x'_{vw} + u_v - u_w \leq k + 2 \qquad \forall (v,w) \in A_{r'} \cup A_r \tag{5.38}$$
$$x'_{r'r} = 1 \tag{5.39}$$
$$u_{r'} = 0 \tag{5.40}$$
$$u_v \in \mathbb{R}^+ \qquad \forall v \in V^* \tag{5.41}$$

The computational results in [QCML10] show that the latter formulation gives much weaker lower bounds than the flow-based formulation. For the related Steiner tree problem, we know that most of the known ILPs, e.g., GSEC-, cut- and multi-commodity flow based formulations are strictly stronger than MTZ-SEC based ILPs [PD01b]. Hence, it seems natural that this is also the case in our setting. In fact, a proof can be devised analogously to the one given in [PD01b].

We now show that MSA′-DFLOW on $(\bar{G}^*, r', c^*)$ is equivalent to KCA′-MCF on $(\bar{G}_r, r, c)$. We will thereby use the fact that the graph $\bar{G}^*$ contains $\bar{G}_r$ as a subgraph and the cost function c^* is identical with the function c on $\bar{G}_r$. Hence, we can identify the variables x' and x on the arc set $A \cup A_r$.

Theorem 5.13. *The ILPs based on* KCA′-MCF *and* MSA′-DFLOW *are equivalent for all considered KCT problems. Yet,* KCA′-MCF *is more compact than* MSA′-DFLOW.

Proof. Let $\mathcal{P}_{F'}$ be the polyhedron containing the feasible solutions of the MSA′-DFLOW LP relaxation. We consider the following projection into the space of x variables:

$$\text{proj}_x(\mathcal{P}_F) = \{x \in [0,1]^{A \cup A_r} \mid (x, y, f) \in \mathcal{P}_F\},$$
$$\text{proj}_x(\mathcal{P}_{F'}) = \{x \in [0,1]^{A \cup A_r} \mid (x', h) \in \mathcal{P}_{F'}, \forall a \in A \cup A_r : x_a = x'_a\}$$

Let $\bar{x} \in \text{proj}_x(\mathcal{P}_F)$ be a feasile solution resulting from any $(\bar{x}, \bar{y}, \bar{f}) \in \mathcal{P}_F$. We show that it is also in $\text{proj}_x(\mathcal{P}_{F'})$, i.e., that we can construct a fractional solution $(\bar{x}', \bar{h}) \in \mathcal{P}_{F'}$ with $\bar{x}'_a := \bar{x}_a$ for each $a \in A \cup A_r$. Therefore, we first set $\bar{x}'_{r'r} := 1$ and $\bar{h}^r_{r'r} := 1$. For each $v \in V$, we set $\bar{x}'_{r'v} := 1 - y_v$ and $\bar{h}^v_{r'v} := 1 - y_v$. We also set $\bar{h}^v_{ij} := \bar{f}^v_{ij}$ for all $v \in V$, $(i,j) \in A \cup A_r$. The resulting solution $(\bar{x}', \bar{h})$ straightforwardly satisfies all MSA′-DFLOW constraints.

On the other hand, given a solution $(\bar{x}', \bar{h}) \in \mathcal{P}_{F'}$, we can construct a feasible $(\bar{x}, \bar{f}, \bar{y})$ with $\bar{x}_a = \bar{x}'_a$ for all $a \in A \cup A_r$: For each $v \in V$, we set $\bar{y}_v := 1 - \bar{h}^v_{r'v}$ and $\bar{f}^v_{ij} := \bar{h}^v_{ij}$ for all $(i,j) \in A \cup A_r$. We now show that $(\bar{x}, \bar{f}, \bar{y})$ satisfies all KCA′-MCF constraints.

As $0 \leq \bar{h}^v_{ij} \leq 1$ for all $(i,j) \in A \cup A'_r \cup A_r$, we also have $0 \leq \bar{y}_v, \bar{f}^v_{ij} \leq 1$ for all $v \in V$ and $(i,j) \in A \cup A_r$. Constraints (5.18) and (5.20) are straightforwardly satisfied, as variables $\bar{x}$ and $\bar{x}'$ are identical on the arc set $A \cup A_r$. Furthermore, since for all $i, v \in V \cup \{r\}$ we have

$$\sum_{j:(j,i)\in A^*} \bar{h}^v_{ji} = \sum_{j:(j,i)\in A\cup A_r} \bar{h}^v_{ji} + \bar{h}^v_{r'i} \quad \text{and} \quad \sum_{j:(i,j)\in A^*} \bar{h}^v_{ij} = \sum_{j:(i,j)\in A\cup A_r} \bar{h}^v_{ij},$$

the flow-conservation constraints (5.21) are also satisfied. It remains to show that $\bar{y}(V) = k + 1$. Since MSA′-DFLOW sends exactly 1 unit of flow for each node $v \in V \cup \{r\}$, using constraint (5.28) we have $\sum_{v\in V} \bar{h}^v_{r'v} = |V| - k - 1$. We can deduce:

$$y(V) = \sum_{v\in V}(1 - \bar{h}^v_{r'v}) = |V| - \sum_{v\in V} \bar{h}^v_{r'v} = k + 1.$$

Although the formulations MSA′-DFLOW and KCA′-MCF are equivalent and MSA′-DFLOW avoids the use of the node variables y_v, our DFLOW formulation is more

compact than MSA′-DFLOW. In fact, the latter formulation contains flow variables $h^v_{r'v}$ that play exactly the role of y_v variables, indicating to what extend the node v is taken into the solution. At the same time, MSA′-DFLOW has a larger amount of network defining variables than KCA′-MCF due to the larger input graph. □

Computational results show that it is very time-consuming to compute the LP relaxation of MSA′-DFLOW. On the other hand, although computing such relaxations for MSA′-MTZ is very fast, the corresponding branch-and-bound algorithm is also very time-consuming due to the weak lower-bounds. Quintao et al. then use the stronger MSA′-DFLOW formulation to derive Lagrangian lower and upper bounds, resulting in a heuristic algorithm whose solution quality can be estimated using the corresponding lower bounds. As mentioned before, computational results in Chapter 7 show that our exact branch-and-cut algorithm based on KCA-DCUT is orders of magnitudes faster than all algorithms proposed in [QCML10].

Chapter 6

Branch-and-Cut

The strongest known ILP models that can be used to solve unrooted KCT problems are Kct-Gsec, Kca′-Mcf and Kca′-DCut. As we have shown in Chapter 5, these formulations are equivalent from the polyhedral point of view. Nevertheless, we can expect our Kca′-DCut approach to have certain advantages in the practice. For the Kca′-Mcf ILP, these were discussed in the previous section. In the following, we outline the practical merits of Kca′-DCut in comparison to the Kct-Gsec model: The dcut-constraints are sparser than the GSECs, which usually leads to a faster optimization in practice. This conjecture was experimentally confirmed, e.g., in [LWP+06] for the related prize-collecting Steiner tree problem, where a directed cut formulation was compared to its GSECs-based counterpart. The former was both faster in overall running time and required 1–2 orders of magnitude fewer iterations. In Section 6.3 we will discuss the formal differences in the performances between the separation routines for dcut-constraints and GSECs.

Hence, we use the Kca′-DCut formulation to develop and implement a branch-and-cut algorithm for our KCT problems. For a general description of the branch-and-cut scheme see Section 2.3.4.

6.1 Additional Constraints

Although Kca′-DCut has practical advantages over Kct-Gsec, we will extend our Kca′-DCut formulation with some special GSECs. These constraints obviously do not strengthen the original formulation, but they may lead to better overall performance of our branch-and-cut routine. We will use such constraints for both initialization and separation steps of our algorithm.

Special GSECs. Consider the following set of constraints:

$$x_{ij} + x_{ji} \leq y_i \quad \forall i \in V, \forall \{i,j\} \in E \tag{6.1}$$

Intuitively, these constraints require for each selected arc that both incident nodes are selected as well and ensure a unique orientation for each edge. For this reason we may call them *orientation constraints*. They do not strengthen the Kca′-DCut formulation as they represent the GSECs for two-element sets $S = \{i,j\} \subset V$; from

the proof of Theorem 5.7, we know that these inequalities can be generated with the help of (5.6) and (5.12).

Our test sets, as we will describe in Section 7, also contain grid graphs. Such graphs have many 4-cycles, i.e., cycles that consist of 4 undirected edges. Since these cycles are easy to identify (cf. Section 6.3), the special GSECs that forbid the existence of such 4-cycles in the solution can be useful. Note that due to our transformation, all 4-cycles are bidirected. Let $\mathcal{C}_4$ be the set of all bidirected 4-cycles. A cycle $C \in \mathcal{C}_4$ then consists of 8 arcs and $V(C)$ gives the nodes on C. We call the following set of constraints *4-cycle GSECs*:

$$\sum_{a \in C} x_a \leq \sum_{i \in V(C) \setminus \{v\}} y_i \quad \forall C \in \mathcal{C}_4, \forall v \in V(C). \tag{6.2}$$

Asymmetry constraints. For each optimal solution $T = (V_T, E_T)$ of the originally unrooted KCT problem, there are $k+1$ corresponding solutions of the KCA′-DCUT ILP. For any node $v \in V_T$, the arborescence obtained by orienting T from v outwards and adding the arc (r, v) represents an optimal solution of the corresponding KCA problem.

Consider any proper $<$-relation defined on V. In order to exclude the above symmetric solutions, and thus reduce the search space, the following *asymmetry constraints* [Lju04] can be used:

$$x_{rj} \leq 1 - y_i \quad \forall i, j \in V, i < j. \tag{6.3}$$

These constraints ensure that for each KCA solution, the node adjacent to the root is the one with the smallest possible "index".

6.2 Initialization

For the initial partial LP we have to find a balance between too few and too many constraints. Too few constraints would require too many time-consuming calls to the separation routine and the LP-solver. On the other hand, too many constraints would lead to long LP-solver running times. Our algorithm starts with the root-out-degree constraint (5.11), the edge cardinality constraint (5.9), and all in-degree constraints (5.14). We prefer the in-degree constraints over the node cardinality constraint (5.10), as they strengthen the initial partial LP. As the proof of Lemma 5.6 shows, all in-degree constraints, together with the root-out-degree and the edge cardinality constraint already induce the node cardinality constraint, and it is therefore not necessary to consider the latter. On the other hand, in order to generate the in-degree constraints, we would require multiple dcut-constraints, which are not available in the initial partial LP.

For the same reason, we also add the orientation constraints to our initial partial LP. This speeds up the algorithm significantly, as these constraints do not have to be separated explicitly by the branch-and-cut algorithm. This was first observed in [Lju04] for the PCST and also confirmed by our own experiments.

We also tried the above asymmetry constraints (6.3) to reduce the search space. Anyhow, we will see in our experiments that the quadratic number of these constraints becomes a hindrance for large graphs and/or small k in practice.

6.3 Separation

The dcut-constraints (5.12) can be separated in polynomial time via the traditional maximum flow separation scheme: We consider $\bar{G}_r$ as a network and interpret the x-values of the current fractional LP solution as arc capacities. We compute the maximum flow in $\bar{G}_r$ from r to each $v \in V$, using the implementation of [CG97]. If the flow is less than y_v, we extract one or more of the induced minimum r, v-cuts (following the ideas of [KM98, LWP+06]) and add the corresponding constraints to our model: We can easily extract the cuts closest to the source (*front cut*) and to the sink (*back cut*). Let S_f be the set of all nodes reachable from the source via unsaturated paths. Similarly, let S_b be the set of nodes with unsaturated paths to the sink. We can identify these sets by simple BFS algorithms. The edges $\delta^+(S_f)$ leaving S_f form the front cut, the edges $\delta^-(S_b)$ entering S_b are the back cut. We can successively enlarge these two node sets by including the target nodes of the front cut to S_f and the source nodes of the back cut to S_b, and restarting the BFS searches. As long as both node sets do not coincide, we can extract so-called *nested cuts*. Indeed, using these additional cuts significantly speeds up the computation.

Recall that in a separation procedure, we may search for the most violated inequality of the current LP relaxation. In order to find the most violated inequality of the Kca′-DCut formulation, or to show that no such inequality exists, we construct the flow network only once and perform at most $|V|$ maximum flow calculations on it. This is a main reason why the Kca′-DCut formulation performs better than Kct-Gsec in practice: a single separation step for Kct-Gsec requires $2|V| - 2$ maximum flow calculations, as already shown in [FHJM94]. Furthermore, the corresponding flow network is not static over all those calculations, but has to be adapted prior to each call of the maximum flow algorithm.

Our test sets, as described in Chapter 7, also contain grid graphs. In such graphs, it is easy to detect and enumerate all 4-cycles by embedding the grids into the plane and traversing all faces except for the single large one. In any planar graph there is only a linear number of faces. For grids with n nodes we even know that there are at most $(\sqrt{n} - 1)^2 = (n + 1 - 2\sqrt{n})$ 4-cycles. Hence we can separate all constraints (6.2) in polynomial time by simply enumerating these cycles.

6.4 Upper Bounds

In the last decade, several heuristics and metaheuristics have been developed for the KCT problem. Traditional branch-and-cut algorithms allow to use such algorithms as *primal heuristics*, giving upper bounds that the branch-and-cut algorithm can use for bounding purposes when branching. The use of such heuristics is two-fold: (a) They can be used as start-heuristics, giving a good initial upper bound before starting the actual branch-and-cut algorithm, and (b) they can be run multiple

times during the exact algorithm, using the current fractional solutions as an additional input, or *hint*, in order to generate new and tighter upper bounds on the fly.

Let h be a primal bound obtained by such a heuristic. We can add this bound to our LP as

$$\sum_{a \in A} c(a) \cdot x_a \leq h - \Delta.$$

Here, $\Delta := \min\{c(a) - c(b) \mid c(a) > c(b),\ a, b \in A\}$ denotes the minimal difference between any two cost values. If the resulting ILP is found to be infeasible, we have a proof that h was optimal, i.e., the heuristic solution was optimal.

As our experiments reveal, our algorithm is already very successful without the use of any primal heuristic. Hence we compared our heuristic-less branch-and-cut algorithm (DC^-) with one using a *perfect heuristic*: a (hypothetical) algorithm that requires no running time and gives the optimal solution. We can simulate such a perfect heuristic by using the optimal solution obtained by a prior run of DC^-. We can then measure how long the algorithm takes to discover the infeasibility of the ILP. We call this algorithm variant DC^+. If the runtime performance of DC^- and DC^+ are similar, we can conclude that using any heuristic for bounding is not necessary. Unless specified otherwise, we always report on the DC^- algorithm, i.e., the branch-and-cut algorithm without using any heuristic for upper bounds.

Chapter 7

Experiments

We implemented our algorithm in C++ using CPLEX 9.0 and LEDA 5.0.1. The experiments were performed on a 2.33 GHz Intel Xeon 5140 in 32 bit mode. Although the machine offers two cores, each process was restricted to a single core and 2 GB RAM. In this chapter we report on the performance of our algorithm for the unrooted EKCT and NKCT problems.

7.1 EKCT Instances

For the EKCT we tested our algorithm on all instances of the KCTLIB [BB03] which consists of the following benchmark sets:

(BX) The instance set suggested in [BX00] contains 35 4-regular graphs with 25–1000 nodes. The value of k is fixed to 20.

(UBM) The set presented in [UBM04] consists of randomly generated 20-regular graphs with 500–5000 nodes. There are 10 graphs per instance size, with edge-weights uniformly distributed from the interval $[50, 500]$ for $|V| \leq 1000$ and $[100, 1000]$ for $V \geq 1500$. We use the notation $k_{\text{rel}} = X\%$ to denote that k is chosen to be $X\%$ of $n = |V|$. E.g., for a graph with 2500 nodes, $k_{\text{rel}} = 10\%$ means that we choose $k = 250$. Each instance of the benchmark set has to be solved for $k_{\text{rel}} = \{10\%, \ldots, 50\%\}$. Urosevic et al. [UBM04] presented a variable neighborhood decomposition search (VNDS), which is still the best known metaheuristic for this benchmark set. However, there are no published results on the behavior of VNDS on any other test instances.

(BB) The instance set presented in [BB05b] is divided into four subsets of dense, sparse, grid and 4-regular graphs, respectively, with different sizes of up to 2500 nodes. We use the notation $k_{\text{rel}} = X\%$ to denote that k is chosen to be $X\%$ of $n = |V|$. E.g., for a graph with 2500 nodes, $k_{\text{rel}} = 10\%$ means that we choose $k = 250$. The values for k are defined as for (UBM) by using $k_{\text{rel}} = \{10\%, \ldots, 90\%\}$[1], and additionally for $k = 2$ and $k = n - 2$. For more details on these instances see Table 7.9. The most successful known

[1] For the grid instances, the values k_{rel} differ slightly.

metaheuristics for (BB) are the hybrid evolutionary algorithm (HyEA) [Blu06] and the ant colony optimization algorithm (ACO) [BS04]. This benchmark set is the one most commonly used in publications.

Remark 7.1. The choices $k = 2$ and $k = n - 2$ for (BB) are rather insignificant for the analysis of general KCT algorithms. We can solve the former case optimally in $\mathcal{O}(|V||E|)$ by building BFS trees of depth 2 for all nodes, thereby enumerating all connected edge pairs. The latter can be solved to optimality by considering the graph $G \setminus \{v\}$—the graph G without the node v—for all $v \in V$. For each such graph we compute a minimum spanning tree and choose the minimum among them as the solution. Hence we require $\mathcal{O}(|E||V| \log |V|)$ time.

The results of [BB05b] have already shown that the (BX) instances are easy, which was also confirmed by our experiments. Our algorithm needed 0.85 seconds on average per instance to solve them to optimality; the median was 0.08 seconds.

Our computational experiments on (UBM) show that all instances can be solved to provable optimality. Except for 20 out of the 350 instances, all of them can even be solved in under two hours. The longer running instances are some large ones with 4000–5000 nodes which require 40–50% of edges to be in the solution.

Table 7.1 gives the average running times of our algorithm, as well as the running times of the VNDS metaheuristic in [UBM04]. Even though the latter times are larger, we can expect them to be substantially smaller on recent machines, as they where achieved on a Pentium II with 450 MHz. Unfortunately, this machine was too old to be considered in the current SPEC performance evaluation tests [Spe08], so we cannot fairly compare these running times. Rough estimates suggest a performance difference of about 150×. Overall, the VNDS metaheuristic by Urosevic et al. [UBM04] is clearly faster than our exact approach, but on the other hand, this is only achieved at the cost of the solution quality. VNDS did never give an optimal solution. Table 7.1 also shows the average differences between the optimal solutions and the previously best known solutions (BKS), obtained by VNDS. We can observe that the running times of the VNDS approach do not increase as strongly as our approach with increasing k. We also report that the gaps of VNDS usually decrease with higher k.

For the following analysis regarding the EKCT problem (Sections 7.2–7.4) we will concentrate on the more common and diverse benchmark set (BB), and compare our results to those of HyEA [Blu06], ACO [BS04], and the ILP-based algorithms of Quintao et al. [QCML10]. An additional advantage of this benchmark set is that the available data on the performance of these other algorithms is much more detailed. In Section 7.5 we will report on our results on some further instance sets that are not part of the KCTLIB.

7.2 Algorithmic Behavior

7.2.1 Parameter Influence

We start with investigating the relative merits of various possible parameter settings. As our base case we consider the above described branch-and-cut algorithm

	nds	500	1000	1500	2000	3000	4000	5000
avg	time	4.1/39.0	25.4/103	91.7/168	202/256	881/737	2853/1104	5919/1270
	gap	1.5%	0.1%	0.1%	0.2%	0.2%	0.3%	0.3%
$k_{\text{rel}}(\%)$	10	2.8/15.3	13.0/52.9	44.4/123	52.2/152	148/225	279/534	506/719
	20	2.6/46.9	14.4 /95.9	52.3/153	120/211	255/627	781/955	1296/1089
	30	4.8/37.4	27.5/157	98.0/221	256/242	584/1135	1631/1083	4126/1381
	40	4.8/48.2	41.4/174	184.3/219	400/323	1205/915	3803/1481	7770/1693
	50	8.0/47.25	65.1/34.7	372/123	652/352	2212/785	8812/1468	14853/1892
	*	5.6/47.25	30.7/34.7	79.6/123	180/352	—	—	—

Table 7.1: Average running times of DC$^-$ for (UBM) in seconds. "*" considers $k_{\text{rel}} = 50\%$ for $|V| \leq 2000$ and is achieved by using the asymmetry constraints (6.3). The times in small font after the slashes are the times of VNDS reported in [UBM04]. We also give the average gap between the optimum and the best solutions obtained by VNDS. Note that the times for VNDS were achieved on a much slower machine (cf. text).

without special constraints or heuristics.

Figure 7.1 illustrates the effectiveness of the asymmetry constraints (6.3) depending on increasing relative cardinality k_{rel}. We measured the speed-up by the quotient $t_\emptyset/t_{\text{asy}}$, where t_{asy} and $t_\emptyset$ denote the running time with and without using (6.3), respectively. The constraints allow a speed-up by more than an order of magnitude for sparse, dense and regular graphs, but only for large cardinality $k \geq \frac{n}{2}$. Our experiments show that for smaller k, a variable x_{ri}, for some $i \in V$, is quickly set to 1 and stays at this value until the final result. In these cases the constraints cannot help and only slow down the algorithm. Interestingly, the asymmetry constraints are never profitable for the grid instances. For graphs with more than 2000 nodes, using (6.3) is not possible due to memory restrictions, as the $\mathcal{O}(|V|^2)$ many asymmetry constraints are too much to handle. Hence, we omitted these graphs in Figure 7.1.

We can validate these observations by reconsidering (UBM). In Table 7.1 we see the improvement achieved by introducing the asymmetry constraints for $k_{\text{rel}} = 50\%$.

We also report on the experiments with the 4-cycle GSECs (6.2) within the separation routine for the grid graphs. The clear advantage of these constraints is shown in Figure 7.2, which shows the speed-up factor $t_\emptyset/t_{\text{gsec}}$ obtained by the use of these constraints.

Based on these results we apply a simple rule for all the remaining experiments: We include the asymmetry constraints for all non-grid instances with less than 2500 nodes and $k \geq \frac{n}{2}$. For the grid instances we always separate the 4-cycle GSECs (6.2).

As described in Section 6.4, we also investigate the influence of primal heuristics in our branch-and-cut algorithm. A comparison of the running times of DC$^+$ (using a *perfect* heuristic, cf. page 58) and DC$^-$ is shown in Figure 7.3 (thereby restricted to instances with 1000 nodes). In general, our experiments show that DC$^+$ is only 10–30% percent faster than DC$^-$ on average, even for the large graphs. Hence, we conclude that a bounding heuristic is not crucial for the success of our algorithm.

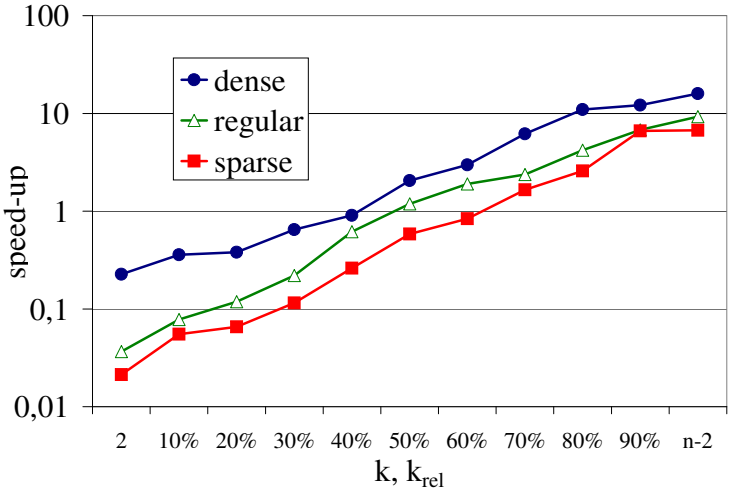


Figure 7.1: Speed-up factors for dense, regular and sparse graphs with $|V| \leq 2000$ obtained when asymmetry constraints (6.3) are included in the initial LP

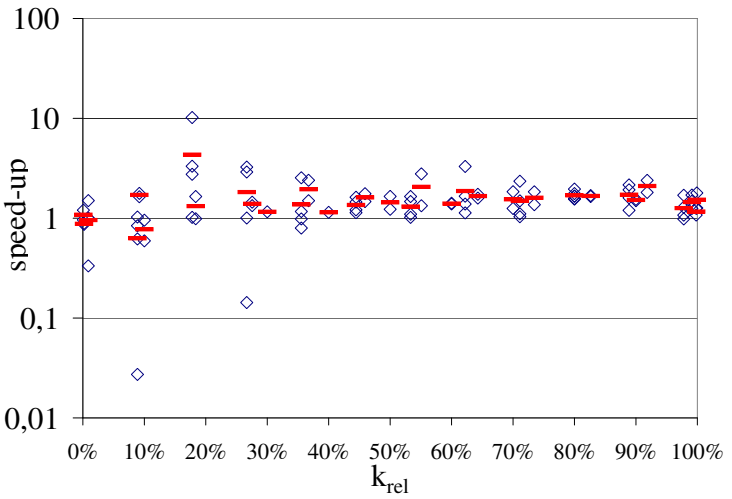


Figure 7.2: Speed-up factors for the grid instances of (BB) when 4-cycle GSECs (6.2) are separated. We have a diamond-shaped data point for each instance and k_{rel} value. This data point gives the corresponding speed-up factor achieved by the use of 4-cycle GSECs. The short horizontal bars denote the average speed-up over all instances per k_{rel}.

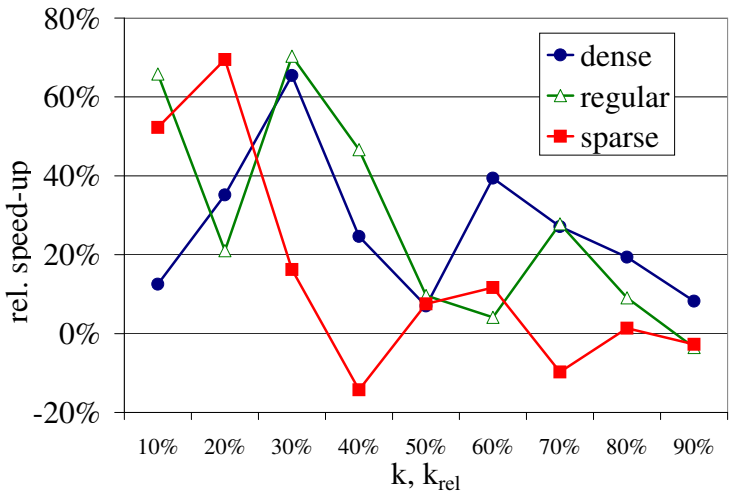


Figure 7.3: Relative speed-up $(t_{\mathrm{DC}^-} - t_{\mathrm{DC}^+})/t_{\mathrm{DC}^-}$ (in percent) of DC^+ compared to DC^- for the instances with 1000 nodes.

7.2.2 Algorithm's Running Times

The detailed running times of our algorithm for all (BB) instances are presented in Tables 7.2, 7.3, 7.4, 7.5, and 7.6 (the five columns on the right are discussed in Section 7.3).

In Table 7.7 we summarize the average and median computation times of our algorithm, sorted by size and categorized according to the special properties of the underlying graphs. Generally, we leave table cells empty if there is no problem instance with the according properties. We can observe that performance does not differ significantly between the sparse, regular and dense graphs, but that the grid instances are more difficult and require more computational power. This was also noticed in [EFHM97, BE03, BUM06].

In Table 7.8, we show that the computation time is not only dependent on the graph size, but also on the density of the graph. Therefore we give the average CPU time for the specified instance sets (all (BB) except the grid instances) with 500 and 1000 nodes and relative cardinalities k of 10%, 20%, ..., 50%.

The behavior of DC^- also has a dependency on k. For the sparse, dense, and regular instances the running time increases with increasing k for up to 50%, see Figures 7.4(a), 7.4(c) and 7.4(d). For larger k it remains relatively stable. In contrast to this, solving the grid instances (cf. Figure 7.4(b)) is more difficult for the relatively small k-values.

7.3 Comparison to Other Methods

7.3.1 Runtime Quality

The original experiments for HyEA and ACO were performed on an Intel Pentium IV, 3.06 GHz with 1GB RAM and a Pentium IV 2.4 GHz with 512MB RAM,

		time DC$^-$		time [QCML10]			obj.val.	
	k	**ILP**	**LP**	**MCF-LP**	**MTZ-ILP**	**LH**	**opt**	**LH**
bb15x15_1	2	0.03	0.03	—	—	—	2	—
	20	0.3	0.3	790.39	81.85	70.46	257	257
	40	25.16	1.23	2427.98	24425.45	90.71	642	642
	60	2.93	1.42	2230.31	153320.69	93.30	977	977
	80	1.01	1.01	915.18	328839.66	100.70	1335	1335
	100	1.01	0.84	—	—	—	1761	—
	120	0.7	0.64	—	—	—	2235	—
	140	0.5	0.5	—	—	—	2781	—
	160	0.93	0.73	—	—	—	3417	—
	180	0.56	0.56	—	—	—	4158	—
	200	0.57	0.57	—	—	—	5040	—
	220	0.66	0.66	—	—	—	6176	—
	223	0.67	0.67	—	—	—	6400	—
bb15x15_2	2	0.04	0.04	—	—	—	6	—
	20	148.39	0.35	809.50	273.16	81.48	253	253
	40	2.3	0.41	314.86	6508.41	86.24	585	585
	60	1.35	0.50	335.60	31317.61	87.03	927	927
	80	0.97	0.64	1163.30	36628.68	96.31	1290	1290
	100	0.67	0.67	—	—	—	1686	—
	120	0.54	0.54	—	—	—	2120	—
	140	0.46	0.46	—	—	—	2634	—
	160	0.51	0.51	—	—	—	3248	—
	180	0.59	0.54	—	—	—	3915	—
	200	0.67	0.67	—	—	—	4718	—
	220	0.99	0.99	—	—	—	5862	—
	223	0.9	0.9	—	—	—	6101	—
bb45x5_1	2	0.02	0.02	—	—	—	2	—
	20	8.24	0.23	390.89	1067.70	79.28	306	306
	40	10.12	2.49	669.35	53752.90	87.48	695	695
	60	7.76	2.63	55.67	OOM	94.58	1107	1107
	80	21.46	2.43	892.37	OOM	103.38	1551	1551
	100	6.54	2.57	—	—	—	1956	—
	120	1.58	1.58	—	—	—	2444	—
	140	1.4	1.4	—	—	—	3024	—
	160	1.05	1.05	—	—	—	3688	—
	180	0.92	0.92	—	—	—	4472	—
	200	0.6	0.6	—	—	—	5461	—
	220	0.9	0.9	—	—	—	6718	—
	223	0.84	0.84	—	—	—	6946	—
bb45x5_2	2	0.04	0.04	—	—	—	8	—
	20	6.17	0.13	412.14	732.20	80.26	302	302
	40	0.84	0.36	462.78	7658.12	86.23	654	654
	60	83.42	2.11	2903.91	OOM	95.39	1122	1122
	80	3.5	3.30	2379.05	OOM	101.85	1617	1617
	100	3.81	2.62	—	—	—	2129	—
	120	3.25	2.34	—	—	—	2631	—
	140	1.51	1.44	—	—	—	3174	—
	160	1.22	1.15	—	—	—	3757	—
	180	1.11	1.11	—	—	—	4458	—
	200	1.22	1.22	—	—	—	5262	—
	220	1.04	1.04	—	—	—	6347	—
	223	0.95	0.95	—	—	—	6568	—

Table 7.2: Times (in sec.) and solution values of DC$^-$ and the algorithms in [QCML10] for **grid** (BB) instances.

		time DC$^-$		time [QCML10]			obj.val.	
	k	**ILP**	**LP**	**MCF-LP**	**MTZ-ILP**	**LH**	**opt**	**LH**
bb33x33_1	2	0.15	0.15	—	—	—	3	—
	100	37.28	34.41	*	OOM	6633.15	1562	1577
	200	146.88	116.01	*	OOM	7079.54	3292	3342
	300	194.3	167.06	*	OOM	7755.70	5088	5207
	400	224.36	207.29	*	OOM	9011.41	7044	7172
	500	113.09	105.65	*	OOM	6710.19	9176	9332
	600	52.42	52.42	—	—	—	11552	—
	700	66.91	62.84	—	—	—	14270	—
	800	80.64	80.64	—	—	—	17363	—
	900	71.39	71.39	—	—	—	20911	—
	1000	76.98	76.98	—	—	—	25199	—
	1087	100.22	100.22	—	—	—	30417	—
bb33x33_2	2	0.16	0.16	—	—	—	3	—
	100	74.88	67.75	*	OOM	5548.97	1524	1547
	200	86.06	71.42	*	OOM	5797.90	3241	3379
	300	75.56	52.54	*	OOM	6096.48	5164	5296
	400	52.61	52.61	*	OOM	6259.84	7212	7339
	500	82.95	68.96	*	OOM	6578.05	9413	9510
	600	58.75	56.05	—	—	—	11791	—
	700	72.15	72.15	—	—	—	14461	—
	800	74.88	70.38	—	—	—	17502	—
	900	69.47	69.47	—	—	—	20989	—
	1000	62.77	62.77	—	—	—	25273	—
	1087	102.12	102.12	—	—	—	30326	—
bb50x50_1	2	0.34	0.34	—	—	—	2	—
	250	5851.12	1649.01	—	—	—	3885	—
	500	19513.1	11950	—	—	—	8059	—
	750	4741.98	4741.98	—	—	—	12433	—
	1000	2710.49	2679.91	—	—	—	17264	—
	1250	2224.24	2132.51	—	—	—	22558	—
	1500	1494.24	1477.69	—	—	—	28454	—
	1750	1644.35	1594.02	—	—	—	35331	—
	2000	1228.48	1228.48	—	—	—	43541	—
	2250	1667.75	1667.75	—	—	—	53406	—
	2498	1426.24	1426.24	—	—	—	67141	—
bb50x50_2	2	0.62	0.62	—	—	—	3	—
	250	957.98	443.54	—	—	—	3581	—
	500	6211.05	5483.35	—	—	—	7797	—
	750	4136.36	3742.99	—	—	—	12346	—
	1000	3208.90	3205.18	—	—	—	17445	—
	1250	2353.30	2335.24	—	—	—	23229	—
	1500	1369.41	1369.41	—	—	—	29697	—
	1750	1390.10	1390.10	—	—	—	36978	—
	2000	1955.45	1955.45	—	—	—	45592	—
	2250	2149.56	2149.56	—	—	—	56029	—
	2498	1991.96	1991.96	—	—	—	70439	—

Table 7.3: Times (in sec.) and solution values of DC$^-$ and the algorithms in [QCML10] for **grid** (BB) instances.

		time DC$^-$		time [QCML10]			obj.val.	
	k	**ILP**	**LP**	**MCF-LP**	**MTZ-ILP**	**LH**	**opt**	**LH**
steinc5	2	0.1	0.04	—	—	—	5	—
	50	1.48	0.43	8124.35	1597.12	465.31	772	772
	100	0.48	0.22	3359.5	2783.31	524.68	1712	1712
	150	0.55	0.55	*	4641.91	619.55	2865	2874
	200	0.88	0.88	17132.52	13695.98	641.64	4271	4271
	250	2.44	2.44	28724.53	20226.13	712.38	5942	5945
	300	3.44	2.73	—	—	—	7938	—
	350	1.79	1.79	—	—	—	10236	—
	400	1.67	1.67	—	—	—	12964	—
	450	1.57	1.57	—	—	—	16321	—
	498	1.86	1.86	—	—	—	20485	—
steind5	2	0.11	0.11	—	—	—	3	—
	100	3.01	1.36	*	27822.98	7741.05	1503	1503
	200	7.38	3.5	*	72865.63	8694.89	3440	3452
	300	5.64	4.35	*	OOM	7815.48	5817	5836
	400	7.01	7.01	*	OOM	8394.2	8685	8690
	500	32.75	32.75	*	OOM	9245.71	12054	12074
	600	23.82	23.82	—	—	—	15911	—
	700	24.93	24.02	—	—	—	20510	—
	800	18.32	18.32	—	—	—	26053	—
	900	15.92	15.92	—	—	—	32963	—
	998	16.09	16.09	—	—	—	41572	—
steine5	2	0.33	0.33	—	—	—	3	—
	250	9.52	7.69	—	—	—	3883	—
	500	29.8	25.26	—	—	—	9270	—
	750	96.69	60.16	—	—	—	15765	—
	1000	180.06	157.04	—	—	—	23495	—
	1250	245.54	245.54	—	—	—	32475	—
	1500	479.72	377.00	—	—	—	42735	—
	1750	741.97	582.38	—	—	—	54729	—
	2000	893.98	893.98	—	—	—	68618	—
	2250	2146.72	2224.48	—	—	—	85360	—
	2498	2224.48	2224.48	—	—	—	106677	—

Table 7.4: Times (in sec.) and solution values of DC$^-$ and the algorithms in [QCML10] for **sparse** (BB) instances.

	k	time DC$^-$ ILP	time DC$^-$ LP	time [QCML10] MCF-LP	time [QCML10] MTZ-ILP	time [QCML10] LH	obj.val. opt	obj.val. LH
le450_15a	2	12.64	0.41	—	—	—	2	—
	45	10.2	2.25		4982.81	7218.96	59	59
	90	1.97	1.97	—	—	—	135	—
	135	5.6	3.39		5501.04	14295.79	226	226
	180	7.37	7.37	—	—	—	336	—
	225	7.41	5.28		99599.15	16179.83	471	471
	270	4.44	4.44	—	—	—	630	—
	315	5.42	5.42	—	—	—	822	—
	360	5.35	5.35	—	—	—	1060	—
	405	7.91	7.91	*	OOM	13351.92	1388	1388
	448	6.02	6.02	—	—	—	2002	—
steinc15	2	0.16	0.16	—	—	—	2	—
	50	0.38	0.38		959.16	1427.78	208	208
	100	0.96	0.71		4577.04	1605.2	481	481
	150	3.19	2.16		5542.21	1804.98	802	802
	200	2.33	2.09		39944.7	2073.63	1182	1182
	250	4.9	4.52	*	OOM	2230.47	1625	1625
	300	6.45	4.18	—	—	—	2148	—
	350	3.24	3.24	—	—	—	2795	—
	400	6.49	6.49	—	—	—	3571	—
	450	4.87	4.87	—	—	—	4553	—
	498	3.58	3.58	—	—	—	5973	—
steind15	2	0.34	0.34	—	—	—	2	—
	100	14.28	3.09	*	OOM	20449.74	454	454
	200	11.58	11.12	*	OOM	22201.96	1018	1023
	300	15.6	15.6	*	OOM	22561.36	1674	1684
	400	41.69	38.70	*	OOM	22794.72	2446	2457
	500	19.62	19.62	*	OOM	26530.49	3365	3372
	600	25.49	25.49	—	—	—	4420	—
	700	23.56	23.56	—	—	—	5685	—
	800	21.6	21.6	—	—	—	7236	—
	900	24.57	24.57	—	—	—	9248	—
	998	21.42	21.42	—	—	—	12504	—

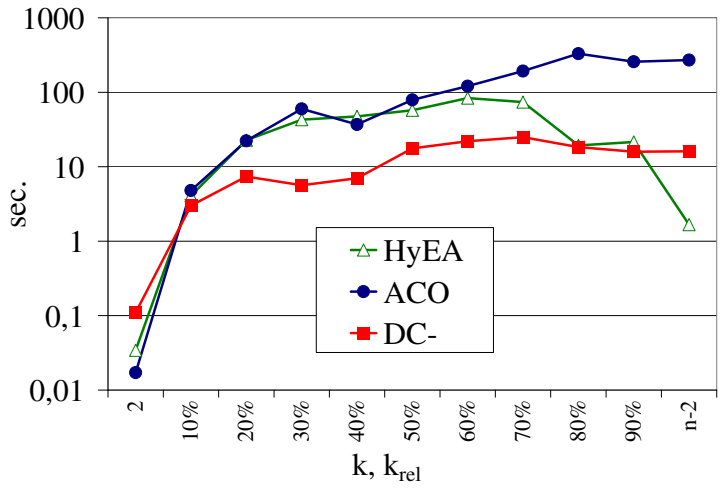
Table 7.5: Times (in sec.) and solution values of DC$^-$ and the algorithms in [QCML10] for **dense** (BB) instances.

		time DC$^-$		time [QCML10]			obj.val.	
	k	**ILP**	**LP**	**MCF-LP**	**MTZ-ILP**	**LH**	**opt**	**LH**
g1000-4-01	2	0.15	0.15	—	—	—	6	—
	100	6.83	2.08	*	*	9291.24	1523	1527
	200	3.37	3.11	*	215106.66	10174.36	3308	3308
	300	16.26	5.58	*	OOM	11794.51	5325	5328
	400	21.64	13.32	*	OOM	10157.27	7572	7590
	500	19.84	19.84	*	OOM	10761.9	10042	10079
	600	17.44	17.44	*	OOM	*	12705	*
	700	31.14	24.30	*	OOM	*	15675	*
	800	26.28	26.28	—	—	—	19015	—
	900	33.1	30.12	—	—	—	22827	—
	998	22.7	22.7	—	—	—	27946	—
g1000-4-05	2	0.16	0.16	—	—	—	7	—
	100	1.94	1.81	*	148251.04	9487.4	1648	1658
	200	2.31	2.31	*	OOM	10246.42	3618	3649
	300	6.76	5.89	*	OOM	10263.63	5797	5821
	400	14.69	13.93	*	OOM	9962.52	8195	8217
	500	21.88	20.96	*	OOM	10432.05	10784	10797
	600	27.79	14.14	*	OOM	*	13579	*
	700	10.74	10.74	*	OOM	*	16674	*
	800	10.78	10.78	—	—	—	20074	—
	900	22.3	22.3	—	—	—	24029	—
	998	18.67	18.67	—	—	—	29182	—
g400-4-01	2	0.05	0.05	—	—	—	8	—
	40	0.14	0.10	2281.68	155.5	406.4	563	563
	80	0.29	0.21	3656.95	979.42	465.76	1304	1304
	120	0.4	0.4	2871.22	3322.27	553.35	2132	2132
	160	1.14	0.93	12296.15	132412.57	526.29	3062	3065
	200	1.74	1.59	*	*	534.37	4086	4095
	240	1.1	1.1	—	—	—	5224	—
	280	2	1.52	—	—	—	6487	—
	320	0.96	0.96	—	—	—	7882	—
	360	0.91	0.91	—	—	—	9468	—
	398	0.84	0.84	—	—	—	11433	—
g400-4-05	2	0.05	0.05	—	—	—	4	—
	40	1.41	0.76	2905.24	1709.59	352.29	670	670
	80	0.9	0.55	2974.7	3774.46	407.92	1445	1445
	120	1.12	0.54	4626.19	6083.98	469.12	2291	2293
	160	0.65	0.65	4217.41	32882.25	494.77	3192	3192
	200	1.42	1.42	3651.64	13962.67	529.76	4156	4156
	240	1.2	1.2	—	—	—	5198	—
	280	3.54	3.09	—	—	—	6350	—
	320	1.33	1.16	—	—	—	7682	—
	360	0.97	0.97	—	—	—	9249	—
	398	0.84	0.84	—	—	—	11236	—

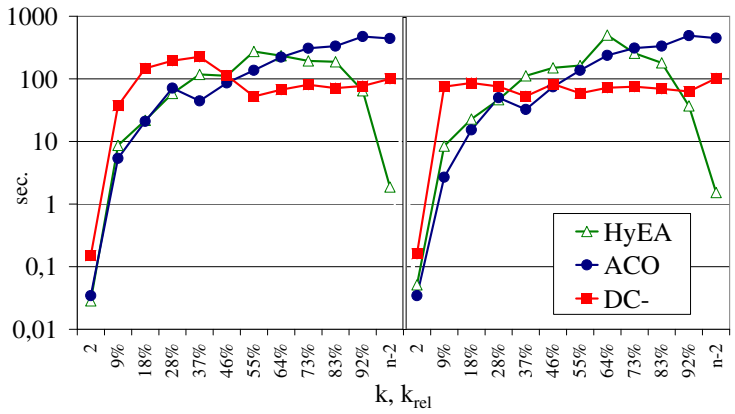
Table 7.6: Times (in sec.) and solution values of DC$^-$ and the algorithms in [QCML10] for **regular** (BB) instances.

# nodes	500		1000–1089		2500	
group	avg/med	$\frac{t_{\text{HyEA}}}{t_{\text{DC}-}}$	avg/med	$\frac{t_{\text{HyEA}}}{t_{\text{DC}-}}$	avg/med	$\frac{t_{\text{HyEA}}}{t_{\text{DC}-}}$
sparse	1.3/1.3	2.7	12.6/15.9	2.9	640.9/245.5	0.4
regular	1.0/1.0	3.9	15.3/16.9	7.5	—	—
dense	5.0/5.1	4.9	19.2/21.4	2.9	—	—
grid	7.0/1.0	0.6	82.4/74.9	1.7	3101.2/1973.7	0.2

Table 7.7: Average/median CPU time (in seconds) and the average speed-up factor of DC$^-$ to HyEA for the instance set (BB). Cells are left empty if there exists no instance matching the given criteria.

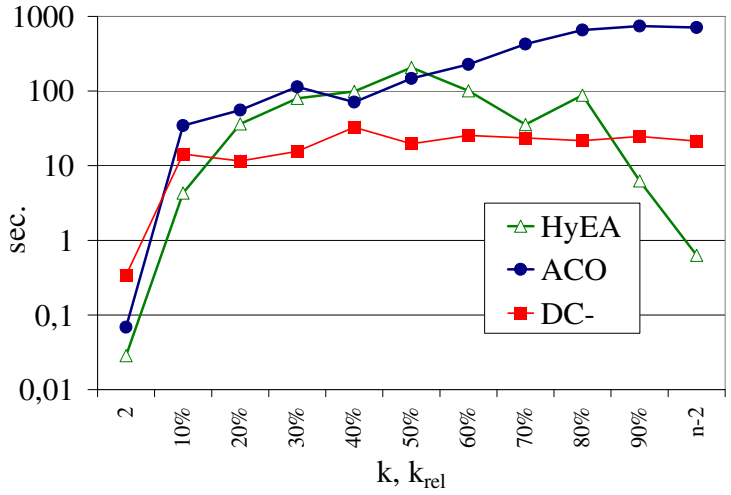


(a) sparse steind5 (1000 nodes, 1250 edges)

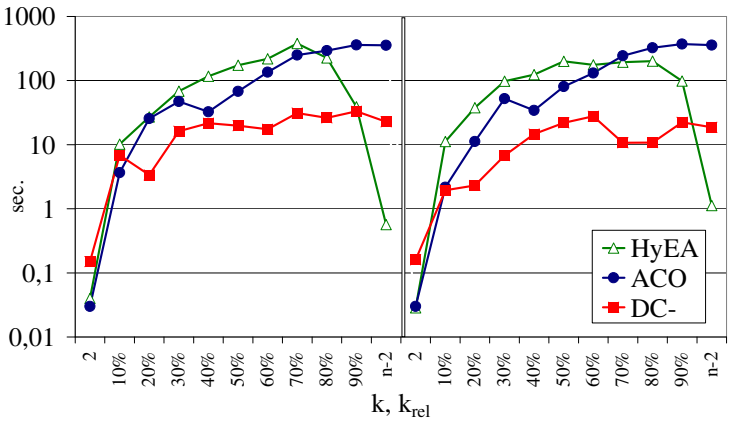


(b) grid 33x33-1 and 33x33-2 (1089 nodes, 2112 edges)

Figure 7.4: Running times of DC$^-$, HyEA, and ACO (in seconds) for instances of (BB) with ∼1000 nodes, depending on k. The figures for the grid and regular instances show the times for two different instances of the same type, respectively.



(c) dense steind15 (1000 nodes, 1250 edges)



(d) 4-regular g1000-4-1 and g1000-4-5 (1000 nodes, 2000 edges)

Figure 7.4: Continued.

avg. deg	set	≤500 nodes	1000 nodes
2.5	(BB)	1.0	8.1
4	(BB)	0.9	11.6
10	(BB)	2.2	18.8
20	(UBM)	4.6	32.3
36.3	(BB)	6.3	—

Table 7.8: Average CPU time (in seconds) over k_{rel} values of 10%, 20%, ..., 50%, sorted by the average degree of the graphs.

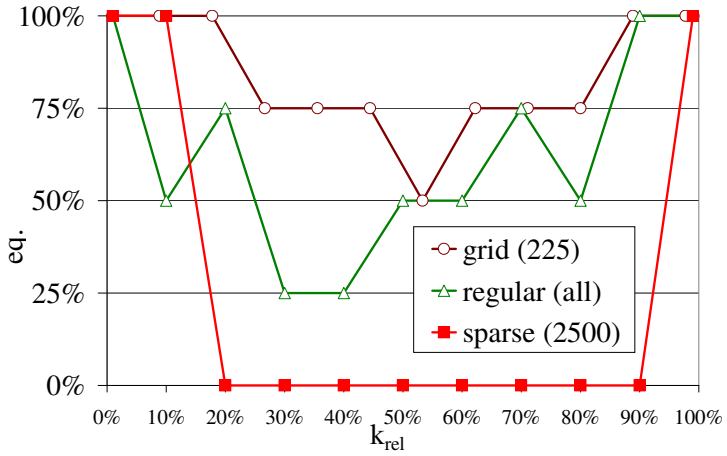


Figure 7.5: Dependency of the BKS quality on k_{rel}, for selected instances. The vertical axis gives the percentage of the tested instances for which the BKS provided in [BB03] are optimal.

respectively. Using the well-known SPEC performance evaluation [Spe08], we computed scaling factors of both machines to our computer: For the running time comparison we divided the times given in [Blu06] and [BS04] by 1.75 and 2.33, respectively. Anyhow, note that these factors are elaborate guesses at best and are only meant to help the reader to better evaluate the relative performance. The computational study on MSA′-DFLOW and MSA′-MTZ based algorithms was performed using CPLEX 10.2.0 on a Pentium Xeon machine running at 3.0 GHz with 2GBytes of RAM. Unfortunately these specification of the used computer is not exact enough to compute the precise scaling factor to our computer, but we can safely assume overall similar performance.

Table 7.7 gives the average factor of $t_{\text{HyEA}}/t_{\text{DC}^-}$, i.e., the running time of our algorithm compared to the (scaled) running time of HyEA. Analogously, Figure 7.4 shows the CPU time in (scaled) seconds of HyEA, ACO and our algorithm. We observe that our DC$^-$ algorithm performs better than the best metaheuristics in particular for the medium values of k, i.e., 40–70% of $|V|$, on all instances with up to 1089 nodes, except for the very dense graph `le450_15a` with 450 nodes and 8168 edges, where HyEA was slightly faster. Interestingly, the gap between the heuristic and the optimal solution tends to be larger especially for medium values of k (cf.

instance	($\|V\|$,$\|E\|$)	**avg. deg**	**eq.**	$\mathbf{gap}_{\text{bks}}$	$\mathbf{gap}_{\text{avg}}$ **ACO**	$\mathbf{gap}_{\text{avg}}$ **HyEA**
regular g400-4-1	(400,800)	4	10/11	0.09	0.07	0.04
regular g400-4-5	(400,800)	4	8/11	0.19	0.31	0.35
regular g1000-4-1	(1000,2000)	4	7/11	0.07	0.65	0.12
regular g1000-4-5	(1000,2000)	4	3/11	0.08	0.45	0.35
sparse steinc5	(500,625)	2,5	11/11	—	0.97	0.06
sparse steind5	(1000,1250)	2,5	11/11	—	0.48	0.11
sparse steine5	(2500,3125)	2,5	3/11	0.13	n/a	0.23
dense le450a	(450,8168)	36,3	11/11	—	n/a	0.04
dense steinc15	(500,2500)	10	11/11	—	0.36	0.02
dense steind15	(1000,5000)	10	10/11	0.22	0.38	0.04
grid 15x15-1	(225,400)	3,7	13/13	—	1.27	0.18
grid 15x15-2	(225,400)	3,7	13/13	—	2.04	0.12
grid 45x5-1	(225,400)	3,6	4/13	0.54	n/a	1.22
grid 45x5-2	(225,400)	3,6	10/13	0.08	n/a	0.13
grid 33x33-1	(1089,2112)	3,9	3/12	0.31	1.70	0.57
grid 33x33-2	(1089,2112)	3,9	3/12	0.39	2.48	0.49
grid 100x10-1	(1000,1890)	3,8	3/12	0,42	n/a	0.30
grid 100x10-2	(1000,1890)	3,8	2/12	0,80	n/a	0.65
grid 50x50-1	(2500,4900)	3,9	2/11	0.95	n/a	1.27
grid 50x50-2	(2500,4900)	3,9	2/11	0.55	n/a	0.82

Table 7.9: Quality of previously best known solutions (BKS) provided in [BB03] for selected instances. "**eq.**" denotes the number of instances for which the BKS was optimal. For the other instances where BKS was not optimal, we give the average relative gap ($\mathbf{gap}_{\text{bks}}$) between OPT and BKS. For all instances we also give the average relative gap ($\mathbf{gap}_{\text{avg}}$) between the average solution of the metaheuristic and OPT—including the 0-gaps for optimally solved instances. All gaps are given in percent. Cells marked as "n/a" cannot be computed as the necessary data for ACO is not available.

next Section and Figure 7.5 for details).

In Tables 7.2–7.6, for each tested (BB) instance, we give the value of the corresponding optimal solution as well as the CPU time DC^- needed to compute this solution, and the value of the KCA′-DCUT LP-relaxation. Additionally, for those instances that were also considered in [QCML10], we give the running times needed to compute the LP relaxation of MSA′-DFLOW (MCF) and those to compute an optimal solution by a MSA′-MTZ-based branch-and-bound algorithm (MTZ). Note that the long LP computation times of MCF made a corresponding branch-and-bound impossible.

Furthermore, the tables contain statistics regarding a Lagrangian heuristic (LH) of [QCML10]. OOM denotes aborted computations due to memory restrictions. We put "—" if the instance has not been considered in [QCML10]. If they give no results (eventhough the instance was considered by another of their algorithms) we denote it by "*".

We observe that MCF can solve LP relaxations only for grid instances with 225 nodes and small k values as well as sparse instances with up to 500 nodes. Although the computation of LP-relaxations of MTZ is very fast, the running times of the corresponding branch-and-bound algorithm are large due the extensive branching. MTZ optimally solves only some of the latter and a few more larger and denser instances. Whereas our DC^- solves all these instances in at most some seconds (usually less than a second), MCF and MTZ already need several hours. Furthermore, DC^- also clearly outperfoms the Lagrangian heuristic which needs significantly higher running times and does not compute provably optimal solutions. Note that such large differences in runnig times cannot be only explained by the use of different computers.

7.3.2 Solution Quality

For each instance of (BB), we compared the previously best known solutions, as published in [BB03], with the optimal solution obtained by our algorithm, in order to assess their quality. Most of the best known solutions (BKS) were found by HyEA, followed by ACO. Note that these solutions were obtained by taking the best solutions over 20 independent runs per instance. In Table 7.9 we show the number of instances for which we proved that BKS was in fact not optimal, and give the corresponding gap $\text{gap}_{\text{bks}} := (\text{BKS} - \text{OPT})/\text{OPT}$ (in percent) per graph, averaged over the different values for k. Here OPT denotes the optimal objective value obtained by DC^- and BKS denotes the best known solution obtained by either ACO or HyEA. Analogously, we give the gaps $\text{gap}_{\text{avg}} := (\text{AVG} - \text{OPT})/\text{OPT}$ (in percent), where AVG denotes the average solution over 20 runs, obtained by a metaheuristic. We observe that—concerning the solution quality—metaheuristics work quite well on instances with up to 1000 nodes and relatively small k. In particular, for $k = 2$ and $k = n - 2$ they always found an optimal solution.

Note that having comparable solution quality, the above metaheuristics are much faster than the Lagrangian heuristic presented in [QCML10].

	$\lvert V\rvert$	sparse, dense, regular						grid					
		500		1000		2500		225			1089	2500	
	k_{rel}	$\emptyset$	asym	$\emptyset$	asym	$\emptyset$		$\emptyset$	gsec	$\emptyset$	gsec	$\emptyset$	gsec
cuts	≤**33**	15.3	6.3	39.7	11.2	64.0		622.3	674.2	961.7	710.0	2978.3	2924.0
	34–66	38.1	5.3	115.7	14.5	209.3		241.4	178.5	752.7	499.0	1770.0	1358.0
	≥**66**	115.5	6.3	250.7	22.2	460.0		107.8	69.9	410.3	242.0	1173.7	7130.0
B&B	≤**33**	7.0	2.9	8.1	3.8	5.3		84.7	108.8	7.3	7.7	9.5	16.3
	34–66	2.0	1.1	5.8	1.0	5.0		5.8	9.1	3.3	4.8	28.5	2.8
	≥**66**	0.7	0.3	0.8	0.5	7.7		0.3	0.4	1.7	1.8	0.8	0.5
gap$_{\text{LP}}$	≤**33**	0.139%		0.032%		0.008%		2.645%		0.053%		0.046%	
	34–66	0.006%		0.003%		0.001%		0.148%		0.002%		0.001%	
	≥**66**	0.001%		0.001%		0.000%		0.006%		0.001%		0.000%	
time	≤**33**	1.9	7.2	16.9	59.6	45.3		14.5	24.7	152.5	102.5	6827.1	6901.9
	34–66	4.6	3.8	26.3	26.6	301.8		3.5	3.1	166.8	97.4	3201.4	2226.8
	≥**66**	24.1	3.2	71.2	21.9	1260.9		1.2	0.8	117.8	72.6	2790.6	1672.6
%rt	≤**33**	67.1%	80.3%	68.8%	76.1%	75.9%		18.6%	26.4%	85.16%	83.3%	82.5%	69.1%
	34–66	93.3%	88.9%	90.9%	93.3%	88.6%		80.4%	82.6%	95.9%	94.1%	97.2%	98.8%
	≥**66**	96.3%	96.7%	96.5%	97.1%	92.8%		98.9%	97.8%	95.9%	98.0%	99.0%	99.5%

Table 7.10: Behavior of DC$^-$ on (BB), depending on graph type, size ($|V|$), k_{rel} (in %), and additional constraints ($\emptyset$ denotes no additional constraints, *asym* and *gsec* denote the constraints (6.3) and (6.2), respectively). We give the number of generated dcut-constraints (*cuts*), the number of additional branch-and-bound nodes apart from the root (*B&B*), the gap between the LP relaxation at the root and the optimal integer solution (gap_{LP}), the overall computation time (*time*) and the percentage of that time spent at the root problem (*%rt*). The table ignores the irrelevant settings $k = 2$ and $k = |V| - 2$.

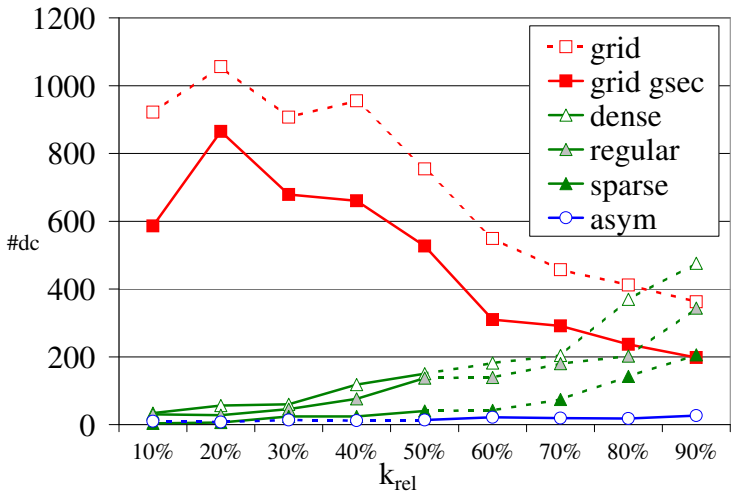


Figure 7.6: Dependency of the number of generated dcut-constraints on k_{rel}, for instances with 1000–1089 nodes. The solid lines denote the parameter choices used in the comparative study. When using the asymmetry constraints (asym), the lines for sparse, regular, and dense graphs become visually indistinguishable; hence we show their average.

7.4 Branch-and-Cut Specific Statistics

We conclude this part of the experimental study by analyzing certain properties of DC^- to better understand why it performs that well. Table 7.10 shows that the gaps between the value of the LP relaxation obtained at the root node of the branch-and-bound tree and the integer optimal solution are very tight and in fact often optimal. Furthermore, we observe that we need only very few cuts and branches to solve the ILPs.

The most interesting fact—in accordance with the runtime dependency on k—is visualized in Figure 7.6, selecting the graphs with 1000 nodes as a representative example. We see that for the sparse, regular, and dense graphs, the number of separated dcut-constraints grows with increasing k. Also observe that when using the asymmetry constraints (6.3), the number of necessary dcut-constraints drops further. In contrast to these observation, we see that for the grid graphs the number of required cuts is actually decreasing for growing k.

7.5 Further EKCT Instances

We investigated our algorithm's behavior on two additional sets of graphs, not considered in other practical papers for the KCT problem.

(UBM) with larger cardinality: In contrast to the originally suggested cardinalities for (UBM), we also performed tests for $k_{\text{rel}} = \{60\%, \ldots, 90\%\}$. This allows us to investigate our algorithm's behavior for large cardinalities and the influence of the asymmetry constraints (6.3). Table 7.11 summarizes the

	$\|V\|$	**500**	**1000**	**1500**	**2000**
k_{rel}	**60%**	6.3	33.0	68.0	177.9
	70%	5.0	27.8	59.8	165.6
	80%	5.0	30.7	63.2	156.5
	90%	4.4	32.4	53.8	151.1

Table 7.11: Average running times (in seconds) of DC$^-$ using asymmetry constraints (6.3), for (UBM) instances with higher k_{rel}.

dimension	**6**	**7**	**8**	**9**	**10**	**11**	**12**
# of nodes	64	128	256	512	1024	2048	4096
avg. time in sec.	0.03	0.16	1.10	9.58	45.59	316.40	

Table 7.12: Average running times for the hypercube instances.

average running times of our algorithm. We are able to solve all instances to provable optimality. The running times are even slightly decreasing for the larger k values, due to the help of (6.3). Overall, these results, as well as the speedup observed in Table 7.1, confirms our finding that these constraints are beneficial for $k_{\text{rel}} \geq 50\%$.

Hypercubes: This benchmark set was introduced in [RdAR$^+$01] and is also part of SteinLib [KMV03]. It contains 6 hypercube graphs of dimension 6–12 with edge weights uniformly distributed in the interval [100,110]. A hypercube of dimension d has 2^d nodes interconnected to form the edges of a hypercube. We choose this benchmark set as these graphs turn out to be highly challenging for the Steiner tree problem, as, e.g., the structure and the similar edge weights do not allow strong preprocessing strategies. In contrast to these observation, our experiments show that in the context of the KCT and for the tested values of $k_{\text{rel}} = \{10\%, 20\%, \ldots, 90\%\}$, these instances can be efficiently solved using Kca′-DCut, cf. Table 7.12.

7.6 Node-weighted KCT

We also applied our algorithm to the NKCT problem using the cost transformation described in Section 5.2.

7.6.1 NKCT Instances

For the NKCT problem there is no established benchmark library such as KCTLIB for EKCT. However, the following instance sets are used in the literature:

(BE) This set was used in [EFHM97, BE03] and contains 20 randomly generated connected graphs and grid graphs with 10, 20 and 30 nodes.

(BUM) Grid graphs with 30×30, 40×40 and 50×50 nodes and random graphs with 3000, 4000, and 5000 nodes of average density 10 are presented in [BUM06]. For each size of a graph there are 10 different instances with random node-weights that are uniformly distributed in the interval [10, 100]. The considered

k_{rel} values are $\{10\%, \ldots, 50\%\}$. This set is used for the VNDS [BUM06], ant colony algorithm with integrated dynamic programming [BB05a] and for the previously mentioned hyEA [Blu06]. A performance comparison of these algorithms on the (BUM) set is presented in [Blu06]. For the random instances, VNDS clearly outperforms other metaheuristics, whereas HyEA gives better results for the grid graphs and small cardinalities.

QCML In [QCML10], computational results are presented on grid instances with 10×10, 15×15 and 20×20 nodes generated as suggested in [BUM06]. Note that as the node weights are randomly chosen, these instances differ from the original grid instances of the above (BUM) set.

Unfortunately, the random instances from the (BUM) set were not available for us. The (BE) instance set contains quite small graphs with up to 30 nodes, which do not represent any challenge for our algorithm and are too small for an extensive analysis. Hence, we report on the performance of our algorithm on the NKCT problem instances only on the grid graphs of the (BUM) and (QCML) sets.

7.6.2 Parameter Influence

In order to test the right parameter setting we randomly took two instances per each instance size and applied different algorithm variants only on this smaller instance set. We tested all combinations of using the following binary parameters, which are 1 when the specified effect is activated:

asym Add asymmetry constraints to the initial LP.

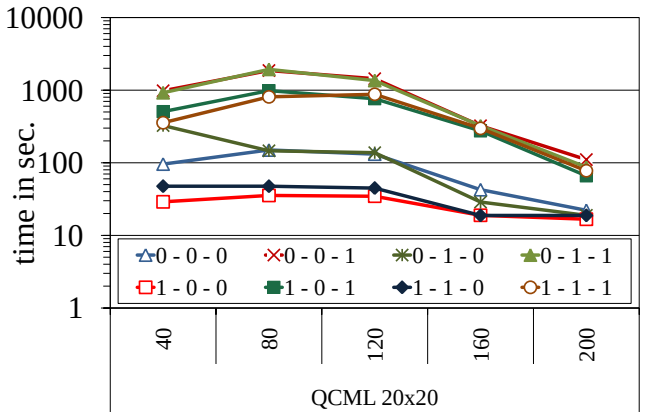
mcc In the separation procedure, look for *minimum cardinality* cuts (mc-cuts), i.e., cuts containing the smallest number of cut edges among all most violated cuts of type (5.12).

gsec Separate 4-cycle GSECs (6.2).

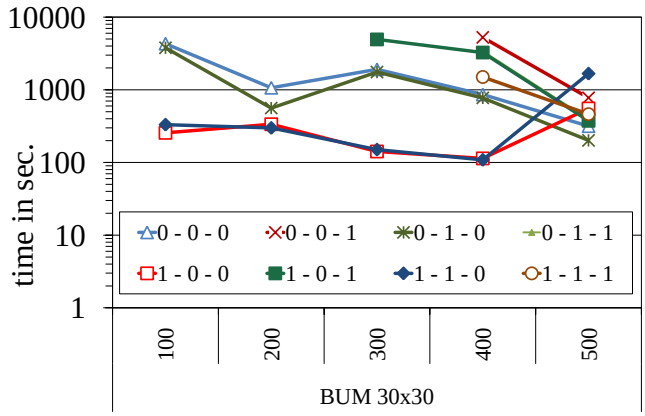
In Figure 7.7 we visualize our findings for the selected (QCML) and (BUM) instances with 400 and 900 nodes, respectively. An interesting finding is that for solving these node-weighted instances with $k_{\text{rel}} \leq 50\%$ efficiently in practice, it is crucial to use mc-cuts. The same strategy did never turn out to be particularly useful for the edge-weighted KCT problem. However, for the survivable network design problems (cf. Part III of this thesis), this technique also significantly reduces the computational time of the corresponding branch-and-cut algorithm. As it was the case for the edge-weighted grid instances, it is always better not to use asymmetry constraints for the initial LP. The impact of 4-cycle GSECs is not as big as for the edge-weighted KCT instances. Hence, we decided to run DC^- with mcc $= 1$ whereas asym $=$ gsec $= 0$ in the following.

7.6.3 Evaluation

Recall that (QCML) instances contain grid graphs with 100, 225 and 400 nodes whereas (BUM) instances are larger grid graphs with 900, 1600 and 2500 nodes.



(a)



(b)

Figure 7.7: Comparison of average running times for different parameter settings on a sample set of node-weighted 20x20 and 30x30 grid graphs. The parameters are listed as follows mcc-gsec-asym.

We solve all (QCML) instances to provable optimality within few seconds. The running times and optimal solutions for all these instances are listed in Table 7.13 that also contains the running times of MCF, MTZ, and the Lagrangian heuristic, as well the solution values of the latter. These results show that also on the node-weighted instances, our algorithms are much faster (by orders of magnitude) than the heuristic and exact approaches of [QCML10].

As it is more difficult to solve the (larger) grid instances of (BUM) than (QCML), we present a detailed analysis of DC$^-$ based on (BUM), cf. Table 7.14. Our experiments show that we can solve all but one of these instances with up to 1600 nodes and most of those with 2500 to provable optimality in under two hours. For graphs with up to 1600 nodes we require under half an hour. Analogously to the edge-weighted grid instances, the lower bounds obtained by solving the LP relaxation of KCA′-DCUT are very close to the optimum and get more tight with growing k_{rel}. This is also the reason why it does not pay of using primal heuristics for our algorithm: most of the CPU time of DC$^-$ is spent over computing the lower bounds and the number of branch-and-bound nodes is comparably small.

For small values of k_{rel}, the corresponding LP relaxation is less tight and more branching is required. For larger values of k_{rel} the lower bounds are much tighter, but the time needed to compute these bounds also grows. This is the reason why instances with $k_{\text{rel}} = 10\%$ and $k_{\text{rel}} = 50\%$ are harder than the others.

Using Table 7.15, we can compare the running times and the solution quality of our algorithm with those of the state-of-the-art metaheuristics for node-weighted grid instances, HyEA [Blu06] and VNDS [BUM06]. Note that the latter algorithm again was run on a comparably old 733MHz Pentium III computer, so we cannot fairly estimate the speed-up factor of our computer. The running times of HyEA are divided by factor 1.75, as explained on page 71. These special purpose metaheuristics for the NKCT are indeed faster than our exact approach, though not optimal in most cases.

Comparing our results on EKCT and NKCT, we can say that our algorithm performs faster on the edge-weighted instances than on the node-weighted ones. This is in particular the case for the large grid instances with $k_{\text{rel}} = 50\%$. We suppose that this is because solving the NKCT problem via KCA′-DCUT produces many symmetric solutions as all arcs entering the same node have the same cost. Breaking this symmetry would possibly lead to better performance of our algorithm.

		time DC$^-$		time [QCML10]			obj.val.	
	k	**ILP**	**LP**	**MCF-LP**	**MTZ-ILP**	**LH**	**opt**	**LH**
n10x10_1	50	0.32	0.31	280.4	16794.61	28.14	16052	16052
n10x10_2	50	0.16	0.16	38.88	1142.42	24.44	14026	14026
n10x10_3	50	0.16	0.16	234.85	535.55	27.21	13117	13117
n10x10_4	50	0.44	0.24	362.4	14087.91	27.19	14254	14283
n10x10_5	50	0.33	0.28	469.26	32373.64	25.06	17093	17093
n15x15_1	112	2.72	2.56	8161.55	55682.19	150.44	38322	38322
n15x15_2	112	2.17	2.17	8192.85	OOM	149.62	32061	32081
n15x15_3	112	3.31	3.31	14822.86	OOM	150.46	31321	31321
n15x15_4	112	2.01	2.01	7247.34	OOM	152.28	30681	30931
n15x15_5	112	2.10	2.10	10541.46	OOM	149.83	29869	30063
n20x20_1	200	13.22	13.22	113885.98	OOM	516.33	55443	55641
n20x20_2	200	13.33	13.33	63061.88	OOM	469.21	54894	55073
n20x20_3	200	11.14	11.14	161523.95	OOM	489.58	53912	54127
n20x20_4	200	10.45	10.45	106490.39	OOM	500.62	53512	53761
n20x20_5	200	15.30	15.30	81925.95	OOM	518.46	62364	62557

Table 7.13: Times (in sec.) and solution values of DC$^-$ and the algorithms in [QCML10] for (QCML) instances.

$\|V\|$	k_{rel}	**o/i**	**time**	**B&B**	**gap$_{\text{LP}}$**	**rt%**	**cuts**
900	10	10/10	381.71	18.10	0.28	58.66	2174.4
	20	10/10	306.04	6.80	0.02	84.19	2090.8
	30	10/10	148.60	5.40	0.01	86.11	1565.6
	40	10/10	143.00	1.90	0.01	92.02	1500.8
	50	10/10	726.57	0.20	0.00	99.77	2445.6
1600	10	9/10	3303.99	340.60	0.43	48.10	4224.0
	20	10/10	1642.59	12.60	0.02	84.89	3536.0
	30	10/10	1110.66	5.10	0.01	93.12	3048.6
	40	10/10	819.38	2.60	0.01	93.72	2582.6
	50	10/10	1252.34	1.30	0.00	95.92	2803.4
2500	10	5/10	5170.57	47.50	0.23	77.37	5328.8
	20	8/10	4868.68	7.40	0.00	93.34	5228.0
	30	10/10	4058.91	4.90	0.00	90.23	4417.2
	40	9/10	4323.48	4.70	0.00	93.77	4455.2
	50	1/10	6337.64	0.10	0.00	99.98	5212.0

Table 7.14: Behavior of DC$^-$ on the node-weighted grid instances of (BUM), depending on graph size ($|V| = 900, 1600, 2500$) and k_{rel} (in %). Column "o/i" gives the number of instances optimally solved in 2 hours together with the number of considered instances. We furthermore give the overall computation time (*time*), the number of additional branch-and-bound nodes apart from the root (*B&B*), the gap between the LP relaxation at the root and the optimal integer solution (gap_{LP}), the percentage of that time spent at the root problem (*%rt*), and the number of generated dcut-constraints (*cuts*).

$\|V\|$	k_{rel}	c_{opt}	$\mathbf{gap}_{\text{avg}}$ **HyEA**	$\mathbf{gap}_{\text{avg}}$ **VNDS**	t_{ILP} **DC$^-$**	$t/1.75$ **HyEA**	t **VNDS**
900	10	8166.7	0.49	4.96	381.71	36.25	(24.00)
	20	17636.1	0.74	2.03	306.04	55.99	(88.00)
	30	28629.3	0.75	0.49	148.60	59.51	(126.00)
	40	42064.9	0.52	0.12	143.00	67.31	(80.00)
	50	59392.9	0.27	0.21	726.57	63.03	(213.00)
1600	10(*)	17365.6	1.59	3.83	5750.71	131.08	(112.00)
	20	37052.4	1.26	5.16	1642.59	170.18	(114.00)
	30	59414.8	1.50	3.16	1110.66	154.17	(261.00)
	40	85375.9	1.40	1.23	819.38	168.88	(261.00)
	50	117503.5	0.94	0.13	1252.34	148.93	(303.00)
2500	30	122690.7	1.89	4.49	4058.91	162.85	(482.00)
	40(*)	179052.5	1.70	1.77	4648.26	179.17	(575.00)

Table 7.15: Comparison of running time (in seconds) and solution quality of DC$^-$ with hyEA [Blu06] and VNDS [BUM06] on node-weighted grid instances of (BUM), depending on graph size ($|V| = 900, 1600, 2500$) and k_{rel} (in %). All values are averaged over 10 instances per each value of $|V|$ and k_{rel}. We denote the value of the optimal solution c_{opt} and by $\mathbf{gap}_{\text{avg}}$ the average relative gap between the best found solution of the metaheuristic and c_{opt}. The running times of HyEA given in [Blu06] are divided by 1.75, due to the speed-up factor of the thereby used computer. The running times of VNDS are given in brackets as we do not know the corresponding speed-up factor, cf. text.

Part III

{0,1,2}-Survivable Network Design

Chapter 8

Considered Problems

In this chapter we consider problems of designing survivable network topologies. Such problems arise in real-world settings, e.g., in telecommunication. The task is to connect each pair of a given set of customers using a set of potential route-segments (e.g., streets) and an additional set of connection points between these route-segments (e.g., some technical devices or road-intersections). There are charges for using a given route-segment, e.g., these can be rental costs or working costs for laying a connection cable. Thus, we want to select the route-segments such that the resulting network is of minimum cost, connecting all customers. Up to now, we could model this problem as a traditional Steiner tree problem. Yet, for some customers it is important to guarantee a *survivable* connection: if some of the connection components do not work properly (e.g., due to cable damage or some other technical problems), alternative connections between such customers should exist.

Formalizing the above task as a graph problem, we can view the set of customers and connection points as graph nodes and the route-segments as edges between these nodes with a corresponding cost function. The task is to find a cost-minimal connected subgraph N that includes all nodes associated with customers. The survivability property can be expressed by requiring N to contain the desired number of disjoint paths between the specified customers. Depending on this number and the definition of disjointness—we may require the paths to be edge- or node-disjoint—we get a set of combinatorial optimization problems, all of which use the following common problem input:

Input. We are given a graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}^+$, and a vector of connectivity requirements $\varrho_G \in \{0, 1, \ldots, k\}^{|V|}$, where k is a positive integer number. We will omit the subscript in ϱ_G if the corresponding graph is clear from the context. For notational simplicity, we define $\mathcal{R}_i := \{v \in V \mid \varrho(v) = i\}$ for all $0 \leq i \leq k$, and call the set $\mathcal{R} := \bigcup_{i \neq 0} \mathcal{R}_i$ the *customer nodes*.

Problem 8.1 (kNCON)**.** The $\{0, 1, \ldots, k\}$*-node-connected Steiner network problem* is to find a subgraph $N = (V_N, E_N)$ of G that contains all nodes $v \in \mathcal{R}$, minimizes $\sum_{e \in E_N} c(e)$ and satisfies the following connectivity property: for every pair of nodes $s, t \in V_N$, N contains $\varrho(s, t) := \min\{\varrho(s), \varrho(t)\}$ node-disjoint paths connecting them.

Problem 8.2 (kECON)**.** The $\{0, 1, \ldots, k\}$*-edge-connected Steiner network problem* is obtained from kNCON by replacing node-disjointness with edge-disjointness.

Problem 8.3 (kCON)**.** We summarize both kECON and kNCON under the term kCON.

Some real-world tasks require a survivable connection between a customer and a special root node $r \in V$. Such a root node can represent an important connection hub or an already existing infrastructure network which should be extended by connecting new customers as it was the case in [WRP+06, WRP+07]. This can be formalized by the following problem:

Problem 8.4 (kRSN)**.** The $\{0, 1, \ldots, k\}$*-root-connected Steiner network problem* is closely related to kNCON but requires the node-wise k-connectedness with respect to the root r: each node $v \in \mathcal{R}$ has to have $\varrho(v)$ node-disjoint paths to r. This problem is also known as *rooted SNDP* [LN09] or single-source SNDP [CCK08].

Note that requiring edge-disjointness in the above problem setting would be equivalent to the already defined kECON problem.

All above problems can also be defined w.r.t. a prize-collecting setting. In this case, we are given a prize function $p : \mathcal{R} \to \mathbb{R}^+$, representing the potential prize (profit) of a node $v \in \mathcal{R}$ for including it into a solution network, and a prespecified node $r \in V$ that has to be included in any optimal solution. The task is to maximize the overall profit defined as the difference between the gains of the nodes contained in the solution and the total network installation costs. Thus we obtain the following three problems:

Problem 8.5 (kPCECON)**.** The task for the *prize-collecting* $\{0, 1, \ldots, k\}$*-edge-connected Steiner network* problem is to find a subgraph $N = (V_N, E_N)$ with $r \in V_N$ that minimizes $\sum_{e \in E_N} c(e) - \sum_{v \in V_N \cap \mathcal{R}} p(v)$. Thereby, the connectivity requirements of the customers chosen for the network have to be satisfied as for the kECON.

Problem 8.6 (kPCNCON)**.** The task for the *prize-collecting* $\{0, 1, \ldots, k\}$*-node-connected Steiner network* problem is to find a subgraph $N = (V_N, E_N)$ with $r \in V_N$ that minimizes $\sum_{e \in E_N} c(e) - \sum_{v \in V_N \cap \mathcal{R}} p(v)$. Thereby, the connectivity requirements of the customers chosen for the network have to be satisfied as for the kNCON.

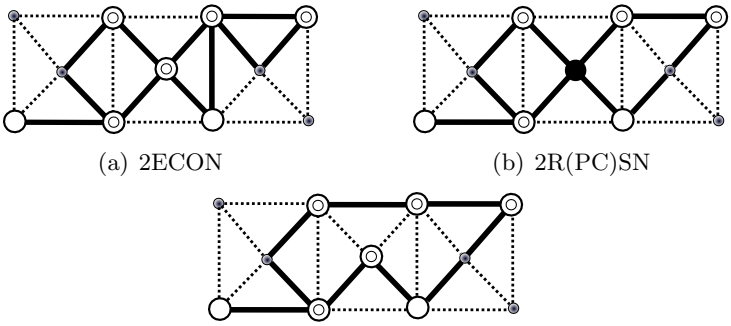
Problem 8.7 (kRPCSN)**.** The task for the *prize-collecting* $\{0, \ldots, k\}$*-root-connected Steiner network* problem is to find a subgraph $N = (V_N, E_N)$ that minimizes $\sum_{e \in E_N} c(e) - \sum_{v \in V_N \cap \mathcal{R}} p(v)$. Thereby, connectivity requirements of those customers chosen for the network have to be satisfied as for kRSN.

Problem 8.8 ($\{0, 1, \ldots, k\}$-SND)**.** We summarize the kECON, kNCON and kRSN problems and their prize-collecting variants under the term $\{0, 1, \ldots, k\}$*-Steiner network design problems*.

In this thesis we only consider $\{0, \ldots, k\}$-SND problems with $k = 2$. A summary of these problems is presented in Table 8. Although our algorithms are mainly

Problem Type	Edge-Connectivity	Node-Connectivity	
		rooted	unrooted
spanning	2ECON	2RSN	2NCON
prize-collecting	PC2ECON	2RPCSN	PC2NCON

Table 8.1: Summary of the problems discussed in this part of the thesis



(a) 2ECON (b) 2R(PC)SN

(c) 2NCON

Figure 8.1: Feasible networks for different {0,1,2}-SND problems. Bold edges belong to the solution networks whereas the dashed ones do not. Double circles represent $\mathcal{R}_2$ nodes, white circles correspond to $\mathcal{R}_1$, and small circles to $\mathcal{R}_0$. (b) The black node in the middle represents the root r.

developed in the context of node-connectivity requirements, they are also able to deal with the corresponding edge-connected problem variants. Figures 8.1(a), 8.1(b) and 8.1(c) illustrate examples of feasible solutions to 2ECON, 2RSN and 2NCON, respectively.

In this part of the thesis we develop new exact algorithms for large real-world 2RSN and 2NCON problem instances and their prize-collecting variants. Thus the main emphasis of our work is on survivable network design problems with low-connectivity requirements and node-disjointness constraints. Both 2RSN and 2NCON problems are prominent special cases of the general *survivable network design problem*, also known as *generalized Steiner network*. In Chapter 9, we will classify our problems within this larger class and give an extensive overview over these and related problems. Thereby, we discuss complexity issues and existing heuristic and approximation algorithms. As we are particularly interested in exact algorithms for our problems we will discuss the work that has been done in this area in a separate section.

Most existing exact algorithms use integer linear program formulations that are based on undirected graphs. However, for 2ECON, we know that orientation properties of the feasible solution networks can be used to derive stronger ILP formulations that lead to more efficient algorithms, in theory as well as in practice. For the feasible solution networks of node-connectivity problems no such orientation properties were known that could be used for stronger ILP formulations. In Chapter 10, we discuss the structure of {0,1,2}-SND solutions and derive novel

orientation properties. We will then exploit these for our new ILP formulations.

Our branch-and-cut algorithm will be described in Chapter 11. Its extensive experimental analysis will be given in Chapter 12, where we also introduce a common benchmark set for our {0,1,2}-SND problems: until now, different test instances were used by numerous research communities. We collected all these instances into a one benchmark set that can be used for future research.

Chapter 9

Literature Overview

9.1 The CON problem

Both kCON and kRSN are special cases of the more general network design problem that was first introduced in [SWK69]. Winter [Win87b] called the problem the *generalized Steiner network problem* and in, e.g., [FJW06, KM05a] it can be found as *survivable network design problem*. In this thesis we use the name $CON(G, k)$ or simply *CON* established by Stoer [Sto92].

Problem 9.1 (CON)**.** Instead of a vector of connectivity requirements for the nodes in V, the input of the CON problem contains a symmetric matrix of connectivity requirements with entries $\varrho(u, v) \in \{0, 1, \ldots, k\}$ for each pair of nodes $u, v \in V$. The *ECON* and *NCON* problems consist of finding a network of minimum cost such that for every pair of nodes $u, v \in V$ there exist at least $\varrho(u, v)$ edge- or node-disjoint paths, respectively. We speak about the CON problem when we do not need to distinguish between the ECON and NCON variants.

The CON problem contains the Steiner tree and the traveling salesman problems as special cases and is therefore, in general, NP-hard. To model the input of kCON as an instance of CON we define $\varrho(u, v) := \min\{\varrho(u), \varrho(v)\}$. Analogously, for the kRSN input we set $\varrho(r, v) = \varrho(v, r) := \varrho(v)$ and $\varrho(u, v) := 0$ for $u, v \neq r$. Recall that for both problems we have $k = \max\{\varrho(v) \mid v \in V\}$.

The CON problem has been extensively studied by various research communities, see, e.g., [KM05a, KN07, Rag95, Sto92, Win87b] for surveys. However, less is known for the prize-collecting survivable network design variant. A survey on approximation algorithms for these latter (and related) variants can be found in [HKKN10]. In this chapter, we will list the most significant results for CON and its most prominent subvariants, in order to give the reader an intuition about the hardness and structure of these problems. For this purpose, we first give a brief overview over the known algorithms for the general CON problem. We then report on the kNCON and kRSN problems, also paying attention to their special cases with uniform connectivity types. In this section, as well as in Sections 9.2 and 9.3, we outline polynomially solvable special cases, heuristics and, finally, approximation algorithms for CON, kCON and kRSN, respectively. A literature overview of the corresponding exact algorithms is given in Section 9.4, followed by an overview over

the published experimental studies for CON problems in Section 9.5. Such studies were conducted only for some heuristics and ILP-based exact algorithms. To our knowledge, the practical performance of approximation algorithms has never been analyzed.

Some papers deal with algorithms for the CON problem that allow the use of multiple edges. We do not report on them here as most of these algorithms cannot be used for the CON problem without multiple edges. Moreover, we do not list the results for CON(G,1), i.e., for MST, STP, PCST, and shortest path problems. The algorithms for these special cases cannot be applied to the case of $k = 2$ which is the focus of this thesis. In the following we will only report on the results for $k \geq 2$ as they may be particularly relevant for our 2RSN and 2NCON problems.

The following two problems are often considered special cases of the CON problem and are subject of extensive research that goes beyond the scope of this thesis (see, e.g., [Nut09c, KN07] for details). We will only report on polynomially solvable special cases of these problems.

Problem 9.2 (CON with binary costs)**.** The *unweighted connectivity augmentation problem* is to augment a given graph $H = (V_H, E_H)$ with a minimum number of edges in $\binom{V_H}{2} \setminus E_H$ such that the resulting graph satisfies the given edge- or node-connectivity requirements $\varrho(u, v) \leq k$ for all $u, v \in V_H$. To model the unweighted connectivity augmentation problem as CON(G, k) we define G as a complete graph with $V = V_H$ and $c(e) := 0$ if $e \in E_H$ and $c(e) := 1$ otherwise.

Problem 9.3 (CON with uniform costs)**.** Given a node set V and edge- or node-connectivity requirements $\varrho(u, v) \leq k$ for every pair of nodes $u, v \in V$, the task is to find a graph satisfying these requirements with the minimum number of edges. This problem can be modeled as CON(G, k) by defining G as a complete graph on a node set V with uniform costs.

The edge-connectivity versions of Problems 9.2 and 9.3 are polynomially solvable if the use of parallel edges or additional nodes is allowed, see [Fra92] and [CF70], respectively. Further polynomial special cases arise when restricting the domain of the connectivity matrix, cf. Section 9.2.2. Note that the complexity of Problem 9.3 with general connectivity requirements is still open. An unpublished 2-approximation is mentioned in [Nut09c].

Heuristics. For ECON, several construction and local search heuristics are given in [SWK69] for complete input graphs. A GRASP heuristic (Greedy Randomized Adaptive Search Procedure) for NCON, which is based on multiple applications of randomized construction heuristics and local search procedures, is presented in [CRR03]. Furthermore, a dual-ascent algorithm for the CON problems was presented in [Rag95] leading to both, heuristic and methods to obtain lower bounds.

Approximation algorithms. For ECON, the best known approximation algorithm guarantees factor 2 [Jai98]. It is based on an iterative rounding technique and therefore not combinatorial. Other algorithms (with worse factors) are either

based on a primal-dual approach [GGP+94, WGMV95] or are purely combinatorial [BMM04].

For the general NCON problem, no constant factor approximation is known. The first non-trivial approximation was very recently published in [CK09] and guarantees the factor $\mathcal{O}(k^3 \log |V|)$. Furthermore, if the costs satisfy the triangle inequality, there exists a $O(k)$-approximation [CV07]. For $k = 2$ and general costs, a 2-approximation was presented in [FJW06] and uses techniques similar to [Jai98]. Previously, a primal-dual 3-approximation for the latter problem was suggested in [RW97].

Clearly, all above algorithms can also be applied to our {0,1,2}-SND problems as they are designed to solve the more general network design problems.

9.2 kCON

Polynomially solvable cases. The kCON problem is known to be polynomially solvable in following cases:

1. The kECON problem with even $\varrho(v)$ for all $v \in V$ is polynomially solvable on series-parallel graphs [KM05b].

2. For both 2ECON and 2NCON problems, i.e., $\varrho \in \{0, 1, 2\}^{|V|}$, Raghavan [Rag04] presented linear time algorithms on series-parallel graphs. Therein he also suggests how to use these algorithms for preprocessing.

3. A polynomial-time algorithm based on the ellipsoid-method was suggested in [BKM08] for the 2ECON problem with $\varrho \in \{1, 2\}^{|V|}$ on a subclass of series-parallel graphs which strictly contains all the outerplanar graphs. Note that a constructive algorithm for a more general 2CON problem on series-parallel graphs was already given by Raghavan [Rag04].

4. The 2NCON problem is polynomially solvable if the underlying graph is a *k-tree* [GR02]. A graph $G = (V, E)$ with $k \leq |V|$ is a k-tree if it can be constructed from a clique on k nodes by successively inserting a new node and connecting it to each node of the clique.

5. Further polynomial special cases arise from restricting the connectivity requirements, see Section 9.2.1 and 9.2.2.

Heuristics. For the case $\varrho \in \{1, 2\}^{|V|}$, several construction and local search heuristics for 2NCON were developed in [MS89]. A genetic algorithm for the 2NCON problem was given in [GR02]. For 2ECON two combinatorial heuristics are presented in [BMM04]. They are special cases of an approximation algorithm for the ECON problem presented in the same paper. Also for 2ECON, an improved dual-ascent procedure was described in [Rag95].

Approximation algorithms. The best known approximation ratio for kECON is 2, as it is a special case of the ECON problem. For 2ECON, if the costs satisfy the triangle inequalities the above mentioned heuristics of [BMM04] have approximation ratios of $\frac{5}{2}$ and $\frac{7}{3}$, respectively.

9.2.1 k-Connected Steiner Networks

Problem 9.4 (kCSN)**.** If a kCON instance has $\varrho(v) \in \{0, k\}$ for every $v \in V$, we are dealing with the *k-connected Steiner network problem*.

Structural results. Monma et al. [MMP90] analyzed the 2CSN problem where the input graph is complete and the cost function is metric. They show that any non-customer node used in an optimal network has degree 3. Furthermore they prove another interesting result: Let N be the set of all nodes with requirement 2. The cost of the minimum 2-connected network spanning N (2CSS, cf. next section) is at most $\frac{4}{3}$ of the cost of the minimum 2-connected Steiner network including N. More structural results on Euclidean 2CSN problem can be found in [WZ05, PLZC07].

Polynomially solvable cases. The following special cases of the kCSN problems are known to be polynomially solvable:

1. The k-shortest path problem, i.e, exactly two nodes in $u, w \in V$ have connectivity requirement k and all other nodes are in $\mathcal{R}_0$ [Suu74, ST84].

2. As mentioned before, the 2CON problem is polynomially solvable on series-parallel graphs. Hence, the 2CSN problem is also polynomially solvable on this graph class. This fact, however, has been shown before: linear-time algorithms for both edge- and node-connectivity were presented for some special graph classes such as outerplanar[1] [Win85] and series-parallel graphs [Win86]. A polynomial algorithm for 2ECSN on series-parallel graphs was given in [BM97].

3. For Halin graphs, linear-time algorithms are known for $k = 2$ and $k = 3$, for both edge- and node-connectivity [Win87a, Win87b]. Later, Coullard et al. [CRRW93] gave a linear time algorithm with node-disjointness constraints for Halin graphs and graphs that do not contain W_4 as a minor.

Approximation algorithms. It was shown in [CCK08] that the kCSN problem is $k^{\Omega(1)}$-hard to approximate. For node-connectivity requirements and general k, Nutov [Nut09b] presented a $\mathcal{O}(k^2 \log k)$-approximation, improving the previously best known factor $\mathcal{O}(k^2 \log |V|)$ due to Chuzhoy and Khanna [CK09].

9.2.2 k-Connected Spanning Subgraph

Problem 9.5 (kCSS)**.** If a kCON instance has a uniform vector ϱ, i.e., $\varrho(v) = k$ for every $v \in V$, we are dealing with the *k-connected spanning subgraph problem*.

[1] Outerplanar graphs are also series-parallel.

Polynomially solvable cases. The kCSS problem is known to be polynomially solvable in the following cases:

1. Problem 9.2 with uniform connectivity requirements $\varrho(v) = 2$ for all $v \in V$ [ET76a, RG77]. All known polynomial algorithms for uniform $\varrho(v) > 2$ for all $v \in V$ require the use of parallel edges [CS89, NGM90, UKW88, WN87].
2. Problem 9.3 with node-connectivity requirements and uniform ϱ [Har62].
3. The general kECSS problem on series-parallel graphs [BM96].

Structural results. The following theorem was shown in [FJ82]:

Theorem 9.6. *Consider a complete graph G with a cost function $c : E \to \mathbb{R}^+$ that satisfies the triangle inequality. There exists an optimal solution to every such 2ECSS instance that is also optimal for 2NCSS. Thus, for the above input 2ECSS and 2NCSS are equivalent.*

Also for the case of a complete input graph G, Monma et al. [MMP90] characterize optimal solutions of the 2-connected spanning subgraph problem as follows:

(1) There exists an optimal solution N where all nodes have degree 2 or 3.

(2) N is edge-minimal, i.e., deleting any edge leaves a bridge.

(3) Deleting any pair of edges in N leaves a bridge in one of the resulting connected components.

Bienstock et al. [BBM90] extended the above characterization for the general k-connected spanning subgraph problem. Monma et al. [MMP90] furthermore showed that every 2-connected graph N with properties (1)–(3) is the unique minimum-cost 2-connected network spanning V for the *graph-theoretic distance* function, i.e., $c(\{u, v\})$ is the number of edges in a shortest path from u to v. This result is, however, not valid for general k, even when restricted to the metric costs [BBM90].

Heuristics. For general k, Ko and Monma [KM89] developed heuristics based on the ideas of [MS89]. For $k = 2$, a bootstrap heuristic was given in [CA95].

Approximation algorithms. For the kCSS problem, a large body of approximation algorithms has been developed in the last 20 years. There are several surveys on this topic, e.g., [Khu97, KM05a].

For general weights and $k = 2$, the best approximation ratio for both edge- and node-connectivity is 2. For edge-connectivity this was achieved by Khuller and Vishkin [KV94]. Penn and Shasha-Krupnik [PSK97] achieved this factor for node-connectivity using results of Khuller and Raghavachari [PSK97].

For general k, the approximation factor 2 holds only for edge-connectivity [KV94, Jai98]. Unfortunately, for node-connectivity no constant factor approximation algorithm is known yet. The first k-approximation was suggested in [KN03], and further improvements were then achieved in [KN05, FL08]. Finally, the factor

$\mathcal{O}(\log k \cdot \log \frac{|V|}{|V|-k})$ was achieved in [Nut09a]. Unless $k = |V| - o(|V|)$, the latter factor reduces to $\mathcal{O}(\log k)$. Note that the $2H(k)$-approximation (where $H(k) = 1 + \frac{1}{2} + \cdots + \frac{1}{k}$) presented in [RW97] was erroneous and does not hold for general k [RW02]. If the costs satisfy the triangle inequalities, a $2 + \frac{k-1}{|V|}$-approximation can be found in [KN03].

Better approximation factors are known for restricted k: For $k \leq \sqrt{|V|/6}$, a $\mathcal{O}(\log k)$-approximation algorithm is given in [CVV02]. For $k \leq 7$, the problem can be approximated with factor $\lceil \frac{k+1}{2} \rceil$ [KR96, ADNP99, DN99, KN03].

9.3 kRSN

Until now, little was known about the kRSN problem. Approximation algorithms appeared only very recently. Several heuristics were presented in the context of ILP-based algorithms for the 2R(PC)SN problems.

Heuristics For 2R(PC)SN, a heuristic was suggested in [LR08]. As it is similar to our heuristic framework for 2NCON and 2R(PC)SN, we briefly describe it here. First, a Steiner tree $T = (V_T, E_T)$ on G with terminal set $\mathcal{R}$ is computed. Then, T is extended by successively ensuring 2-connectivity of $\mathcal{R}_2$ customers: For every $v \in \mathcal{R}_2$, two node-disjoint paths from r to v are computed in a transformed directed graph using the algorithm of Suurballe and Tarjan [ST84]. Thereby, the costs of arcs corresponding to edges in E_T are set to 0.

Approximation algorithms. The general kRSN problem is hard to approximate, see [LN09,KKL04] for details. For $k = 2$, there is a 2-approximation [FJW06], as 2RSN is a special case of NCON$(G, 2)$ (cf. Section 9.2). For $k \geq 3$, the first approximation algorithm (with factor $k^{\mathcal{O}(k^2)} \log^4 |V|$) was given in [CCK08]. Recently, the general kRSN problem was shown to be approximatable within $k \log k$ [Nut09b]. This improves the factor $k^2 \log |V|$ presented in [CK09, Nut09d]. For the *k-out-connectivity problem*, which is the kRSN problem with $\varrho(v) = k$ for all $v \in V \setminus \{r\}$, there exists a 2-approximation [KR96].

9.4 ILP-based Exact Algorithms for {0,1,2}-SND

In [Rag95], Raghavan developed and analyzed several cut- and flow-based ILP models for the general CON problem and its variants with connected feasible solutions. Further ILPs were suggested in [BMM04]. However, for such general connectivity requirements there are no experimental studies on exact ILP-based algorithms.

In the following we describe the known ILPs for 2CON and 2R(PC)SN in detail as they form the foundation of our own research. The corresponding polytopes for 2CON have been investigated, and different classes of valid inequalities have been derived, in particular in papers by Stoer and Raghavan (with coauthors Grötschel, Monma and Magnanti) [GMS91, GMS92a, GMS92b, MR05]. From now on, we will usually reference their theses [Rag95, Sto92] for simplicity and consistent notations. A survey on polyhedral results regarding this topic can also be found in [KM05a].

9.4.1 ILP Based on Undirected Graphs

2CON. A cut-based ILP for the 2CON problem was first developed by Grötschel, Monma and Stoer [GMS91]. We call this formulation 2CON-UCut. Its central idea is to express the connectivity requirements by considering undirected cuts: To guarantee the connectedness of a solution subgraph, all cuts separating each pair of $\mathcal{R}$ nodes have to contain at least one edge. Furthermore, for each pair of $\mathcal{R}_2$ nodes, the cardinality of all corresponding cuts has to be at least 2. To ensure the 2-node-connectedness for 2NCON instances, we require that all cuts between pairs of $\mathcal{R}_2$ nodes contain at least one edge, when considering any graph resulting from removing a single node.

The 2CON-UCut ILP uses binary variables z_e, for all $e \in E$, that are set to 1 if the corresponding edge is selected into the solution, and to 0 otherwise. In the following we use the shorthand G_w for $G \setminus \{w\} := (V \setminus \{w\}, E \setminus \delta(w))$ for $w \in V$. Recall that, given a variable vector ξ and a set of indices $\mathcal{J}$, we use the shorthand $\xi(\mathcal{J}) := \sum_{j \in \mathcal{J}} \xi_j$.

$$\text{2CON-UCut}: \qquad \min \sum_{e \in E} c(e) \cdot z_e \tag{9.1}$$

$$z(\delta(S)) \geq 1 \qquad \forall S \subset V, \emptyset \neq S \cap \mathcal{R} \neq \mathcal{R} \tag{9.2}$$

$$z(\delta(S)) \geq 2 \qquad \forall S \subset V, \emptyset \neq S \cap \mathcal{R}_2 \neq \mathcal{R}_2 \tag{9.3}$$

$$z(\delta_{G_w}(S)) \geq 1 \qquad \forall w \in V, \forall S \subset V, \emptyset \neq S \cap (\mathcal{R}_2 \setminus \{w\}) \neq \mathcal{R}_2 \setminus \{w\} \tag{9.4}$$

$$z_e \in \{0,1\} \qquad \forall e \in E \tag{9.5}$$

We call (9.2) and (9.3) *undirected cut constraints* and (9.4) *undirected node-cut*. The latter are only included when considering 2NCON, otherwise we solve 2ECON. Grötschel et al. [GMS92b] describe the polyhedral structure of this formulation. In particular, they analyze in which situations the inequalities (9.2)–(9.4) define facets of the corresponding 2CON polytope. Moreover, they introduce further classes of inequalities that may strengthen the corresponding LP relaxations. Among these classes are the *partition*, *node-partition*, and *lifted 2-cover* inequalities, which were later used within effective branch-and-cut algorithm for the 2CON problems [GMS92a]. Note that the partition and node-partition inequalities are generalizations of the undirected cut- and node-cut inequalities. Kerivin et al. [KMN04] enhance the 2CON-UCut ILP with *F-partition* inequalities and include them in their branch-and-cut algorithm, cf. next section. These inequalities are valid when $\varrho(v) \in \{1,2\}$ for all $v \in V$.

When designing a branch-and-cut algorithm, effective separation algorithms for the considered inequality classes are of major importance. The separation problem for the undirected cut- and node-cut inequalities is polynomially solvable. The separation problem for partition, node-partition and lifted 2-cover inequalities is in general NP-hard [GMS92a]. Yet, heuristic separation algorithms for these classes can be found in [GMS92a]. In [KM02,BK04], it was shown that partition inequalities can be separated in polynomial time if $\varrho(v) \in \{1,2\}$ for all $v \in V$. However, these algorithms require $\mathcal{O}(|V|^9)$ and $\mathcal{O}(|V|^7)$ time, respectively, making them unusable

in practice. Heuristic separation routines for (F-)partition inequalities are also presented in [KMN04]. Note that for F-partition inequalities the corresponding complexity is still open.

There also exists an ILP on undirected graphs based on multi-commodity flow (2CON-UFlow) due to Raghavan [Rag95, p. 26], which is equivalent to 2CON-UCut from the polyhedral point of view [Rag95, pp. 85–87]. The idea of 2CON-UFlow is the following: Establishing the connectivity requirement between a pair of customers $s, t \in \mathcal{R}$ can be expressed by sending exactly $\min\{\varrho(s), \varrho(t)\}$ units of flow between them. Therefore, a straightforward idea would be to define the set of commodities $\mathcal{U} = \{(s,t) \mid s,t \in \mathcal{R}\}$. This would require $\mathcal{O}(|\mathcal{R}|^2)$ commodities. However, Raghavan [Rag95, pp. 89–92] showed that $\mathcal{O}(|\mathcal{R}|)$ commodities actually suffice. In fact, therein 2CON-UFlow and the corresponding commodity reduction are given for the more general CON(G,k) problem. We restate it in the context of the 2CON problem:

Consider the following two sets of commodities: The "cyclic" commodity set $\mathcal{C}_2 := \{(v_i, v_{i+1}) \mid 0 \leq i < |\mathcal{R}_2|\}$, whereby $\langle v_1, v_2, v_3, \ldots \rangle$ is an arbitrary ordering of the nodes of $\mathcal{R}_2$ and $v_0 := v_{|\mathcal{R}_2|}$. This commodity set is required to establish the 2-node-connectivity between all $\mathcal{R}_2$ customers. Furthermore, in order to ensure the simple connectivity for the nodes of $\mathcal{R}_1$ we arbitrarily choose a fixed node $r \in \mathcal{R}_2$ and define the set of commodities $\mathcal{C}_1 = \{(r,v) \mid v \in \mathcal{R}_1\}$. A flow of commodity $\chi \in \mathcal{C}_1 \cup \mathcal{C}_2$ on the edge $\{i,j\} \in E$ is modeled by a continuous variable h_{ij}^{χ}. The network-defining variables z are the same as for 2CON-UCut.

$$\text{2CON-UFlow}: \qquad \min \sum_{e \in E} c(e) \cdot z_e \tag{9.6}$$

$$\sum_{i:\{i,v\}\in E} h_{iv}^{\chi} - \sum_{i:\{v,i\}\in E} h_{vi}^{\chi} = \begin{cases} -1, & \text{if } v = s \\ 1, & \text{if } v = t \\ 0, & \text{else} \end{cases} \quad \forall \chi = (s,t) \in \mathcal{C}_1, \forall v \in V \tag{9.7}$$

$$\sum_{i:\{i,v\}\in E} h_{iv}^{\chi} - \sum_{i:\{v,i\}\in E} h_{vi}^{\chi} = \begin{cases} -2, & \text{if } v = s \\ 2, & \text{if } v = t \\ 0, & \text{else} \end{cases} \quad \forall \chi = (s,t) \in \mathcal{C}_2, \forall v \in V \tag{9.8}$$

$$\sum_{i:\{v,i\}\in E} h_{vi}^{\chi} \leq 1 \qquad \forall \chi = (s,t) \in \mathcal{C}_2, \; v \in V \setminus \{s,t\} \tag{9.9}$$

$$0 \leq h_{vw}^{\chi} \leq z_{vw} \qquad \forall \{v,w\} \in E, \forall \chi \in \mathcal{C}_1 \cup \mathcal{C}_2 \tag{9.10}$$

$$z_e \in \{0,1\} \qquad \forall e \in E \tag{9.11}$$

We call the above formulation 2NCON-UFlow if the inequalities (9.9) are included in the model and 2ECON-UFlow otherwise.

Theorem 9.7 ([Rag95])**.** 2CON-UCut *and* 2CON-UFlow *are equivalent from the polyhedral point of view.*

2R(PC)SN. Wagner et al. [WRP+06, WRP+07] gave an ILP for 2R(PC)SN. Thereby, they use the term "directed cuts" when describing their formulation. How-

ever, we can show that this formulation is equivalent to the traditional undirected approach. We show this in the next section where we discuss orientation-based ILPs.

Further ILP-based algorithms were given for the $b_{\max}$-SND problem, which is the 2RPCSN problem where the connectivity requirement for $\mathcal{R}_2$ customers is relaxed as follows: for each node $v \in \mathcal{R}_2$ there exists a path of length at most $b_{\max}$ to some node which has two disjoint paths to the root node r. For such problems, several path-based ILP formulations can be found in [LR08, LRP09, LR10].

9.4.2 Orientation-based ILPs

Edge-connectivity. We recall that an orientation of an undirected graph G is a directed graph $\hat{G}$ that is obtained by transforming each edge of G into a directed arc. For 2ECON, it was shown by Chopra [Cho92] and Raghavan [Rag95] that considering a certain orientability property of feasible solutions leads to ILP formulations that are polytope-wise stronger than the undirected formulations mentioned above. These ILPs exploit Theorem 2.7, i.e., Robbin's characterization of the 2-edge-connected graphs: An undirected graph $G' = (V', E')$ is 2-edge-connected if and only if there exists an orientation $\hat{G}'$ of G' such that for every pair of nodes $u, v \in V'$ there are directed paths $(u \to v)$ and $(v \to u)$ in $\hat{G}'$.

Optimal solutions to the 2ECON problem always have the following structure: they consist of exactly one non-trivial 2-edge-connected component that contains all $\mathcal{R}_2$ nodes and there may be several trees attached to this component. Using Theorem 2.7, one can easily see that every such network can be oriented as follows: For every pair of nodes $u, v \in \mathcal{R}_2$ there is a directed path from u to v. Furthermore, if we choose a root node $r \in \mathcal{R}_2$, there is a directed path from r to w for every node $w \in \mathcal{R}_1$. Vice versa, each network that can be oriented in such a way is a feasible solution to the 2ECON problem.

Thus, the input of the 2ECON problem can be transformed into the following equivalent problem: Let $\bar{G} = (V, A)$ be the bidirection of G with costs $c((u, v)) = c((v, u)) = c(\{u, v\})$. We search for a cost-minimum subgraph of $\bar{G}$ that has the above orientation properties. We rephrase the ILP formulation due to Chopra using our notations. Therefore, we introduce binary variables x_{ij} that are 1, if the solution network contains the arc $(i, j) \in A$ and 0 otherwise. Furthermore, we fix some arbitrary node $r \in \mathcal{R}_2$.

$$\text{2ECON-DCut}_{\text{C}}: \qquad \min \sum_{a \in A} c(a) \cdot x_a \tag{9.12}$$

$$x_{vw} + x_{wv} \leq 1 \qquad \forall \{v, w\} \in E \tag{9.13}$$

$$x(\delta^-(S)) \geq 1 \qquad \forall S \subseteq V, \emptyset \neq S \cap \mathcal{R}_2 \neq \mathcal{R}_2 \tag{9.14}$$

$$x(\delta^-(S)) \geq 1 \qquad \forall S \subseteq V \setminus \{r\}, S \cap \mathcal{R}_1 \neq \emptyset \tag{9.15}$$

$$x_a \in \{0, 1\} \qquad \forall a \in A \tag{9.16}$$

Observation 9.8. *In order to obtain the orientation described above it suffices to require a directed path from some root node $r \in \mathcal{R}_2$ to every other node in $\mathcal{R} \setminus r$ and a directed path $(v \to r)$ for every node $v \in \mathcal{R}_2$.*

The above observation allows us to formulate an orientation-based ILP based on directed cuts which uses less inequalities than 2ECON-DCUT$_C$. From now on we will use the shorthands $\mathcal{R}'_i := \mathcal{R}_i \setminus \{r\}$ for $0 \leq i \leq 2$, and $\mathcal{R}' := \mathcal{R} \setminus \{r\}$.

$$\text{2ECON-DCUT}: \qquad \min \quad \sum_{a \in A} c(a) \cdot x_a \tag{9.17}$$

$$x_{vw} + x_{wv} \leq 1 \qquad \forall \{v, w\} \in E \tag{9.18}$$
$$x(\delta^-(S)) \geq 1 \qquad \forall S \subseteq V \setminus \{r\}, S \cap \mathcal{R}' \neq \emptyset \tag{9.19}$$
$$x(\delta^+(S)) \geq 1 \qquad \forall S \subseteq V \setminus \{r\}, S \cap \mathcal{R}'_2 \neq \emptyset \tag{9.20}$$
$$x_a \in \{0, 1\} \qquad \forall a \in A \tag{9.21}$$

Chopra showed that 2ECON-DCUT is strictly stronger than 2ECON-UCUT even if the latter is enhanced with partition inequalities. In fact, the partition inequalities are already induced by the directed cut inequalities. It was shown by Stoer [Sto92] that partition inequalities can be facet-defining for 2CON problems under certain assumptions. This suggests that the LP relaxation of 2ECON-DCUT may deliver quite good lower bounds.

In [KMN04] it is observed that if $V = \mathcal{R}_2$, the partition inequalities are already induced by the undirected cut inequalities. In fact, Raghavan showed that under these circumstances the undirected and the directed formulations are equivalent for 2ECON, cf. Theorem 9.11.

Raghavan [Rag95] stated a 2ECON formulation based on directed multi-commodity flow, which he showed to be equivalent to 2ECON-DCUT. Again, a naive idea would be to create a commodity for each pair of nodes $s, t \in \mathcal{R}_2$. However, considering Observation 9.8, less commodities are sufficient. Consider the set of commodities $\mathcal{C} = \{(r, v) \mid v \in \mathcal{R}'\} \cup \{(v, r) \mid v \in \mathcal{R}'_2\}$. The directed flow of a commodity $\chi \in \mathcal{C}$ on the arc $(i, j) \in A$ is modeled by a continuous variable f^χ_{ij}. We furthermore use the network-defining variables x as for 2ECON-DCUT.

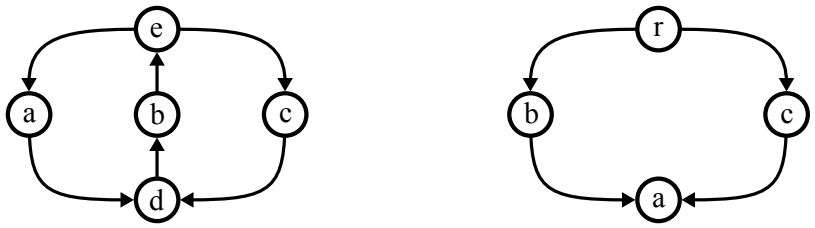
$$\text{2ECON-DFLOW}: \qquad \min \sum_{a \in A} c(a) \cdot x_a \tag{9.22}$$

$$\sum_{i:(i,v)\in A} f^\chi_{iv} - \sum_{i:(v,i)\in A} f^\chi_{vi} = \begin{cases} -1, & \text{if } v = s \\ 1, & \text{if } v = t \\ 0, & \text{else} \end{cases} \qquad \forall \chi = (s, t) \in \mathcal{C}, \forall v \in V \tag{9.23}$$

$$x_{vw} + x_{wv} \leq 1 \qquad \forall \{v, w\} \in E \tag{9.24}$$
$$0 \leq f^\chi_a \leq x_a \qquad \forall a \in A, \forall \chi \in \mathcal{C} \tag{9.25}$$
$$x_a \in \{0, 1\} \qquad \forall a \in A \tag{9.26}$$

The proofs of the following theorems can be found in [Rag95, pp.162–163] where they were given in the context of the CON(G, k) problem.

Theorem 9.9. 2ECON-DCUT *and* 2ECON-DFLOW *are equivalent.*



(a) (b)

Figure 9.1: Examples of infeasible orientations, cf. text.

Theorem 9.10. *Consider general connectivity requirements* $\varrho \in \{0,1,2\}^{|V|}$. *Then* 2ECON-DCut *and* 2ECON-DFlow *are strictly stronger than* 2ECON-UCut *and* 2ECON-UFlow.

Theorem 9.11. *If* $\varrho(v) \in \{0,2\}^{|V|}$, *i.e.,* $\mathcal{R}_1 = \emptyset$, 2ECON-DCut, 2ECON-DFlow, 2ECON-UCut, *and* 2ECON-UFlow *are all equivalent.*

Node-connectivity. Prior to our solution, finding an orientation-based formulation for {0,1,2}-SND problems with node-connectivity requirements was a long-standing open problem, see, e.g., [Rag95, p. 183] and [Sto92, pp. 32,134]. The main hindrance why such formulations were not known before was that there were no suitable characterizations of the 2-node-connected graphs that could be used in this context. Note that extending Theorem 2.7 in a natural way for these graphs, i.e., requiring node-disjoint directed paths between each pair of the graph nodes, does not yield a valid characterization: Assume that such an orientation would always exist, and consider the shadow of the directed graph shown in Figure 9.1(a). In order to obtain two node-disjoint paths between the nodes a and b, we have only two (symmetric) possibilities to orient the edges $\{a,d\},\{d,b\},\{b,e\},\{e,a\}$. Choosing any of these orientations, we have only one possible orientation of the edges $\{e,c\},\{c,d\}$, as we have to guarantee node-disjoint paths $(b \to c)$ and $(c \to b)$. Yet, the obtained orientation does not contain the required node-disjoint paths $(a \to c)$ and $(c \to a)$.

For 2RPCSN, Wagner et al. tried to give an orientation-based formulation for 2RPCSN [WRP+07]. Similar to 2ECON-DCut they considered the bidirected graph $\bar{G}$. The idea was to characterize feasible solutions of the 2RPCSN problem by subgraphs of $\bar{G}$ where every chosen customer $v \in \mathcal{R}_2$ has two directed paths $(r \to v)$. Figure 9.1(b) shows that such orientations do not actually describe all feasible solutions to a 2RPCSN problem: The shadow of the shown graph represents a feasible solution of a 2RPCSN problem. The drawn arcs represent the only possibility of orienting the graph edges in order to connect the node a in the required way. This prohibits the nodes b and c to be connected correctly. Hence, in order to ensure feasibility, the authors permit solutions, where both directions (v,w) and (w,v) are allowed simultaneously. Thus, this formulation is not stronger but equivalent to an undirected one and furthermore requires twice as many variables.

Raghavan [Rag95, pp. 180–181] suggested the strongest known formulation for 2NCON. Thereby, he strengthened the 2NCON-UFlow formulation by integrating

central ideas of 2ECON-DFlow. We denote Raghavan's formulation by 2NCON-MFlow (mixed flow). It uses two multi-commodity flows g and h simultaneously: g represents directed flow for the induced 2ECON problem, whereas h represents an non-oriented flow with node-disjointness constraints.[2] The two flows are bound to each other only by their common use of the z_e variables. Raghavan also posed the questions whether there is a way to link g and h more directly and whether such linking would strengthen the formulation. In this thesis, we positively answer both these questions.

2NCON-MFlow uses the commodity sets $\mathcal{C}_2$ and $\mathcal{C}$ as defined for 2CON-UFlow and 2ECON-DFlow, respectively:

$$\text{2NCON-MFlow}: \qquad \min \sum_{e \in E} c(e) z_e \tag{9.27}$$

$$\sum_{(i,v)\in A} g^{\chi}_{iv} - \sum_{(v,i)\in A} g^{\chi}_{vi} = \begin{cases} -1, & \text{if } v = s \\ 1, & \text{if } v = t \\ 0, & \text{else} \end{cases} \quad \forall \chi = (s,t) \in \mathcal{C}, \forall v \in V \tag{9.28}$$

$$g^{\chi}_{vw} + g^{\chi'}_{wv} \leq z_{vw} \qquad \forall \{v,w\} \in E, \forall \chi, \chi' \in \mathcal{C} \tag{9.29}$$

$$g^{\chi}_{vw} \geq 0 \qquad \forall (v,w) \in A, \forall \chi \in \mathcal{C} \tag{9.30}$$

$$\sum_{(i,v)\in A} h^{\chi}_{iv} - \sum_{(v,i)\in A} h^{\chi}_{vi} = \begin{cases} -2, & \text{if } v = s \\ 2, & \text{if } v = t \\ 0, & \text{else} \end{cases} \quad \forall \chi = (s,t) \in \mathcal{C}_2, \forall v \in V \tag{9.31}$$

$$0 \leq h^{\chi}_{vw} \leq z_{vw} \qquad \forall (v,w) \in A, \forall \chi \in \mathcal{C}_2 \tag{9.32}$$

$$\sum_{(v,i)\in A} h^{\chi}_{vi} \leq 1 \qquad \forall \chi = (s,t) \in \mathcal{C}_2, v \in V \setminus \{s,t\} \tag{9.33}$$

$$z_e \in \{0,1\} \qquad \forall e \in E \tag{9.34}$$

9.5 Experimental Studies for $\{0,1,2\}$-SND Problems

In this section we point out papers where different solution methods are experimentally compared and give an overview over the used test instances for the {0,1,2}-SND problems. We thereby do not report on the running times of the different algorithms: In the course of the last 20 years considerable hard- and software improvements have been achieved, rendering direct comparison across different papers meaningless. As we already mentioned before, no computational study exists on any approximation algorithm for survivable network design. Therefore, we only report on experimental studies for heuristics and exact algorithms.

Heuristics. For the 2NCON problem with $\varrho(v) \in \{1,2\}$, Monma and Shallcross [MS89] tested their heuristics on 20 randomly generated dense graphs with Euclidean edge weights, containing up to 200 nodes. Furthermore, they applied

[2] Note that this formulation has been developed for general kNCON problems, where it is called *improved undirected flow formulation with node-disjointness constraints*.

their algorithm on 3 real-world sparse graphs from [Sto92]. Stoer [Sto92] applied these heuristics to further real-world instances to find good upper bounds for her exact algorithm. Thereby, the relative gap between the heuristic solution and the optimal solution was always below 1,5%.

Ghashghai and Rardin [GR02] tested their genetic algorithm for the 2CSN problem on 6 complete graphs with 40 nodes and Euclidean costs. They tried these instances with selecting 30% and 50% of the nodes into $\mathcal{R}_2$. The above papers do not compare their heuristics to any other algorithms.

For the 2CSS problem, Clark and Anandalingam [CA95] compared their bootstrap heuristics with those of [MS89]. They used a large set of complete graphs with up to 200 nodes. Unfortunately, the considered edge weights are not specified. The authors claim that their heuristics lead to better solution quality at the cost of higher running times.

For 2ECON, Raghavan [Rag95] presented a computational study of his dual-ascent algorithm that is based on the 2ECON-DFlow ILP. He used random graphs with up to 300 nodes and 3000 edges with Euclidean costs and also some generated test problems with up to 120 nodes and 180 edges similar to the telecommunication data used in [Sto92]. As the algorithm produces both lower and upper bounds for the 2ECON problem, the quality of the algorithm can be judged by considering the running times and relative gaps between these bounds.

Exact algorithms. For 2ECON, computational studies were conducted for exact algorithms using both undirected and orientation-based ILPs. For 2NCON, no effective orientation-based ILP was known except for the 2NCON-MFlow. The latter ILP, however, has never been implemented. Hence, all known experiments for 2NCON use ILPs based on undirected graphs. The only relevant computationaly study for this problem was published in [Sto92] and for the 2R(PC)SN problem in [WRP$^+$06, WRP$^+$07].

Raghavan [Rag95] used the 2ECON-DFlow ILP in order to solve small 2ECON problems to optimality. Thereby he did not use column generation or similar methods to reduce the running times. Using CPLEX 2.1 on a Sun SPARCstation 10 the largest solvable graphs (within a half an hour) contained 40 nodes.

Also for the 2ECON problem, Chopra [Cho92] tested his branch-and-cut algorithm for the orientation-based ILP on several random sparse and complete graphs with up to 100 nodes and both random and Euclidean edge weights. Furthermore, neither preprocessing routines nor additional strengthening inequalities were used. The algorithm solved all instances to provable optimality. The author also notes that the running time of his algorithm decreases, if the instance does not contain $\mathcal{R}_0$ nodes. The solution of the LP relaxation was already optimal a significant number of times, thus emphasizing the strength of the corresponding formulation. The relative optimality gap averaged over all instances was 0.1%.

Stoer [Sto92] presented computational results on a branch-and-cut algorithm for 2CON problems based on 2CON-UCut. The algorithm was tested on 7 real world instances provided by Bell Communication Research. The input graphs were rather sparse containing up to 116 nodes and 173 edges. Only one of these instances also contained $\mathcal{R}_0$ nodes. Moreover, the structure of the instances allowed preprocessing

routines to reduce the size of the input graphs to at most 39 nodes and 86 edges. The authors used heuristic methods to separate partition, node-partition and 2-cover inequalities. Although the algorithm was tested using quite old software, all above instances could be solved to optimality. Note that Chopra's algorithm [Cho92] had comparable running times on sparse instances of the same size. The algorithm was also tested on some random (not further specified) instances. As the density of these graphs was significantly higher, the algorithm did not perform as well as on the real-world data. Furthermore, the running time dramatically increased when the number of $\mathcal{R}_0$ nodes increased.

Kerivin et al. [KMN04] tested complete graphs with up to 574 nodes from the benchmark library TSPLIB [TSP]. The edge costs are Euclidean and therefore satisfy the triangle inequality. The authors give results for 2CSS problems with both edge- and node-connectivity constraints. Yet, note that the 2ECSS and 2NCSS problems are equivalent on the above problem input, as we mentioned in Theorem 9.6. Therefore, the performance of the algorithm on these two problems is identical. Also note that for such instances the undirected formulation is equivalent to the directed one due to Theorem 9.11. Therefore, for such input graphs the undirected formulation will always give better practical results, as the size of the undirected ILP is smaller. Kerivin et al. also used the above graphs for the 2ECON problem with $\varrho(v) \in \{1, 2\}$. However, for this problem variant the size of the input graphs was restricted to 101 nodes. They used 2ECON-UCut and separated the partition and F-partition inequalities heuristically. All instances were solved to optimality, and the authors figure out that partition and F-partition inequalities significantly improve the obtained lower bounds from the LP relaxation, although only few F-partition constraints were separated. However, it is not clear whether separating these constraints also speeds up the computation, as the running times without separating them are not given.

Experiments for 2R(PC)SN were conducted in [WRP+06, WRP+07]. The considered instances are based on real-world access net data of the city district Cologne-Ossendorf. We thoroughly describe these instances in Section 12 in more detail.

9.6 Our Contribution

As stated in Section 9.4.2, a strong orientation-based ILP for {0,1,2}-SND problems with node-connectivity constraints was not known before. The central result of this part of the thesis is the development of such a formulation. To this ends we provide a characterization of 2-node-connected graphs via rooted orientation properties. This graph-theoretical result allows us to derive two classes of orientation-based ILP formulations for 2RSN, 2RPCSN and 2NCON: DFlow and DCut. We prove the theoretical advantages of these directed models compared to the previously known ILP approaches. On the other hand, we show that our two concepts are equivalent from the polyhedral point of view. Nonetheless, our experimental study shows that the cut formulation is much more powerful in practice. Based on DCut, we develop a branch-and-cut algorithm for {0,1,2}-SND problems which allows us to solve test instances with up to 4900 nodes to provable optimality.

As we could see in the previous section, till now there is no common benchmark

set in order to test algorithms for {0,1,2}-SND problems. Most of the previously used instances are not directly available. Although randomized routines for generating problem instances are sometimes precisely described, it is not sufficient for a fair comparison with other algorithms for the same problem. In this thesis, we propose TSNDLib [TSN08], a collection of benchmark instances for {0,1,2}-SND problems and encourage other researchers to use them for this type of problems.

The only relevant experimental study on exact algorithms for the 2NCON problem is published in [Sto92]. However, the size of the thereby considered instances did not exceed 40 nodes (after applying preprocessing routines). Hence, our computational study is the first for 2NCON conducted on a rich and diverse benchmark library that contains instances with up to 4900 nodes. Unfortunately, the code of the branch-and-cut algorithms based on the 2CON-UFLOW (developed in [Sto92] and [KMN04]) were not available for us and we were not able to evaluate their performance for 2NCON on our benchmark instances. For the 2(PC)SN problem, we compare the perfomance of our algorithm with those of [WRP$^+$06, WRP$^+$07].

Although the main focus of our work lies on the node-connectivity aspect of {0,1,2}-SND problems, combining our results with the results on 2ECON in [Cho92, Rag95], we were for the first time able to develop a common branch-and-cut framework based on graph orientation that solves 2ECON, 2NCON, and 2R(PC)SN problems.

Chapter 10

Orientation-based Modeling

In Sections 10.1 and 10.2 we describe and characterize the particular structure of feasible solutions to the 2NCON and 2RSN problems. The thereby induced novel characterization of general 2-node-connected graphs is given in Section 10.3. We use the latter to model two novel cut- and flow-based ILPs for the {0,1,2}-SND problems in Section 10.4. A polyhedral analysis of our models and a comparison to the previously known ILPs is presented in Section 10.5.

10.1 Solution Structure

Exploiting knowledge about the particular structure of feasible solutions can help to design algorithms that are more effective in practice than the previous ones. We therefore analyze networks that are feasible for our problems. For this purpose we need some basic definitions and notations.

Definition 10.1. Let (G', U) be a tuple of an undirected connected graph $G' = (V', E')$ and $U \subseteq V'$ with $|V'| \geq 3$ and $|U| \geq 2$. G' with respect to U is:

- $(1,2)$-*edge-connected*, if all nodes $u \in U$ lie in the same 2-edge-connected component.
- $(1,2)$-*root-node-connected*, if for a given root node $r \in V'$ each node $u \in U$ belongs to a block that also contains r.
- $(1,2)$-*node-connected*, if all nodes $u \in U$ lie in the same block.

Observation 10.2. *Let (G', U) be a tuple as defined above.*

- *If G' is $(1,2)$-root-node-connected w.r.t. U, then it is also $(1,2)$-edge-connected w.r.t. U.*
- *G' is $(1,2)$-node-connected w.r.t. U if and only if G' is $(1,2)$-root-node-connected w.r.t. U, for all possible choices of $r \in U$.*

The following observation allows us to identify feasible 2ECON, 2RSN and 2NCON networks with the notations given in Definition 10.1.

Observation 10.3. *Given an instance of a $\{0,1,2\}$-SND problem with the customer set $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$. Let $N = (V_N, E_N)$ be any feasible solution of the 2ECON, 2RSN, or 2NCON problem. We can observe that $\mathcal{R} \subseteq V_N$ and $(N, \mathcal{R}_2)$ is $(1,2)$-edge-connected, $(1,2)$-root-node-connected, or $(1,2)$-node-connected, respectively.*

10.2 Orientation Theorems

10.2.1 2ECON – Excursion

Due to the existence of Theorem 2.7, it was possible to give the characterization of feasible 2ECON networks [Cho92, Rag95], i.e., $(1,2)$-edge-connected graphs with respect to $\mathcal{R}_2$. In order to give a better insight into the similarities and differences between the problem with edge-connectivity (2ECON) and our node-connectivity problems 2RSN and 2NCON, we rephrase this result using our own notations and definitions.

Theorem 10.4. *Let (G', U) be a tuple as defined above. G' is $(1,2)$-edge-connected with respect to U if and only if for any node $r \in U$ there exists an orientation $\hat{G}$ such that:*

(P1) For each node $v \in V' \setminus U$, $\hat{G}$ contains a directed path $(r \to v)$.

(P2) For each node $v \in U \setminus \{r\}$, $\hat{G}$ contains a directed path $(r \to v)$ and a directed path $(v \to r)$.

The above equivalence between $(1,2)$-edge-connected graphs and graph orientations with the properties (P1) and (P2) allows us to transform 2ECON into an equivalent problem defined on directed graphs (cf. Section 9.4.2).

We now characterize all feasible solutions to 2RSN and 2NCON, i.e., graphs that contain $\mathcal{R}_1$ and are $(1,2)$-root-node-connected or $(1,2)$-node-connected w.r.t. $\mathcal{R}_2$, respectively.

10.2.2 2RSN

Theorem 10.5. *For a given tuple (G', U) as defined above and a root node $r \in V'$, G' is $(1,2)$-root-node-connected with respect to U if and only if there exists an orientation $\hat{G}$ of G' that satisfies the properties (P1), (P2) and:*

(P3) For each node $v \in U \setminus \{r\}$ the directed paths $(v \to r)$ and $(r \to v)$ are node-disjoint except for r and v.

Indeed, given a valid orientation, it is trivial to show that (G', U) is $(1,2)$-root-node-connected. Hence, in order to prove the theorem we have to show that if (G', U) is $(1,2)$-root-node-connected then there exists a valid orientation $\hat{G}$ of G'. Before doing this we define an orientation procedure for a general 2-node-connected graph $B = (V_B, E_B)$ and a given root node $r_B \in V_B$. The following procedure basically consists of identifying and orienting an open ear decomposition $P_0, \ldots, P_l$, where P_0 and P_1 form a cycle containing r_B. Recall that the concept of open ear decomposition for the 2-node-connected graphs was discussed in Section 2.1.

Nevertheless, to make the proof of Theorem 10.5 selfcontained, we also prove the existence of such a decomposition and the correctness of our orientation procedure.

Definition 10.6 (Orientation Procedure)**.** We use $\ell : V' \to [0,1] \cup \{\infty\}$ as a labeling function and call a node $v \in V$ *labeled* if $\ell(v) < \infty$. Initially, we set $\ell(v) := \infty$ for all $v \in V$. We start by identifying a cycle Z in B containing r_B, and orient its edges consistently in one of the two possible directions. We then label each node on Z with increasing fractional numbers between 0 and 1, according to this orientation, starting with $\ell(r_B) := 0$. Hence, all edges of Z (except its last edge $\hat{e}$) are oriented from the smaller towards the larger label number. We will now orient the remaining undirected edges in such a way that this invariant is valid for all oriented edges:

We define an *augmenting path* $P = [a \to b]$ as a simple path of unoriented edges where only the disjoint start and end nodes are labeled, and $\ell(a) < \ell(b)$. To orient B, we repeatedly find an augmenting path $P = [a \to b]$ and orient it from a to b, labeling all inner nodes with increasing fractional numbers greater than $\ell(a)$ but smaller than $\ell(b)$; these labels are to be unique over all labelings so far. The procedure terminates as soon as no augmenting path can be found in B.

Lemma 10.7. *The above orientation procedure, if applied to a 2-node-connected graph B, produces an orientation $\hat{B}$ of B, i.e., all edges of B are uniquely oriented, and $\hat{B}$ satisfies: For each node $v \in B$ there are node-disjoint paths $(r_B \to v)$ and $(v \to r_B)$.*

Proof. Assume that at some point there is at least one unoriented edge e left, but we cannot find any augmenting path. Clearly, e has to be part of some path $Q = [c \to d]$ of unoriented edges with labeled nodes c and d. Since neither Q nor its reversal is an augmenting path, we have $\ell(c) = \ell(d)$ and therefore $c = d$, i.e., Q is a cycle of unoriented edges, and none of its nodes except for c are labeled. Since B is 2-node-connected, there has to be an additional unoriented path from some node $q \in Q$ to some labeled node p ($p, q \neq c$). But then, the path $[p \to q \to c]$ (or its reversal) would be an augmenting path, which is a contradiction.

By the above construction, we guarantee that each labeled node has at least one incoming and one outgoing edge. Furthermore, each oriented edge (except for $\hat{e}$) is oriented from the smaller towards the larger label number. Hence, each oriented path will always contain monotonously increasing label numbers (with the exception of $\hat{e}$). This means that any directed circle starting from r_B and going through any labeled node v will be simple, and we therefore have node-disjoint paths $(r_B \to v)$ and $(v \to r_B)$. □

The above orientation procedure produces only one edge that is oriented from the larger label number to the smaller one. As $\ell(r) < \ell(v)$ for each $v \in V_B \setminus \{r\}$ we have the following important observation:

Observation 10.8. *The orientation procedure guarantees that there is only a single edge $\hat{e}$ that is directed towards r.*

We are now ready to prove the Theorem 10.5:

Proof. (of Theorem 10.5) Let B_i, $i \in I$, be the blocks of G' with $B_i \cap U \neq \emptyset$. For each $i \in I$ we have $r \in B_i$ by definition and clearly $U \subseteq \bigcup_{i \in I} B_i$. We orient each B_i according to the orientation procedure defined above. Due to Lemma 10.7 the resulting orientation satisfies properties (P1), (P2) and (P3) for all nodes $v \in \bigcup_{i \in I} B_i$.

The nodes $v \notin \bigcup_{i \in I} B_i$ form subgraphs attached to $\bigcup_{i \in I} B_i$. Note that whenever G' represents an optimal solution to the 2RSN problem we can assume these subgraphs to be trees. If we orient these subgraphs using DFS away from the corresponding cut nodes, we obtain directed paths $(r \to v)$ for all $v \in V \setminus \{r\}$ which concludes the construction of $\hat{G}$. □

10.2.3 2NCON

We now characterize feasible solutions to the 2NCON problem, i.e., $(1, 2)$-node-connected graphs with respect to $\mathcal{R}_2$.

Theorem 10.9. *Let (G', U) be a tuple as described above. G' is $(1, 2)$-node-connected with respect to U if and only if for any root node $r \in U$ there exists an orientation $\hat{G}$ of G' with the properties (P1)–(P3) and additionally:*

(P4) The in-degree of r is equal to one.

Proof. Let $\hat{G}$ be a valid orientation of G' with respect to some root node $r \in U$. We first show that (G', U) is $(1, 2)$-node-connected. Obviously, (G', U) is (1,2)-root-node-connected for r. Therefore, each node $v \in U$ belong to some block B_i that also contains r. Assume that there are at least two different blocks B_1 and B_2 containing the nodes $v_1, v_2 \in U \setminus \{r\}$, respectively. Then r has to be a cut node contained in both blocks. But since a valid orientation requires directed paths $(v_1 \to r)$ and $(v_2 \to r)$ there have to be at least two edges being directed towards r which is a contradiction to the validity of the orientation $\hat{G}$. Hence we know that all nodes $u \in U$ lie in the same block, i.e., G' is (1,2)-node-connected.

Now, assume (G', U) is $(1, 2)$-node-connected, and we show that there exists a valid orientation $\hat{G}$ of G'. By definition, all nodes $u \in U$ lie in the same block. According to Observation 10.2, for any chosen root $r \in U$ it is also $(1, 2)$-root-node-connected with respect to $U \setminus \{r\}$. Using the orientation procedure above for any arbitrarily chosen root $r \in U$, we obtain an orientation satisfying (P1)–(P3).

As there is only one single block containing all nodes $v \in U$, the orientation procedure guarantees that r has only one ingoing arc (cf. Observation 10.8). □

10.3 Orientations of 2-Node-Connected Graphs

10.3.1 Our Characterization

The orientation procedure defined in the previous section actually gives us a characterization of general 2-node-connected graphs.

Theorem 10.10. *An undirected graph $G' = (V', E')$ is 2-node-connected if and only if for an arbitrary chosen root node $r \in V'$ there exists an orientation $\hat{G}$ such*

that the in-degree of the root node is exactly 1 and for each node $v \in V' \setminus \{r\}$, $\hat{G}$ contains a directed path $(r \to v)$ and a directed path $(v \to r)$ that are node-disjoint except for r and v.

Proof. Lemma 10.7 and Observation 10.8 guarantee that applying our orientation procedure to a 2-node-connected graph G' gives a valid orientation.

On the other hand, given a valid orientation $\hat{G}$, we show that the underlying graph G' is 2-node-connected. For any pair of nodes $u, w \in V'$ there exists a path $(u \to w)$ that is, e.g., the concatenation of directed paths $(u \to r)$ and $(r \to w)$ and analogously a backward path $(w \to u)$. Due to Theorem 2.7 the underlying undirected graph G' is hence 2-edge-connected and does not contain any bridges. We show that all nodes $v \in V' \setminus \{r\}$ share a common block with r. Assume that there are at least two blocks B_1 and B_2 containing the nodes $v_1, v_2 \in V' \setminus \{r\}$, respectively. Then r has to be a cut node contained in both blocks. But since a valid orientation requires directed paths $(v_1 \to r)$ and $(v_2 \to r)$ there have to be at least two edges being directed towards r which is a contradiction to the validity of the orientation $\hat{G}$. Hence we know that $\hat{G}$ will only contain a single non-trivial block and therefore it is 2-node-connected. □

10.3.2 s,t-Orientation: A Different Approach?

There is another well-known orientation-based characterization of 2-node-connected graphs. Recall Theorem 2.9 stated in Section 2.1: A given graph G' is 2-node-connected if and only if for any edge $\{s,t\} \in E'$ there exists an s,t-orientation, i.e., a directed acyclic graph, where s is the only source and t is the only sink. Such an orientation can be obtained, e.g., from an open ear decomposition of G' that starts with the edge $\{s,t\}$. Indeed, for a fixed choice of $\{s,t\} = \{r,v\}$ for some $v \in V'$, the corresponding construction is very similar to our orientation procedure, with the only difference that $\{s,t\}$ is oriented from s to t, and not from t to s as in $\hat{G}$. However, this characterization could not be efficiently used before as the basis of orientation-based ILPs for the 2NCON problem, as it requires the knowledge of at least one edge that is contained in all 2NCON solutions. Meanwhile, our characterization is constructed such that it is directly applicable in our context: it chooses a reference node r and orients the graph with respect to it. Choosing r among the $\mathcal{R}_2$ nodes guarantees that r is contained in all 2NCON solutions.

10.4 ILP Modeling

We transform the input of 2RSN and 2NCON problems into an equivalent input for the *directed 2-root-connected Steiner network (D2RSN)* and the *directed {0,1,2}-node-connected Steiner network problem (D2NCON)* problems, respectively, and provide novel directed cut and directed flow based formulations. Finally, we modify the former model for some prize-collecting problem variants.

Both D2RSN and D2NCON problems can be defined on any directed graph D. However, in the context of this thesis, we always deal with the input $(\bar{G}, \bar{c}, r, \mathcal{R}_1, \mathcal{R}_2)$ where $\bar{G} = (V, A)$ is the bidirection of G with costs $\bar{c}((u,v)) = \bar{c}((v,u)) := c(\{u,v\})$.

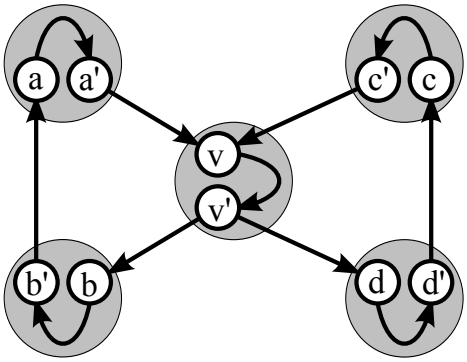


Figure 10.1: Counterexample to modeling 2-node-connectivity via 2-edge-connectivity, cf. text.

If we deal with 2NCON, the root r is an arbitrary chosen $\mathcal{R}_2$ node. Recall that we already defined $\mathcal{R}'_i := \mathcal{R}_i \setminus \{r\}$ for $0 \leq i \leq 2$, and $\mathcal{R}' := \mathcal{R} \setminus \{r\}$.

The optimal solution to D2RSN on the instance $(\bar{G}, \bar{c}, r, \mathcal{R}_1, \mathcal{R}_2)$ is a weight-minimal oriented subgraph $\hat{N} = (V_N, A_N)$ in $\bar{G}$ with $\mathcal{R} \subseteq V_N$ that is $(1,2)$-root-node-connected—i.e., it satisfies the properties (P1)–(P3)—with respect to r and $U = \mathcal{R}_2$. Analogously, if we require $(1,2)$-node-connectedness—i.e., properties (P1)–(P4)—then $\hat{N}$ is an optimal solution to D2NCON. Note that if we ignore both (P3) and (P4), $\hat{N}$ is an optimal solution to the D2ECON problem.

From Theorems 10.5 and 10.9 we have:

Corollary 10.11. *Any feasible 2RSN and 2NCON solution for $(G, c, [r], \mathcal{R}_1, \mathcal{R}_2)$ can be transformed into a corresponding feasible D2RSN and D2NCON solution for $(\bar{G}, \bar{c}, r, \mathcal{R}_1, \mathcal{R}_2)$, respectively, with the same objective value, and vice versa.*

Modeling 2-node-connectivity via 2-edge-connectivity? One may try to model node-connectivity by only computing edge-connectivity in a modified underlying graph. In related problems it is often possible to split each node v into a pair of nodes v, v', connected via an arc (v, v'). Then, each undirected edge (u, v) is transformed into two directed arcs (u', v), (v', u). We give a counterexample to this strategy for 2NCON. Assume a complete graph on five nodes $\{a, b, c, d, v\}$ with two distinct edge cost values: all edges incident to v, as well as the edges $\{a, b\}$ and $\{c, d\}$ are cheap, whereas the other edges are expensive. Figure 10.1 shows an optimal directed solution w.r.t. 2-edge-connectivity in this so-modified graph. Observe that the solution's shadow is in fact 2-node-connected. Yet, when transforming the solution back into the original graph, we have to merge v, v', and v becomes a cut node. This renders the solution infeasible for 2NCON.

10.4.1 DFlow

DFlow is a multi-commodity-based ILP that is be able to solve 2ECON, 2NCON and 2RSN problems. We first formulate it for 2RSN. We then extend it with a

single additional inequality in order to solve 2NCON and also show how to use it for 2ECON.

Connecting each customer $v \in \mathcal{R}'$ to a root node r can be expressed by sending exactly one unit of flow from r to v in $\bar{G}$. To guarantee the redundant connection for each customer $v \in \mathcal{R}'_2$, we send one unit of flow back to the root. Thereby it has to be ensured that the pairs of forward and backward flows do not use common nodes and edges except for v and r. The arcs with a positive amount of flow then define our solution network.

We use the same set of variables and commodities as for 2ECON-DFlow: Binary variables x_{ij} are 1, if the solution network contains the arc $(i,j) \in A$ and 0 otherwise. The commodity set is $\mathcal{C} = \{(r,v) \mid v \in \mathcal{R}'\} \cup \{(v,r) \mid v \in \mathcal{R}'_2\}$. A flow of commodity $\chi \in \mathcal{C}$ on the arc $(i,j) \in A$ is modeled by the continuous variable f_{ij}^{χ}. Recall that, given a variable vector ξ and a set of indices $\mathcal{J}$, we use the shorthand $\xi(\mathcal{J}) := \sum_{j \in \mathcal{J}} \xi_j$.

$$\text{DFlow}: \qquad \min \sum_{a \in A} c(a) \cdot x_a \tag{10.1}$$

$$\sum_{i:(i,v)\in A} f_{iv}^{\chi} - \sum_{i:(v,i)\in A} f_{vi}^{\chi} = \begin{cases} -1, & \text{if } v = s \\ 1, & \text{if } v = t \\ 0, & \text{else} \end{cases} \qquad \forall \chi = (s,t) \in \mathcal{C}, \forall v \in V \tag{10.2}$$

$$\sum_{i:(i,w)\in A} \left(f_{iw}^{(v,r)} + f_{iw}^{(r,v)} \right) \leq 1 \qquad \forall v \in \mathcal{R}'_2, \forall w \in V \setminus \{r,v\} \tag{10.3}$$

$$x_{vw} + x_{wv} \leq 1 \qquad \forall \{v,w\} \in E \tag{10.4}$$

$$0 \leq f_a^{\chi} \leq x_a \qquad \forall a \in A, \forall \chi \in \mathcal{C} \tag{10.5}$$

$$x_a \in \{0,1\} \qquad \forall a \in A \tag{10.6}$$

The *flow-conservation constraints* (10.2) define the sent flow and ensure the flow balances, while the *coupling constraints* (10.3) ensure the node-disjointness of the pairs of forward and backward flow, by guaranteeing that at most one unit of flow per commodity pair is sent into any node. The inequalities (10.5) ensure the flow capacities and bind the flow variables f to the network-defining variables x. For the latter, (10.4) ensures that we have a unique orientation for the selected edges. Due to Theorem 10.4, the constraints (10.2), (10.5), (10.4), and (10.6) ensure that every $\mathcal{R}'_2$ customer belongs to the same edge-biconnected component as r. Theorem 10.9 and 10.5, and constraint (10.3) guarantee that this component is $(1,2)$-root-node-connected with respect to $\mathcal{R}'_2$.

According to Theorem 10.9, adding the following equation (10.7) leads to a multi-commodity flow ILP for the 2NCON problem. It ensures that we select only a single arc that is oriented towards the root r:

$$\sum_{(i,r)\in A} x_{ir} = 1. \tag{10.7}$$

Without (10.3) and (10.7), our DFlow formulation is equivalent to the 2ECON-DFlow.

10.4.2 DCut

Alternatively, we can formulate an ILP based on directed cuts—DCut—that also solves the 2ECON, 2NCON and 2RSN problems.

$$\text{DCut}: \qquad \min \sum_{a \in A} c(a) \cdot x_a \tag{10.8}$$

$$x_{vw} + x_{wv} \leq 1 \qquad \forall \{v, w\} \in E \tag{10.9}$$

$$x(\delta^-(S)) \geq 1 \qquad \forall S \subseteq V \setminus \{r\}, S \cap \mathcal{R}' \neq \emptyset \tag{10.10}$$

$$x(\delta^+(S)) \geq 1 \qquad \forall S \subseteq V \setminus \{r\}, S \cap \mathcal{R}'_2 \neq \emptyset \tag{10.11}$$

$$x(\delta^-_{\bar{G}_w}(S_1)) + x(\delta^+_{\bar{G}_w}(S_2)) \geq 1 \qquad \begin{cases} \forall w \in V \setminus \{r\}, \forall S_1, S_2 \subseteq V \setminus \{r, w\}, \\ S_1 \cap S_2 \cap \mathcal{R}'_2 \neq \emptyset \end{cases} \tag{10.12}$$

$$x_a \in \{0, 1\} \qquad \forall a \in A \tag{10.13}$$

As before, (10.9) guarantees the unique orientation of chosen edges. The constraints (10.10) and (10.11) are called *forward* and *backward dcut-constraints*, respectively. They ensure the existence of the required paths, and (10.12) assures the node-disjointness of these paths: After removing any node w, either the forward or the backward path still has to exist. Thus an optimal solution of the above ILP defines an optimal solution of the 2RSN problem.

Analogous to DFlow, extending DCut with (10.7) gives us a cut-based ILP on bidirected graphs for the 2NCON problem. Without (10.12) and (10.7), our DCut formulation is equivalent to 2ECON-DCut.

10.4.3 Extension to the Prize-Collecting Model

For prize-collecting variants of 2CON problems we cannot easily choose a root node in advance since we do not know any node that will definitively be selected. The situation does not arise for 2RSN, as even in the prize-collecting setting the prespecified root will always have to be contained in the solution network. We show how our DCut model can be extended to model the prize-collecting variant 2RPCSN. Observe that we can extend the DFlow model analogously. As introduced in Section 8, we are given a prize $p(v)$ for each node v. We introduce a second set of binary variables y_v for $v \in V$ that are 1, if v is in the solution network $\hat{N}$, and 0 otherwise. By definition, $y_r = 1$. The cut constraints require a non-zero cut only if a node in the considered cut set is selected:

$$\text{2R(PC)SN-DCut}: \qquad \min \sum_{a \in A} c(a) \cdot x_a - \sum_{v \in V} p(v) \cdot y_v \tag{10.14}$$

$$x_{uv} + x_{vu} \leq y_v \qquad \forall v \in V, \forall (v,u) \in A \tag{10.15}$$

$$x(\delta^-(S)) \geq y_v \qquad \forall S \subseteq V \setminus \{r\}, \forall v \in S \cap \mathcal{R} \tag{10.16}$$

$$x(\delta^+(S)) \geq y_v \qquad \forall S \subseteq V \setminus \{r\}, \forall v \in S \cap \mathcal{R}_2 \tag{10.17}$$

$$x(\delta^+_{G_w}(S_1)) + x(\delta^-_{G_w}(S_2)) \geq y_v \qquad \begin{cases} \forall w \in V \setminus \{r\}, \forall S_1, S_2 \subseteq V \setminus \{r, w\}, \\ \forall v \in S_1 \cap S_2 \cap \mathcal{R}'_2 \end{cases} \tag{10.18}$$

$$x_a, y_v \in \{0,1\} \qquad \forall a \in A, \forall v \in V \tag{10.19}$$

By adding the constraint (10.7) to this formulation, we obtain an ILP model of the 2PCNCON problem. Alternatively, by omitting (10.18) we obtain a 2PCECON ILP model.

10.5 Polyhedral Comparison

The polyhedral comparison provided in this section is done for 2NCON. The results related to 2R(PC)SN can be derived correspondingly, so we omit their proofs. We first compare the strength of the two concepts proposed above, DCut and DFlow. We then compare them with previously known ILP formulations and conclude this section with a hierarchy of the strength of LP relaxations for 2NCON.

10.5.1 Strength of DCut and DFlow

Let

$$\mathcal{P}_{DF} := \{(x,f) \in [0,1]^{|A|} \times [0,1]^{|A|\cdot|\mathcal{R}|} \mid (x,f) \text{ satisfies (10.2)–(10.4) and (10.7)}\}$$

and

$$\mathcal{P}_{DC} := \{x \in [0,1]^{|A|} \mid x \text{ satisfies (10.9)–(10.12) and (10.7)}\}$$

be the polytopes corresponding to LP relaxations of DFlow and DCut for 2NCON, respectively. To compare these polyhedra, we consider the projection of $\mathcal{P}_{DF}$ into the space of x variables, i.e.,

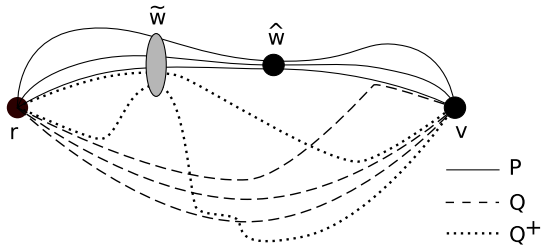
$$\text{proj}_x(\mathcal{P}_{DF}) := \{x \mid (x,f) \in \mathcal{P}_{DF}\}.$$

Theorem 10.12. *For 2NCON,* DFlow *and* DCut *are equivalent.*

Proof. Due to Observation 2.17 we have to show $\text{proj}_x(\mathcal{P}_{DF}) = \mathcal{P}_{DC}$.

$\underline{\text{proj}_x(\mathcal{P}_{DF}) \subseteq \mathcal{P}_{DC}}$: It is a classical and direct consequence of the max-flow min-cut theorem that if an assignment $(\bar{x}, \bar{f})$ for (x,f) is feasible for DFlow, then $\bar{x}$ is feasible for DCut, i.e., $\bar{x}$ satisfies the constraints (10.10)–(10.12). Assume there is a set $S \subseteq V \setminus \{r\}$ with $v \in S \cap \mathcal{R}'$ and $\bar{x}(\delta^-(S)) < 1$. Then the minimum r,v-cut—and therefore the maximum r,v-flow—is less than 1. This is a contradiction to the corresponding flow constraint (10.2) with commodity (r,v). Therefore, the constraints (10.10) and, analogously also (10.11), are satisfied.

Let $v \in \mathcal{R}'_2$. Since f satisfies DFlow, there is exactly one unit of commodity (r,v) going from r to v, and one unit of (v,r) going from v to r: the total amount



(a)

Figure 10.2: Sketch for the proof of Theorem 10.12, part of $\mathcal{P}_{DC} \subseteq \text{proj}_x(\mathcal{P}_{DF})$.

of flow between r and v is 2. The constraints (10.3) ensure that deleting any node $w \neq r$ in G can decrease this amount by at most one flow unit. Hence there is an (undirected) max-flow—and therefore a minimum undirected cut—of at least 1 between r and v in any G_w, which induces (10.12).

$\mathcal{P}_{DC} \subseteq \text{proj}_x(\mathcal{P}_{DF})$: We show that if an assignment $\bar{x}$ for x is feasible for DCut, then there exists a flow $\bar{f}$ such that $(\bar{x}, \bar{f})$ is feasible for DFlow. Clearly, all DFlow constraints dealing only with x-variables are satisfied as they are identical to the DCut formulation. It remains to show that we can fit flow into the network using the x-values as capacities. Note that the flows of different commodities are mostly independent of each other, as only the coupling constraints (10.3) define a dependency between the forward and the backward flow for each $v \in \mathcal{R}'_2$. It is clear that we can find a 1-flow from r to any $v \in \mathcal{R}'_1$, since (10.10) guarantees a minimum cut between r and v of at least 1. Analogously, because of (10.10) and (10.11), we can also find a forward and a backward flow $(\bar{f}^{(r,v)}, \bar{f}^{(v,r)})$ for each $v \in \mathcal{R}'_2$, and it remains to show that there always exists such a pair of flows that satisfies (10.3) for all nodes $w \in V \setminus \{r, v\}$.

Let us assume there exists no such pair of flows. Let $(\hat{f}^{(r,v)}, \hat{f}^{(v,r)})$ be the flows satisfying (10.2) and (10.5) such that the maximal violation of (10.3) is minimal. Let $\hat{w}$ be a node where such a maximal violation occurs. The paths used by the flow can then be divided into multiple paths that go through $\hat{w}$ and multiple paths which do not go through $\hat{w}$. We denote these path sets by P and Q, respectively. Let $\alpha > 1$ be the amount of flow over P, and we have $\beta = 2 - \alpha < 1$ as the flow over Q.

Due to (10.12) we know that there exists a set Q^+ of additional paths not going through $\hat{w}$ over which we can send $\beta^+ > 0$ amount of flow, such that $\beta + \beta^+ = 1$. Consider the flow pair $(\tilde{f}^{(r,v)}, \tilde{f}^{(v,r)})$, where the flow over $\hat{w}$ is only 1, and the second flow unit is sent over that paths of Q and Q^+.[1] Since the original flow was minimal in terms of constraint violation, this new pair of flows has a different node $\tilde{w}$ over which at least α flow units are sent, say $\gamma \geq \alpha$. Clearly, $\tilde{w}$ is part of Q^+.

[1] We can choose this new pair of flows such that (10.2) and (10.5) still holds, since, even if the newly routed flow over Q^+ sends the total amount β^+ in a single direction (say from r to v), we know that $\hat{f}$ satisfies (10.2) and therefore sends at least $\alpha - 1 = \beta^+$ units into each direction. In the modified flow we can hence remove enough directed flow per direction from P to allow valid flow using Q^+ instead.

Even if Q^+ contributes all of its flow units to γ, we have $\gamma = \gamma' + \beta^+$ and thus: $\alpha \leq \beta^+ + \gamma' = 1 - \beta + \gamma' = 1 - 2 + \alpha + \gamma' \quad \Longrightarrow \quad 1 \leq \gamma'$

The paths of Q cannot contribute to γ, since then we could modify the original flow $(\hat{f}^{(r,v)}, \hat{f}^{(v,r)})$ such that (10.3) is less violated for $\hat{w}$ (without introducing an additional violation of at least α). Thus γ' has contributions from paths of P. Since the new flow sends exactly 1 unit over P, all paths in P have to go through $\hat{w}$ and $\tilde{w}$: otherwise we could choose a path going through $\hat{w}$ and not through $\tilde{w}$ and further reduce the flow through the latter. Hence, in $(\hat{f}^{(r,v)}, \hat{f}^{(v,r)})$ both $\hat{w}$ and $\tilde{w}$ have a through-flow of α, and the paths in P can be subdivided into subpaths between r and $\tilde{w}$, $\tilde{w}$ and $\hat{w}$, and $\hat{w}$ and v, assuming w.l.o.g. that $\tilde{w}$ is closer to r than $\hat{w}$. But then we can iterate the above argument, send only 1 unit of flow through $\hat{w}$ and $\tilde{w}$, and find an additional node $\check{w}$ with too much through-flow. This argument can be iterated ad infinitum, requiring an infinitely large graph. □

By analogous proofs we obtain:

Corollary 10.13. DFlow *and* DCut *are equivalent for 2RSN and 2RPCSN.*

10.5.2 Comparison to the Undirected Cut Formulation

We compare our 2NCON-DCut formulation with the currently best known and widely used 2NCON-UCut formulation. For better readability we restate it here:

$$\text{2NCON-UCut}: \qquad \min \sum_{e \in E} c(e) \cdot z_e \tag{10.20}$$

$$z(\delta(S)) \geq 1 \qquad \forall S \subset V, \emptyset \neq S \cap \mathcal{R} \neq \mathcal{R} \tag{10.21}$$

$$z(\delta(S)) \geq 2 \qquad \forall S \subset V, \emptyset \neq S \cap \mathcal{R}_2 \neq \mathcal{R}_2 \tag{10.22}$$

$$z(\delta_{G_w}(S)) \geq 1 \qquad \forall w \in V, \forall S \subset V, \emptyset \neq S \cap (\mathcal{R}_2 \setminus \{w\}) \neq \mathcal{R}_2 \setminus \{w\} \tag{10.23}$$

$$z_e \in \{0,1\} \qquad \forall e \in E \tag{10.24}$$

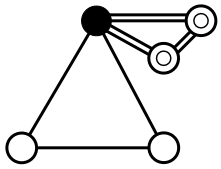
For 2ECON it is known (cf. [Cho92, Rag95]) that directed formulations are strictly stronger than undirected ones, as long as $\mathcal{R}_1 \neq \emptyset$. For 2NCON, we can also show that our (rooted, directed) 2NCON-DCut formulation is strictly stronger than the (unrooted, undirected) 2NCON-UCut formulation. Furthermore, this even holds if $\mathcal{R}_1 = \emptyset$.

Let $\mathcal{P}_{UC}$ be the polyhedron corresponding to the LP relaxation of 2NCON-UCut. For $\mathcal{P}_{DC}$, we can use the natural projection

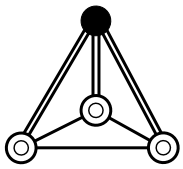
$$\text{proj}_z(\mathcal{P}_{DC}) := \{z \in [0,1]^{|E|} \mid x \in \mathcal{P}_{DC}, z_{ij} = x_{ij} + x_{ji} \; \forall e = \{i,j\} \in E\}.$$

Theorem 10.14. 2NCON-DCut *formulation is strictly stronger than* 2NCON-UCut. *This also holds if* $\mathcal{R}_1 = \emptyset$.

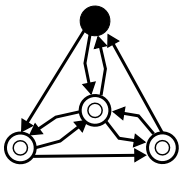
Proof. We first show that $\text{proj}_z(\mathcal{P}_{DC}) \subseteq \mathcal{P}_{UC}$, i.e., that any $\bar{z} \in \text{proj}_z(\mathcal{P}_{DC})$ satisfies constraints (10.21)–(10.23). Let $\bar{x}$ be the vector from which we projected $\bar{z}$. For any set $S \subset V$ we have $\bar{z}(\delta(S)) = \bar{z}(\delta(V \setminus S)) = \bar{x}(\delta^-(V \setminus S)) + \bar{x}(\delta^-(V))$. Consider any constraint (10.21) with its corresponding set S. If $r \in S$, we can consider (10.10) and



(a) $\mathcal{R}_1 \neq \emptyset$: 2NCON-DCut is strictly stronger than 2NCON-UCut.



(b) $\mathcal{R}_1 = \emptyset$: 2NCON-DCut is strictly stronger than 2NCON-UCut.



(c) The node-partition inequalities are not satisfied.

Figure 10.3: In the above figures, the root node is denoted by the black circle. A empty circle denotes a $\mathcal{R}_1$ node, a double circle denotes a $\mathcal{R}_2$ node. Single edges correspond to a fractional solution of 0.5, double edges correspond to a fractional solution of 1.

have $\bar{x}(\delta^-(V \setminus S)) \geq 1$. Analogously, if $r \notin S$, we have $\bar{x}(\delta^-(S)) = \bar{x}(\delta^+(V \setminus S)) \geq 1$. Therefore, in both cases we have $\bar{z}(\delta(S)) \geq 1$.

Consider any constraint (10.22) with its corresponding set S; we show: $\bar{z}(\delta(S)) = \bar{x}(\delta^-(S)) + \bar{x}(\delta^+(S)) \geq 2$. If $r \in V \setminus S$, the inequalities (10.10) and (10.11) directly give the above formula. If $r \in S$, we can consider the cut set $V \setminus S$ instead of S, as $\bar{z}(\delta(S)) = \bar{z}(\delta(V \setminus S))$. Using the same argument for the graph G_w we use the inequalities (10.23) to show that $\bar{z}$ satisfies (10.12) with $S_1 = S_2$.

It remains to show that there exists a 2NCON instance where $\mathcal{P}_{DC}$ leads to strictly stronger bounds, i.e.,

$$\min\{c^{\mathrm{T}} z \mid z \in \mathcal{P}_{UC}\} \lneq \min\{c^{\mathrm{T}} z \mid z \in \mathrm{proj}_z(\mathcal{P}_{DC})\}.$$

We therefore use the graph $G = (\{a, b, r, c, d\}, E_1 \cup E_2)$ with the root node r, depicted in Figure 10.3(a). Thereby, we have $a, b \in \mathcal{R}_1$, $r, c, d \in \mathcal{R}_2$ and the edges of E_1 and E_2 of uniform costs α forming triangle graphs on the node sets $\{a, b, r\}$ and $\{r, c, d\}$, respectively.

Setting $\bar{z}_e = 0.5$ for each $e \in E_1$ and $\bar{z}_e = 1$ for each $e \in E_2$ gives us a feasible solution to 2NCON-UCut of cost $1.5\alpha + 3\alpha$, cf. Figure 10.3(a). However, every feasible solution of 2NCON-DCut for the above instance will cost at least $2\alpha + 3\alpha$: Consider the bidirection $\bar{G} = (\{a, b, r, c, d\}, A_1 \cup A_2)$ of G: As the forward dcut-constraints have to be satisfied for the $\mathcal{R}_1$ nodes, $x(A_1)$ has to be at least 2. Analogously, satisfying forward and backward dcut-constraints for the nodes c and d, we obtain $x(A_2) \geq 3$.

Figure 10.3(b) shows that $\mathrm{proj}_z(\mathcal{P}_{DC}) \neq \mathcal{P}_{UC}$ even if $\mathcal{R}_1 = \emptyset$. The drawing represents a feasible fractional solution $\bar{z} \in \mathcal{P}_{UC}$, whereby all nodes are $\mathcal{R}_2$ customers. The edges incident to the root node have the costs α' and other edges are of cost β', with $0 \leq \alpha' \ll \beta'$. The value of the above solution is $3\alpha' + 1.5\beta'$. However, for the same problem instance, the optimal value of the LP relaxation of 2NCON-DCut is $2\alpha' + 2\beta'$, as it can be easily verified by solving the LP relaxation. □

We can easily give an undirected cut formulation 2R(PC)SN-UCut that is very similar to 2NCON-UCut. The main difference is that the cuts are defined with respect to the given root:

$$\text{2RPCSN-UCUT}: \qquad \min \sum_{e \in E} c(e) \cdot z_e - \sum_{v \in V} p(v) \cdot y_v \tag{10.25}$$

$$z(\delta(S)) \geq y_v \qquad \forall S \subseteq V \setminus \{r\}, \forall v \in S \cap \mathcal{R}_1 \tag{10.26}$$
$$z(\delta(S)) \geq 2y_v \qquad \forall S \subseteq V \setminus \{r\}, \forall v \in S \cap \mathcal{R}_2 \tag{10.27}$$
$$z(\delta_{G_w}(S)) \geq y_v \qquad \forall S \subseteq V \setminus \{r\}, \forall v \in S \cap \mathcal{R}_2, \forall w \in V \setminus \{r, v\} \tag{10.28}$$
$$z_e, y_v \in \{0, 1\} \qquad \forall e \in E, v \in V \tag{10.29}$$

Theorem 10.15. 2R(PC)SN-DCUT *is strictly stronger than* 2R(PC)SN-UCUT*, in general. If* $\mathcal{R}_1 = \emptyset$*, they are equivalent.*

Proof. For $\mathcal{R}_1 \neq \emptyset$, the proof is analogous to the proof of Theorem 10.14.

Consider now the case $\mathcal{R}_1 = \emptyset$. The idea of the following proof was first used by Raghavan [Rag95] in the context of 2ECON. We show that if $(\bar{z}, \bar{y})$ is feasible for 2R(PC)SN-UCUT, there exists a solution $(\bar{x}, \bar{y})$ with $\bar{z}_{ij} = \bar{x}_{ij} + \bar{x}_{ji}$ for all $\{i, j\} \in E$ feasible to 2R(PC)SN-DCUT. In fact, a solution $(\bar{x}, \bar{y})$ obtained by setting $\bar{x}_{ij} = \bar{x}_{ji} := \bar{z}_{ij}/2$ for all $\{i, j\} \in E$ satisfies the inequalities (10.16) (and analogously (10.17)): For a set $S \in V \setminus \{r\}$ and $v \in S \cap \mathcal{R}_2$, we have

$$\bar{x}(\delta^-(S)) = \bar{z}(\delta(S))/2 \overset{(10.27)}{\geq} 2y_v/2 = y_v.$$

Furthermore, for any $v \neq w \in V$, $S_1, S_2 \subseteq V \setminus \{r, w\}$ we have

$$\bar{x}(\delta^+_{G_w}(S_1)) + \bar{x}(\delta^-_{G_w}(S_2)) = \bar{z}(\delta(S_1))/2 + \bar{z}(\delta(S_2))/2 \overset{(10.16),(10.17)}{\geq} y_v.$$

□

Summarizing the results of Theorems 9.10, 9.11, 10.14, 10.15 we obtain:

Corollary 10.16. *For our considered problem classes, the relationship between* DCUT *and* UCUT *is established as below.* $\surd$ *means that the* DCUT *formulation is strictly stronger,* $\Leftrightarrow$ *means that both are equivalent. The relationships hold true independent on whether* $\mathcal{R}_0 = \emptyset$ *or not; clearly,* $|\mathcal{R}_2| \geq 2$*.*

	2ECON	2NCON	2RSN, 2RPCSN
$\mathcal{R}_1 = \emptyset$	$\Leftrightarrow$	$\surd$	$\Leftrightarrow$
$\mathcal{R}_1 \neq \emptyset$	$\surd$	$\surd$	$\surd$

10.5.3 Comparison to the Mixed Flow Formulation

Till now, the strongest ILP for 2NCON was the MFLOW-ILP (cf. Section 9.4). We show that our 2NCON-DCUT is beneficial. Recall that MFLOW is stated as follows:

$$\text{MFLOW}: \qquad \min \sum_{e \in E} c(e) z_e \tag{10.30}$$

$$\sum_{(i,v)\in A} g^{\chi}_{iv} - \sum_{(v,i)\in A} g^{\chi}_{vi} = \begin{cases} -1, & \text{if } v = s \\ 1, & \text{if } v = t \\ 0, & \text{else} \end{cases} \quad \forall \chi = (s,t) \in \mathcal{C}, \forall v \in V \tag{10.31}$$

$$g^{\chi}_{vw} + g^{\chi'}_{wv} \leq z_{vw} \qquad \forall \{v,w\} \in E, \forall \chi, \chi' \in \mathcal{C} \tag{10.32}$$

$$g^{\chi}_{vw} \geq 0 \qquad \forall (v,w) \in A, \forall \chi \in \mathcal{C} \tag{10.33}$$

$$\sum_{(i,v)\in A} h^{\chi}_{iv} - \sum_{(v,i)\in A} h^{\chi}_{vi} = \begin{cases} -2, & \text{if } v = s \\ 2, & \text{if } v = t \\ 0, & \text{else} \end{cases} \quad \forall \chi = (s,t) \in \mathcal{C}_2, \forall v \in V \tag{10.34}$$

$$0 \leq h^{\chi}_{vw} \leq z_{vw} \qquad \forall (v,w) \in A, \forall \chi \in \mathcal{C}_2 \tag{10.35}$$

$$\sum_{(v,i)\in A} h^{\chi}_{vi} \leq 1 \qquad \forall \chi = (s,t) \in \mathcal{C}_2, v \in V \setminus \{s,t\} \tag{10.36}$$

$$z_e \in \{0,1\} \qquad \forall e \in E \tag{10.37}$$

Let $\mathcal{P}_{MF}$ be the polyhedron of the feasible solutions of the LP relaxation of MFlow and

$$\text{proj}_z(\mathcal{P}_{MF}) := \{z \in [0,1]^{|E|} \mid (z,g,h) \in \mathcal{P}_{MF}\}$$

its projection into the variable space of z. We also consider extended projections including the flow variables $f \in [0,1]^{|A|\cdot|\mathcal{C}|}$ eventhough they are not in the objective function. Let

$$\text{proj}_{z,f}(\mathcal{P}_{DF}) := \{(z,f) \mid (x,f) \in \mathcal{P}_{DF}, z_e = x_{ij} + x_{ji} \ \forall e = \{i,j\} \in E\}$$

be such a projection of $\mathcal{P}_{DF}$ (cf. page 113) and

$$\text{proj}_{z,f}(\mathcal{P}_{MF}) := \{(z,f) \mid (z,g,h) \in \mathcal{P}_{MF}, f = g\}$$

the projection of $\mathcal{P}_{MF}$ ignoring the h flow. In other words, we identify the flows f and g.

We show that the lower bounds obtained by the LP relaxations of our 2NCON-DFlow formulation are at least as tight as those of the mixed flow formulation. Therefore note that the flow f is a kind of natural fusion of the flow g and the node-disjointness properties of h.

Theorem 10.17. 2NCON-DFlow *is weakly stronger than* MFlow. *Furthermore, there exist instances such that* $\text{proj}_{z,f}(\mathcal{P}_{DF}) \subset \text{proj}_{z,f}(\mathcal{P}_{MF})$.

Proof. We show that for any feasible solution $(\bar{x}, \bar{f})$ of 2NCON-DFlow we can obtain an equivalent feasible solution $(\bar{z}, \bar{g}, \bar{h})$ of MFlow, using $\text{proj}_{z,f}$ as described above. Based on these projections, it is easy to see that (10.31), (10.32), and (10.33) are satisfied. It remains to show that we can always find a feasible flow solution h within the network G with the projected edge capacities z. If $\chi = (s,t) \in \mathcal{C}$, we can choose $\bar{h}^{\chi}_e := \bar{f}^{\chi}_e + \bar{f}^{(t,s)}_e$, which satisfies (10.34), (10.35), (10.36) due to (10.5) and proj_z. For $\chi \in \mathcal{C}_2$, we look at the maximum s,t-flow in G with capacities z and consider any corresponding minimum s,t-cut; let S and $V \setminus S$ be the cut sets containing s and t, respectively. W.l.o.g. assume that $r \in S$. Since (10.2)

is satisfied for the commodities (r,t) and (t,r), the maximum undirected r,t-flow, and therefore also the maximum undirected s,t-flow h^χ, is at least 2. Assume we cannot send h^χ without violating (10.36). Then there exists a single node w such that the total capacity κ of the cut edges which do not send their flow through w is less than one. If $w \neq r$, (10.3) guarantees that we can send two (undirected) flow units between r and t whereby at most one unit is sent through w. This is a contradiction to κ. If $w = r$, (10.7) guarantees an in-flow into r of exactly 1 for both the s,r- and the t,r-flow. Hence, using the two 1-flows $\bar{f}^{(s,r)}$ and $\bar{f}^{(t,r)}$ we can send an undirected flow of at least 1 between s and t without using r, which is again a contradiction to κ.

To establish the second claim it is enough to construct a feasible flow g in MFlow which sends more than one unit into the root r. This is infeasible for 2NCON-DFlow as (10.7) is violated. □

2NCON-DFlow requires less variables and constraints than MFlow, hence:

Observation 10.18. 2NCON-DFlow *is more compact than* MFlow.

Our formulation answers the question by Raghavan [Rag95, p. 183] whether his flow variables g,h can be bounded together more tightly. Note that Theorem 10.9 is crucial for the validity of our approach, which explains why this compact formulation could not be used legitimately before.

10.5.4 Additional Cut-Constraints

Recall that 2NCON-UCut without (10.23) is the traditional 2ECON-UCut. It has been shown in [Sto92] that the latter formulation can be strengthened by adding certain classes of valid inequalities that are NP-hard to separate. Chopra [Cho92] showed that his 2ECON-DCut formulation inherently includes one of these classes, namely the *partition inequalities*: Let $\{S_1,\dots,S_p\}$ be a proper partition of V into p non-empty sets such that $S_i \cap \mathcal{R} \neq \emptyset$ for each $1 \leq i \leq p$. We denote by $\delta(S_1,\dots,S_p)$ the set of edges connecting different partition sets. We can require

$$z(\delta(S_1,\dots,S_p)) \geq \begin{cases} p & \text{if at least two } S_i \text{ contain } \mathcal{R}_2 \text{ nodes,} \\ p-1 & \text{otherwise.} \end{cases} \tag{10.38}$$

Stoer [Sto92, pp. 130–134] showed that the latter formulation also includes the more general class of the (polynomially separable) *Prodon inequalities*: Let b be any positive vector indexed over all subsets S for which the inequalities (9.14) and/or (9.15) are defined. We denote by $B(b) := \sum_S b_S$ the sum over all its entries and let $B_i^j(b) := \sum_{S:i\in S, j\notin S} b_S$. We can require

$$\sum_{\{i,j\}\in E} \max\{B_i^j(b), B_j^i(b)\} \cdot z_{ij} \geq B(b).$$

Moreover, Raghavan showed that his improved undirected multi-commodity flow formulation for kECON, which for $k = 2$ is equivalent to both 2ECON-DFlow and 2ECON-DCut, also includes the *odd-hole inequalities* and the more general

combinatorial-design inequalities [Rag95, pp. 165–180]. These inequalities require certain minimum edge selections based on special subgraphs in G.

In fact, the validity of the last three inequality classes was shown by projecting them from the directed cut based formulations. By dropping the constraints (10.12) and (10.7) from 2NCON-DCut, we obtain 2ECON-DCut. Hence we can conclude:

Proposition 10.19. 2NCON-DCut *and* 2NCON-DFlow *inherently ensure the validity of the partition, Prodon, odd-hole, and combinatorial-design inequalities.*

For a node $v \in V$ let $\{S_1, \ldots, S_p\}$ be a proper partition of $V \setminus \{v\}$ into p non-empty sets such that $S_i \cap \mathcal{R}_2 \neq \emptyset$ for each $1 \leq i \leq p$. Let $\delta_{G_v}(S_1, \ldots, S_p)$ be the set of edges connecting different partition sets in G_v. The *node-partition inequalities*—i.e., undirected partition inequalities where one node is removed from the graph—strengthen the 2NCON-UCut formulation and can be written as

$$z(\delta_{G_v}(S_1, \ldots, S_p)) \geq p - 1. \tag{10.39}$$

These inequalities were first proposed in [GM90] and then generalized in [Sto92, pp. 91–94]. It was an open question [Rag95, p. 183] whether MFlow would induce this constraint class. Our following result constitutes a negative answer for this question:

Proposition 10.20. *In general, none of the above formulations induces the node-partition inequalities* (10.39)*.*

Proof. See Figure 10.3(c) for an example, which denotes the x variables of the arcs. Once more, we use the natural projection $x_{vw} + x_{wv} = z_{vw}$ for all $\{v, w\} \in E$ to obtain an undirected network: When removing the central node, the partition inequality with three partition sets is violated. Yet the solution is feasible for the LP relaxation of 2NCON-DCut. □

Some of the undirected node-partition inequalities are, however, inherently included in 2NCON-DCut:

Proposition 10.21. *Let $\bar{x} \in \mathcal{P}_{DC}$. Then $\bar{x}$ satisfies the node-partition inequalities for all valid partitions of $V \setminus \{r\}$.*

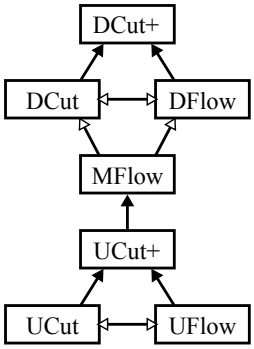
Proof. Consider any partition $S_1, \ldots, S_p$, $p \geq 2$, of the node set $V \setminus \{r\}$ such that $S_i \cap \mathcal{R}_2 \neq \emptyset$ for all $1 \leq i \leq p$. From (10.10) und (10.7) it follows:

$$z(\delta_{G_r}(S_1, \ldots, S_p)) = \sum_{1 \leq i \leq p} x(\delta^-(S_i)) - x(\delta^+(r)) \geq p - 1. \qquad \square$$

Observation 10.22. *For nodes $v \in \mathcal{R}_2$ with* $\deg(v) = 2$ *in G, the node-partition inequalities are always induced. This holds even for* 2NCON-UCut*. Hence, such nodes in general do not represent good choices as the root nodes in* 2NCON-DCut*, as we gain more by choosing another $\mathcal{R}_2$ node with larger degree.*

We obtain the following hierarchy of formulations for 2NCON:

Corollary 10.23. *The following hierarchical scheme summarizes the relationships between the LP relaxations of the ILP models considered throughout this chapter for 2NCON.*



A filled arrow specifies that the target formulation is strictly stronger than the source formulation. An empty arrow specifies that the target formulation is weakly stronger as the source formulation. Thereby, UCut+ *denotes* UCUT *with partition inequalities, and* DCut+ *denotes* DCUT *with node partition inequalities.*

Chapter 11

Branch-and-Cut

From the point of formulation strength, using DFlow instead of DCut might seem like a reasonable choice in general, as both the number of variables and constraints are polynomially bounded whereas DCut has an exponential number of constraints. However, this drawback of DCut can turn out to be beneficial, as the actual computation of an optimal solution will in general not require all of these constraints. The required constraints in DCut can be easily separated using simple polynomial maximum flow algorithms, see below. Hence, we can obtain the optimal fractional solution of the LP relaxation at the root of the branch-and-bound tree in polynomial time, based on the theorem regarding the equivalence of optimization and separation; cf. Section 2.3.4 and, e.g., [Wol98]. Furthermore, the number of variables in DCut is independent of $|\mathcal{R}|$ as it only requires variables x_{ij} for all $(i,j) \in A$. This is beneficial especially for large $|\mathcal{R}|$.

Based on the DCut approach, we develop a branch-and-cut code, where we can use the same branch-and-cut strategy for both problems, 2R(PC)SN and 2NCON.

11.1 Initialization

We start with the constraints (10.9) and the subset of the constraints (10.10), (10.11) for $|S| = 1$. Even for 2NCON and 2RSN, we use an extended DCut formulation, including binary variables y_v for $v \in \mathcal{R}_0$. Although they do not strengthen the DCut model, the following *flow-preservation* constraints significantly speed up the computation time:

$$\sum_{k:(k,i)\in A} x_{ki} \geq y_i \quad \text{and} \quad \sum_{k:(i,k)\in A} x_{ik} \geq y_i, \quad \forall i \in \mathcal{R}_0. \tag{11.1}$$

These inequalities specify that no node from $\mathcal{R}_0$ will ever have only incoming or only outgoing arcs. They are especially useful for instances with few customers and comparably long paths connecting them in the optimal solution.

11.2 Separation

The forward dcut-constraints (10.10) can be separated in polynomial time via the traditional separation scheme based on maximum flow computations: after obtain-

ing some LP relaxation for our partial ILP, we compute the maximum flow from r to each $v \in \mathcal{R}$ in $\bar{G}$ using the arc values of the current solution as arc capacities. If the resulting value is less than 1 (or y_v, in case of 2RPCSN), we use front, back, and nested cuts (cf. Section 6.3) to extract one or more of the induced minimum r, v-cuts, and add the corresponding constraints to our (I)LP model. The backward dcut-constraints (10.11) can be separated analogously.

If there are no violated constraints of type (10.10) nor (10.11), we solve the separation problem for the constraints of type (10.12) in a similar way: for each node $v \in \mathcal{R}_2$ and for each node $w \in V$, $w \neq v$ we compute maximum flows in $\bar{G}_w$ from r to v and from v to r. If the sum of these flows is less than 1 (or y_v, for 2RPCSN), we add the corresponding inequalities.

Furthermore, to speed up the separation of node-disjointness constraints, we use the following idea: consider an integer solution where the constraints (10.10) and (10.11) are satisfied, i.e., we have directed paths $(r \to v)$ and $(v \to r)$ for any $v \in \mathcal{R}_2$. If we assume that these paths have a common node w, then there are often at least two incoming and two outgoing edges at w. Therefore, when separating constraints (10.12), we first try nodes $w \in V$ with $\bar{x}(\delta^-(w)) > 1$ and $\bar{x}(\delta^+(w)) > 1$ in our fractional solution. Finally, it turns out to be beneficial to select the cuts containing the smallest number of arcs among all violated cuts of type (10.10), (10.11) or (10.12). In fact, this simple property is crucial for solving large graphs efficiently in practice.

11.3 Upper Bounds

In the following, we present a general heuristic for our {0,1,2}-SND problems. We then show how to use a fractional solution of the LP relaxation to obtain an LP-based heuristic. This heuristic can then be used in our branch-and-cut framework in order to obtain upper bounds for the optimal solution.

11.3.1 Initial Heuristic

Let $(G, \mathcal{R} = \mathcal{R}_1 \dot{\cup} \mathcal{R}_2, r, c)$ be an instance of a {0,1,2}-SND problem with node-connectivity, i.e., 2(PC)NCON or 2(PC)RSN. The first step of our heuristic is to compute a subgraph G that is feasible for the given problem. The second step is to remove redundant edges, i.e., edges whose removal does not violate the given connectivity requirements and descreases the overall solution cost.

Construct a Feasible Solution. First we compute a Steiner tree T in G connecting the set $\mathcal{R}$, using the traditional Steiner tree heuristic of Mehlhorn [Meh88]. In order to assure the required 2-node-connectivity of $\mathcal{R}_2$ nodes, we iteratively extend T: Let $G' = (V' \subseteq V, E' \subseteq V)$ with $T \subseteq G'$ be our current solution and $B \subseteq V'$ the set of nodes in G' that are already biconnected with r. Initially, we have $G' = T$, $B = \{r\}$ and set $c(e) := 0$ for all $e \in E'$. Consider a node $v \in \mathcal{R}_2 \setminus B$ that is currently not biconnected correctly. Let P_v be the unique $[r \to v]$-path in T, b be the node in $P_v \cap B$ closest to v, and $P_v(b)$ the subpath of P_v from v to b. We then extend G' by ensuring two node-disjoint $[r \to v]$-paths as follows:

(a) (b)

Figure 11.1: Double circles represent $\mathcal{R}_2$ nodes, white circles correspond to $\mathcal{R}_1$, and small circles to $\mathcal{R}_0$. The black node represents the root r. Bold edges belong to a feasible Steiner tree. (a) If we delete all bold edges, there is no feasible path between the $\mathcal{R}_2$ customer and r. (b) The nodes b and d are $\mathcal{R}_2$-leaves: if we biconnect b with r, e.g. via the edge $\{r, b\}$, the node a is also biconnected.

1. We temporarily remove all inner nodes of $P_v(b)$ from G. If $b \neq r$, b is also temporarily deleted from G.

2. If we are dealing with the 2NCON problem: if $B \neq \{r\}$ and $b = r$, we also temporarily remove r from G.

3. We use Dijkstra's algorithm to find a shortest path $P(v, B)$ between v and the component B in G.

4. We add $P(v, B)$ to G', set the corresponding edge-weights to zero, and update B.

Note that step 2 of the above procedure constitutes the main and only difference between a heuristic for the 2(PC)NCON problem and the one for the 2R(PC)SN problem. It ensures that the current 2NCON solution always contains only one 2-node-connected component.

By the above operations, all nodes of P_v also become 2-connected. Hence, we have to perform these steps only for $\mathcal{R}_2$-*leaves*, i.e., $\mathcal{R}_2$ nodes that do not have any further $\mathcal{R}_2$ customers in their subtrees of T, see Figure 11.1(b). In order to save computation time, this set can be efficiently updated in step 4. The quality of the obtained solution may strongly depend on the order in which the $\mathcal{R}_2$-leaves are processed in our algorithm. In order to achieve better solutions, this set can be sorted according to the number of $\mathcal{R}_2$ nodes or/and the number of edges on the path $P(v, B)$.

Observe that even if the given problem instance is feasible, it may happen that our algorithm does not find any feasible solution. The removal of some nodes in step 1 may disconnect the current solution subgraph such that no shortest path can be found in step 3, cf. Figure 11.1(a) for an example. However, our experiments show that for the tested instances this happens quite rarely, allowing us to successfully use this heuristic within our branch-and-cut framework.

Shrinking. In general, G' can be further improved by local optimizations. We know that G' consists of one or more non-trivial 2-connected components, which

have only the root node in common. All other components of the graph form trees that are attached to some 2-connected component.

For every 2-connected component B of G' we compute its *core graph* $\tilde{B}$, where every line with inner nodes $v \in V \setminus (\mathcal{R}_2 \cup r)$ is replaced by a single edge. Let $P_{\tilde{e}}$ denote the path in G' corresponding to an edge $\tilde{e}$ of $\tilde{B}$.

In the next step, we look for the redundant edges in $\tilde{B}$, i.e., edges that can be removed without destroying its biconnectivity. We thereby consider its edges in decreasing order of their corresponding paths' costs.

2NCON and 2RSN. Consider the non-prize-collecting problem variants. An edge $\tilde{e}$ can be removed from $\tilde{B}$ if all connectivity requirements are still satisfied for $\tilde{B} - e$. If $P_{\tilde{e}}$ does not contain any inner $\mathcal{R}_1$ node, we remove it completely from G'. Otherwise, as these $\mathcal{R}_1$ nodes have to be in part of the solution, we remove only a subpath of maximum cost where all inner nodes are from $\mathcal{R}_0$. Clearly, such a subpath can be easily computed in a linear time.

Prize-collecting problem variants. When not all customer nodes have to be included into the final solution, we can apply the following local improvement steps:

1. As described above, the solution subgraph may contain trees that are attached to some 2-connected component. For each such tree, using the attachment node as its root, we can solve the prize-collecting tree problem (PCST) in order to decide which $\mathcal{R}_1$ are profitable enough to be contained in our solution. As described in [Wol98], the rooted PCST problem can be solved in linear time, when applied to trees. For the next step, we temporarily remove these trees from G and consider the attachment nodes to be $\mathcal{R}_1$ customers with the corresponding prizes of their subtrees added to their original prize.

2. Let $\tilde{B}$ be defined as above, $\tilde{e}$ a removable edge in $\tilde{B}$ (in the non-prize-collecting setting), and $P_{\tilde{e}}$ the corresponding path in $\tilde{B}$. If $P_{\tilde{e}}$ contains no $\mathcal{R}_1$ node, we can easily delete $P_{\tilde{e}}$ also in the prize-collecting setting. Otherwise we have to decide whether it is worth to include such nodes into the solution and if this is the case, which edges of $P_{\tilde{e}}$ are superfluous for the 1-connectedness of these. More formally, let $P_{\tilde{e}} = [v_1 \to v_n]$ be a line in $\tilde{B}$ with inner nodes $v_2, \ldots, v_{n-1}$, some of which are $\mathcal{R}_1$ nodes. We have to determine the nodes v_a and v_b, $1 \leq a < b \leq n$, such that removing $[v_a \to v_b]$ maximizes the overall profit of connecting the remaining (not necessarily all) $\mathcal{R}_1$ nodes. Algorithm 1 gives a procedure to determine this subpath in $O(|P_{\tilde{e}}|)$ time: In the first sweep, we compute the profit of the (forward) path $[v_1 \to v_i]$ for each node $v_i \in \mathcal{R}_1$ of $P_{\tilde{e}}$. In the second sweep, we not only compute the profit of the (backward) path $[v_n \to v_i]$ for each $v_i \in \mathcal{R}_1$, but also determine a j, $i < j \leq n$, such that the sum of the profits of $[v_0 \to v_i]$ and $[v_j \to v_n]$ is maximized. By keeping the most profitable pair i, j we identify optimal v_a and v_b.

Complexity. We can construct the initial Steiner tree T in $\mathcal{O}(|E| + |V| \log |V|)$ time [Meh88]. We then have $\mathcal{O}(|\mathcal{R}_2|)$ many path addition steps. We can easily find and sort all $\mathcal{R}_2$ leaves in $\mathcal{O}(|V|)$ time using, e.g., bucket sort. For each such step,

Algorithm 1 Optimal pruning of a line contaning $\mathcal{R}_1$ nodes for the prize-collecting problem variants.

```
1: fsum := 0                                          ▷ running sum of profit for [v_1 → v_i]
2: fcost[i] := 0∀1 ≤ i ≤ n                            ▷ store profit for [v_1 → v_i]
3: for all i = 2, ..., n − 1 do
4:     fsum := fsum − c({v_{i−1}, v_i})
5:     if v_i ∈ R_1 then
6:         fsum := fsum + p(v_i)
7:         fcost[i] := fsum
8:     end if
9: end for
10: bsum := 0                                         ▷ running sum of profit for [v_n → v_i]
11: bcost := 0                                        ▷ current profit for [v_n → v_j] over all j > i
12: overall := 0                                      ▷ current (and final) overall profit
13: v_a := v_1, v_b := v_n, v_j := v_n
14: for all i = n − 1, ..., 1 do
15:     bsum := bsum − c({v_i, v_{i+1}})
16:     if v_i ∉ R_0 then
17:         if fcost[i] + bcost > overall then
18:             overall := fcost[i] + bcost
19:             v_a := v_i
20:             v_b := v_j
21:         end if
22:         bsum := bsum + p(v_i)
23:         if bsum > bcost then
24:             bcost := bsum
25:             v_j := v_i
26:         end if
27:     end if
28: end for
```

Dijstkra's $\mathcal{O}(|E| + |V| \log |V|)$ algorithm dominates the other substeps (temporary removing of tree edges, updating of B and list of $\mathcal{R}_2$ leaves) that are linear in $|V|$. Hence, constructing an initial solution requires $\mathcal{O}(|\mathcal{R}_2|(|E| + |V| \log |V|))$ time.

Observe that decomposing G' in its biconnected components requires $\mathcal{O}(|E|)$ time and that the biconnected components and attached trees are pairwise edge-disjoint. Optimally shrinking an attached tree requires only linear time in its number of edges. Hence, optimizing all these trees takes $\mathcal{O}(|E|)$ time. For any biconnected component B with q edges, we can compute its core graph $\tilde{B}$ in $\mathcal{O}(q)$ time and then sort its edges in $\mathcal{O}(q \log q)$ time. For each edge $\tilde{e}$ of $\tilde{B}$ we test whether its removal leaves a biconnected graph ($\mathcal{O}(q)$) and optimally reduce $P_{\tilde{e}}$ ($\mathcal{O}(|P_{\tilde{e}}|)$). Since all these paths are disjoint, pruning any biconnected component takes $\mathcal{O}(q^2)$ time. Hence, the overall worst case for the shrinking step arises when there is a biconnected component with $\mathcal{O}(|E|)$ edges.

The running time of our heuristic is hence dominated by the shrinking procedure

and overall takes $\mathcal{O}(|E|^2)$ time.

11.3.2 LP-based Heuristic

During the branch-and-cut computation we are given a sequence of fractional solutions $(\bar{x}, \bar{y})$ (or only $\bar{x}$) of LP relaxations that are defined on the bidirected graph $\bar{G} = (V, A)$. We can use these values as hints in order to find better upper bounds.

We obtain our LP-based heuristic by applying our initial heuristic to an instance (G, C, r, w) with C and w defined as follows: for the 2RSN and 2NCON problems we set $C := \mathcal{R}'$. In the case we are dealing with their prize-collecting variants, we use the $\bar{y}$-values of the current LP relaxation in order to determine the set C as it was done by Ljubic [Lju04]. In our context, we tested the following strategies:

- A node v is included in C with probability $\bar{y}_v$.
- A node v is included in C if $\bar{y} \geq 0.5$.

For each edge $\{i, j\} \in E$ we used the following strategies to define the edge cost function w:

W1 $w(\{i, j\}) = 1 - \max\{\bar{x}_{ij}, \bar{x}_{ji}\}$

W2 $w(\{i, j\}) = (1 - \max\{\bar{x}_{ij}, \bar{x}_{ji}\}) \cdot c(\{i, j\})$

Overall, our algorithm computes (if successful) a feasible undirected solution. Its solution value can be used an upper bound in our branch-and-cut framework. If, for some reason, we need a corresponding oriented solution, we can use our orientation procedure (cf. Definition 10.6) or any s, t-numbering algorithm w.r.t. any chosen arc incident to r.

Chapter 12

Experiments

We implemented our DCut based branch-and-cut (*DC*) and a DFlow based branch-and-bound (*DF*) algorithm using CPLEX 9.0's branch-and-bound framework. Both implementations are able to solve 2ECON, 2NCON, 2RSN and their prize-collecting variants. The additionally necessary separation routines for DC are implemented in C++ using LEDA 5.1.1 and the efficient maximum flow algorithm of [CG97]. All tests were performed on an Intel Xeon 5140 2.33Ghz CPU in 32 bit mode. Each process is restricted to a single core, 2 GB RAM, and 2 hours computation time per problem instance.

Our computational study concentrates on the instances of the {0,1,2}-SND problems with node-connectivity requirements for which the directed formulations are stronger from the polyhedral point of view, cf. Corollary 10.16. E.g., we do not consider instances for 2R(PC)SN problems with $\mathcal{R}_1 = \emptyset$ or 2NCON problems on complete graphs with Euclidean costs with $\mathcal{R}_1 = \emptyset$. Our goal is to experimentally confirm the strength of our orientation-based formulations for node-connectivity, and show its general applicability in practice.

In Section 12.1 we suggest a benchmark library TSNDLib that can also be used for further research on {0,1,2}-SND problems. In Section 12.2 we experiment with different parameter settings for the DC algorithm using a representative subset of the former benchmark library. The performance of DC is then analyzed and compared to the performance of DF, cf. Sections 12.3 and 12.4. For the 2RPCSN problem we use a superset of the instances used in [WRP$^+$07, WRP$^+$06]; we can experimentally compare our algorithms to those based on undirected formulations given in those publications.

12.1 Benchmark Instances: TSNDLib

In general, until now only few computational results for {0,1,2}-SND problems with node-connectivity requirements were published in the literature and there is no common benchmark set of instances, cf. Section 9.5. Hence, one of our additional aims was to create such a benchmark library.[1] We therefore collected the available

[1] A known library of test instances SNDLib [OPTW07] provides instances for survivable *capacitated* network design problems, i.e., problems where the aim is not only to topologically design

test instances used by various authors for different problem settings and included them—together with several new instances—in our TSNDLib (*Topological Survivable Network Design Library*) [TSN08]. On the cited web page, one can also find information on the computational results conducted on each of the benchmark sets. In the following, we briefly describe these instances and discuss their usefulness for our computational study.

Most real-world applications of {0,1,2}-SND problems seem to be based on rather sparse graphs [Bac05, Sto92]. For the 2NCON problem, Stoer [Sto92] used a set of sparse real-world instances with up to 116 nodes. Unfortunately, this data is not available anymore[2]. On the other hand, our results indicate that such small and sparse networks are usually solved within less than a second.

For the 2R(PC)SN problems, Bachhiesl [Bac05] proposed the following three different benchmark sets that have also been used in [WRP+06, WRP+07, LR08, LRP09]. These instances can be used not only for the 2R(PC)SN but also for the 2NCON problem, interpreting the given root node as an $\mathcal{R}_2$ customer. Some of these instances are infeasible for 2NCON as the underlying graph does not allow two node-disjoint paths between certain customers. Hence, for 2NCON, we report only on the feasible instances.

ClgS and ClgM. These instances use the real-world access net data of the city district Cologne-Ossendorf. For our experiments we consider the small (ClgS) and medium (ClgM) sized instance sets. Thereby, each set contains 25 instances. The underlying graphs have 190 nodes and 377 edges, and 1757 nodes and 3877 edges, respectively. The instances differ in the choice of customer nodes, and have 3–6 $\mathcal{R}_1$, and 2–3 $\mathcal{R}_2$ customers.

Grid. This set contains artificial instances based on grid graphs with 100, 400, 900,..., 4900 nodes. For each graph size there are 2×15 instances, using two different cost functions, respectively. They differ in their choice of the 5–13 $\mathcal{R}_1$ and 3–8 $\mathcal{R}_2$ customers, respectively.

PCSTLib+. The PCSTLib benchmark [JMP00] for the prize-collecting Steiner tree problem was used in several studies, e.g., [JMP00, LWP+06, LR03], and contains random graphs divided into two groups K and P, where 15%–27% and 34%–50% of the nodes are ($\mathcal{R}_1$) customers, respectively. Both classes are constructed such that the resulting graphs would have roughly constant expected average degree and (average non-zero prize)/(average edge cost) ratios. The average degree is roughly 6 for P and 7–8 for K instances. Beyond this, the graphs of K are designed to be similar to street map layouts, those of P do not have any specific structure. More details on the corresponding instances can be found in [JMP00]. For the {0,1,2}-SND problems, [Bac05] generated PCSTLib+ by considering the underlying graphs with 100, 200, and 400 nodes in each group and modifying them such that roughly 1/3 of the customer nodes are selected to be in $\mathcal{R}_2$.

---

a network, but demand routing and capacity issues have to be considered. Consequently, those graphs are usually smaller than the graphs we want to consider for {0,1,2}-SND problems.

[2] Personal communication.

Due to the diversity of these instances and the partial real-world aspect, we take them as a basis for our computational study. As the original Cologne and Grid instances have rather few customers—which seems unusual in practice—we also generated modified instances:

***ClgS*$^+$.** These are the ClgS instances, selecting 20% (10 % $\mathcal{R}_1$ and 10% $\mathcal{R}_2$) of the nodes as customers.

***Grid*$^+$.** These are the Grid instances, selecting 20% (10 % $\mathcal{R}_1$ and 10% $\mathcal{R}_2$) of the nodes as customers.

Besides, Wagner [Wag07] proposed an additional artificial benchmark set:

***T*$^+$.** Considering the well-known TSPLib (see below), these instances are corresponding Euclidean Delaunay triangulations on varying graph sizes where 25% (10%) of all nodes are $\mathcal{R}_1$ ($\mathcal{R}_2$) customers.

Even though most real-world applications seem to be based on rather sparse graphs, other papers also consider dense and complete graphs [KMN04, Rag95, GR02, MS89]. Thus, in order to test the general applicability of our algorithm and to create a diverse benchmark library, we also use the following benchmark set:

***TSPLIB*$^+$.** This set contains complete graphs with Euclidean weights from the TSPLIB benchmark library that was originally proposed for the traveling salesman problem [TSP]. In the context of the 2CON problems, it was used by Kerivin et al. [KMN04]. We generate instances with the following customer distribution ($\%\mathcal{R}_1, \%\mathcal{R}_2$): (0,25), (0,50), (0,100), (10,10), (25,25), (25,75), (50,50), (75,25). Recall that for 2NCON instances consisting of complete graphs ($G = V, E$) with Euclidean edge-weights and $\rho_v \in \{0, 2\}$ or $\rho_v \in \{2\}$ for all graph nodes v (used, e.g., in [KMN04]), a 2ECON model is sufficient to generate 2NCON solutions. In this case, the (more compact) undirected model is preferable from a theoretical point of view, cf. Corollary 10.16. Nevertheless, we included such instances in our benchmark library in order to give other researchers the possibility to test and compare the special purpose algorithms for the corresponding problems. In this thesis, however, we present computational results on instances with $\mathcal{R}_1 \neq \emptyset$. Furthermore, as the set TSPLIB$^+$ is very unlikely in real-world scenarios, we test the performance of our algorithm on these instances in a separate section.

12.2 Tuning of DC

We first want to investigate on the impact of different parameter settings on the running time of DC for 2NCON and 2RPCSN. We therefore use a subset of our benchmark instances that contains 2–3 instances of each type. To obtain an unskewed comparison, we turned off all automatic cut-generation etc., usually performed by CPLEX.

We first consider minimum cardinality cuts (mc-cuts), i.e., the cuts containing the smallest number of arcs among all most violated cuts of type (10.10), (10.11)

			2NCON				2RPCSN			
			#o		$t(s)$		#o		$t(s)$	
	$\|V\|$	#i	mc 0	mc 1	mc 0	mc 1	mc 0	mc 1	mc 0	mc 1
ClgS	190	3	3	3	0.31	0.06	3	3	0.21	0.06
ClgS$^+$	190	3	3	3	3.17	0.35	3	3	0.00	0.01
ClgM	1757	3	0	2	—	177.86	1	2	5252.89	152.72
ClgM$^+$	1757	2	0	0	—	—	2	2	0.22	0.23
Grid	100	2	2	2	0.15	0.05	2	2	0.09	0.04
	400	2	2	2	16.59	2.12	2	2	13.38	1.32
	900	2	2	2	993.18	53.85	2	2	385.88	40.35
	2500	2	2	2	1872.80	114.99	2	2	3370.47	163.98
	4900	2	0	1	—	4973.29	0	1	—	2392.59
Grid$^+$	100	2	2	2	0.61	0.61	2	2	0.13	0.06
	400	2	2	2	37.97	7.78	2	2	12.28	3.30
	900	2	0	0	—	—	1	1	1723.13	1746.38
T$^+$	<200	2	2	2	4.78	2.57	2	2	5.03	3.02
	200-400	5	4	5	77.28	17.03	5	5	34.94	14.30
	>400	2	1	1	415.63	170.80	2	2	481.31	127.81
K	100	3	3	3	1.30	0.75	3	3	0.96	0.59
	400	3	2	2	111.67	45.29	3	3	274.34	119.66
P	100	5 (4)	4	4	0.50	1.02	5	5	0.29	0.45
	200	1	1	1	2.73	3.11	1	1	1.91	3.37
	400	5 (1)	1	1	394.99	205.93	5	5	116.99	210.40

Table 12.1: Comparison of average running times of DC with (mc 1) and without (mc 0) using mc-cuts within our separation routine for a representative subset of instances. For each set we also give the number of tested instances (#i), indicating the number of 2NCON feasible instances in brackets. #o denotes the number of instances that could be solved to optimality within 2 hours. The times are averaged over those instances that were solved by both algorithm variants (if both were successful at least once).

or (10.12). Table 12.1 shows that separating mc-cuts, instead of just any identified ones, is crucial for solving large graphs efficiently in practice. In fact, this speeds up our algorithm on all instance sets except for the P instances. This speed-up is particularly impressive for ClgM and large Grid instances. Based on these result, we decide to use mc-cuts for all instances, except for the set P, for our further experiments.

Further experiments on the set TSPLIB$^+$, see Section 12.6, show that using mc-cuts is only advantageous for the setting where $\mathcal{R}_1$ and $\mathcal{R}_2$ contain 10% of the graph nodes each. For the other settings, i.e., when there are 50% or 100% customer nodes, the use of mc-cuts significantly slows down the performance of DC, especially for larger input graphs. As this finding is consistent to the results for the P instances (which are also more dense and contain more customers) we deduce that the use of mc-cuts is only beneficial for sparse graphs with a small percentage of customers.

We further compared the running times of DC, starting with different initial ILPs. Thereby we considered the following settings:

I1. We start with the constraints (10.9) that ensure unique orientations of edges,

and the forward dcut-constraints (10.10) for $|S| = 1$, i.e.,

$$x(\delta^-(v)) \geq y_v \quad \forall v \in \mathcal{R}'.$$

I2. Additionally to the setting I1, we add the backward dcut-constraints (10.11) for $|S| = 1$, i.e.,

$$x(\delta^+(v)) \geq y_v \quad \forall v \in \mathcal{R}'_2.$$

I3. Additionally to the setting I2, we add the flow-preservation constraints (11.1).

In Table 12.2 we summarize the performance of the corresponding algorithmic variants. For instances with a very small percentage of customers, using flow-preservation constraints seem to be very intuitive, as their solutions usually contain long paths between each pair of customers. Our results show that using the setting I3, we can significantly reduce the running time not only for the latter instances, but for nearly all benchmark sets. For TSPLIB$^+$ instances (again see Section 12.6 for details) the performance difference between I2 and I3 is rather insignificant. In fact, for $\mathcal{R}_0 = \emptyset$ the setting I2 and I3 naturally coincide. We use the setting I2 for all TSPLIB$^+$ instances, otherwise we always use I3.

We also experimented with primal heuristics developed in Section 11.3.2 within our DC algorithm. In Table 12.3 we show the results of the following three algorithm variants:

H0 We run DC without using any heuristic.

H1 We use our LP-based heuristic using the strategy W1 to define the edge costs, cf. Section 11.3.2.

H2 We use our LP-based heuristic using the strategy W2 to define the edge costs, cf. Section 11.3.2.

Intuitively, the use of a primal heuristic can be advantageous for instances with a large gap between the value of the LP relaxation and the optimal solution, i.e., instances that require a large branch-and-bound tree. Our analysis of DC shows that this is the case for the instances with a high number of customers. However, it turns out that even in such cases, the running times of DC using the settings H1 or H2 are even higher than the running times for pure DC. Hence, we will always use H0 in the following.

12.3 Comparison of DFlow and DCut

In this section, we compare the performance of DF and DC. Again, to obtain an unskewed comparison, we turned off all automatic cut-generation etc., usually performed by CPLEX. Furthermore, we use the parameter settings as described in the above section. Our experiments show that DC outperforms DF in terms of running time and success ratio on all instance sets and for all considered {0,1,2}-SND problem variants.

			2NCON						2RPCSN					
			#o			$t(s)$			#o			$t(s)$		
	$\|V\|$	#i	I1	I2	I3	I1	I2	I3	I1	I2	I3	I1	I2	I3
ClgS	190	3	3	3	3	0.06	0.05	0.05	3	3	3	0.06	0.06	0.05
ClgM	1757	3	2	2	3	177.86	171.96	69.68	2	2	3	96.89	190.14	71.96
ClgM$^+$	1757	2	0	0	0	—	—	—	2	2	2	0.23	0.23	0.23
ClgS$^+$	190	3	3	3	3	0.35	0.33	0.22	3	3	3	0.01	0.00	0.01
Grid	100	2	2	2	2	0.05	0.04	0.04	2	2	2	0.04	0.03	0.03
	400	2	2	2	2	2.12	2.42	1.82	2	2	2	1.32	1.40	1.01
	900	2	2	2	2	53.85	29.90	14.44	2	2	2	40.35	38.45	17.78
	2500	2	2	2	2	114.99	104.32	62.52	2	2	2	163.98	185.75	96.67
	4900	2	1	1	1	4973.29	3288.89	3981.79	1	1	1	2392.59	2152.26	1658.38
Grid$^+$	100	2	2	2	2	0.61	0.51	0.28	2	2	2	0.06	0.06	0.05
	400	2	2	2	2	7.78	8.10	5.26	2	2	2	3.30	3.05	1.92
	900	2	0	0	0	—	—	—	1	1	2	1746.38	403.94	378.27
T$^+$	<200	2	2	2	2	2.57	0.89	0.82	2	2	2	3.02	2.19	1.92
	200–400	5	5	5	5	482.29	137.17	87.62	5	5	5	14.30	21.11	9.61
	>400	2	1	1	1	170.80	118.44	150.14	2	2	2	127.81	83.44	84.14
K	100	3	3	3	3	0.75	0.68	0.68	3	3	3	0.59	0.57	0.45
	400	3	2	3	2	45.29	35.07	32.85	3	3	3	119.66	84.05	85.34
P	100	5(4)	4	4	4	0.50	0.43	0.35	5	5	5	0.29	0.30	0.19
	200	1	1	1	1	2.73	1.99	1.49	1	1	1	1.91	1.70	1.13
	400	5(1)	1	1	1	394.99	295.55	51.94	5	5	5	116.99	570.52	42.70

Table 12.2: Comparison of average running times of DC with different settings for the initial ILP. We used the mc-cuts for all instances, except for P.

			2NCON						2RPCSN					
			#o			$t(s)$			#o			$t(s)$		
	$\|V\|$	#i	H0	H1	H2	H0	H1	H2	H0	H1	H2	H0	H1	H2
ClgS	190	3	3	3	3	0.05	0.06	0.06	3	3	3	0.05	0.04	0.05
ClgS$^+$	190	3	3	3	3	0.22	0.24	0.23	3	3	3	0.01	0.01	0.00
ClgM	1757	3	3	2	2	69.68	102.72	110.69	3	2	2	71.96	119.85	131.91
ClgM$^+$	1757	2	0	0	0	—	—	—	2	2	2	0.23	0.26	0.26
Grid	100	2	2	2	2	0.04	0.03	0.04	2	2	2	0.03	0.03	0.03
	400	2	2	2	2	1.82	1.79	1.73	2	2	2	1.01	1.07	1.08
	900	2	2	2	2	14.44	22.49	18.96	2	2	2	17.78	15.85	15.84
	2500	2	2	2	2	62.52	114.61	103.18	2	2	2	96.67	111.95	111.63
	4900	2	1	1	1	3981.79	3266.41	3609.35	1	1	1	1658.38	1528.08	1973.78
Grid$^+$	100	2	2	2	2	0.28	0.33	0.34	2	2	2	0.05	0.05	0.05
	400	2	2	2	2	5.26	4.76	4.72	2	2	2	1.92	2.01	2.04
	900	2	0	0	0				2	2	2	961.00	1388.18	1243.42
T$^+$	<200	2	2	2	2	0.82	0.95	0.89	2	2	2	1.92	3.72	3.72
	200–400	5	5	5	5	87.62	81.96	80.68	5	5	5	9.61	21.94	21.90
	>400	2	1	1	1	150.14	153.57	123.63	2	2	2	84.14	947.47	956.65
K	100	3	3	3	3	0.68	1.42	1.72	3	3	3	0.45	2.26	2.78
	400	3	2	2	2	32.85	58.18	72.33	3	3	3	85.34	286.22	437.67
P	100	5(4)	4	4	4	0.35	0.41	0.46	5	5	5	0.19	0.16	0.17
	200	1	1	1	1	1.49	1.43	1.45	1	1	1	1.13	0.99	1.02
	400	5(1)	1	1	1	51.94	56.61	58.82	5	5	5	42.70	58.00	76.40

Table 12.3: Comparison of average running times of DC with (and without) different heuristics. We use the mc-cuts for all instances except for the set P, and the setting I3 for the initial LP.

		ClgS	ClgS$^+$	Grid				Grid$^+$	K	P
		190	190	100	400	900	1600	100	100	100
2NCON	**DC**	**0.06**	**0.44**	**0.10**	**1.54**	**17.73**	**86.15**	**0.13**	**0.69**	**0.35**
	DF	0.3	446.3	7.0	226.0	1505	(6209)*	22.4	19.9	1500
2RSN	**DC**	**0.07**	**0.4**	**0.08**	**1.3**	**17.4**	**94.0**	**0.09**	**1.0**	**0.7**
	DF	0.4	154.5	6.4	240.2	1554	—	30	13.1	2194
2RPCSN	**DC**	**0.06**	**0.01**	**0.08**	**1.4**	**12.7**	**74.6**	**0.06**	**0.35**	**0.18**
	DF	0.3	7.6	5.9	261.3	1778	—	8.6	19.1	1136

Table 12.4: Average CPU time in seconds for instance groups solved by both DC and DF. **K** and **P** denote the groups of PCSTLib$^+$. "*" DF solves only 67% of the instances. None of the instances not in this table can be solved by DF within 2 hours.

		ClgM	Grid			Grid$^+$		K	P
		1757	2500	3600	4900	400	900	400	400
2NCON	o/i	24/25	30/30	30/30	24/30	30/30	11/30	4/6	1/1
	avg	(777)	294	746	(2671)	37	(1267)	(1466)	52
	med	(51)	144	458	(2051)	4	(297)	(34)	52
2RSN	o/i	24/25	30/30	30/30	23/30	30/30	5/30	3/6	1/1
	avg	734	141	423	(2307)	31	(645)	(53)	84
	med	(53)	109	480	(2156)	8.0	(652)	(52)	84
2RPCSN	o/i	24/25	30/30	30/30	26/30	30/30	26/30	11/11	1/1
	avg	(763)	129	518	(2459)	7.0	(742)	209	38.21
	med	(41)	131	458	(1913)	4	(376)	165	26

Table 12.5: Average and median running times for instance groups only solved by DC. Times in brackets denote that not all instances are solved within 2 hours. "o/i" gives the number of solved instances and the total number of instances, respectively. None of the instances left of the double vertical line require branching, except for one Grid instance with 3600 nodes and three Grid instances with 4900 nodes. For PCSTLib$^+$, we report only on feasible instances.

An overview on the average running times of DC and DF is given in Table 12.4. We thereby only report on the instance groups which can—at least in part—be solved to optimality by both DC and DF. Table 12.5 shows the results for the other (larger) instance groups that can be solved only by DC. Note that, except for 2NCON on grids with 1600 nodes, all groups where either completely solvable by DF, or DF solved none of its instances.

We observe that the runtime performance of DC is quite similar on the different problem classes 2NCON, 2RSN and 2RPCSN. The same holds for DF. All instances with less than 200 nodes are solved to optimality by DC in less than a second on average. The only instance set where both algorithms perform comparably well is ClgS, which is due to the fact that the underlying LPs of DFlow are rather small due to the small number of customers, and the overhead of DC's cut separation routines is comparably expensive. Note that neither approach requires any branching for ClgS and small Grid instances. Already a slight increase in the number of customers is sufficient for DC to outperform DFlow, see, e.g., the results

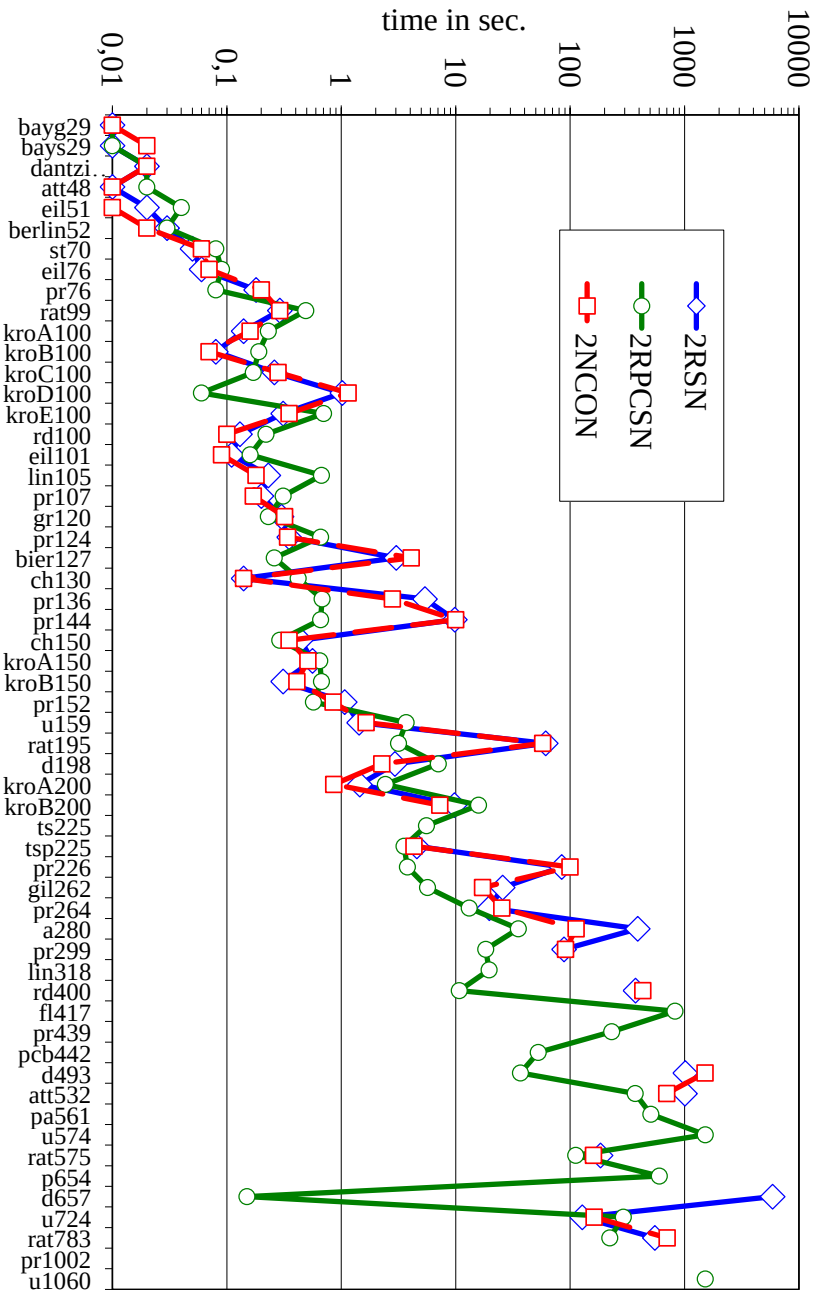


Figure 12.1: Running time (in seconds) of DC for T^+, comparing different problem settings. Missing data points denote that the corresponding problem was not solved to provable optimality within 2 hours.

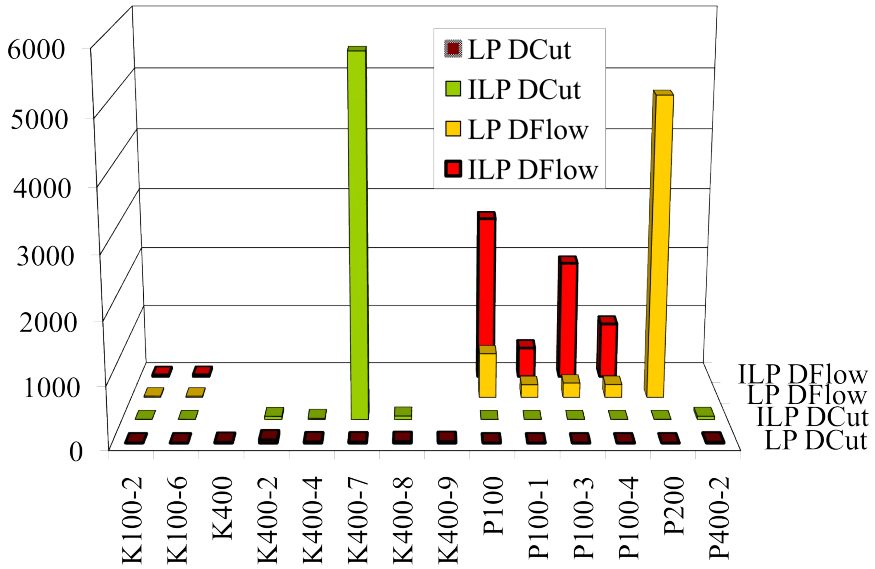


Figure 12.2: 2NCON for PCSTLib$^+$. Time required (in seconds) to solve the DFlow and DCut ILPs and their corresponding LP relaxations. Data points are missing when the computation exceeded the 2 hours timeout limit.

for Grid instances of size 100. This effect is further amplified by larger underlying graphs, as this results in an even larger increase of variables for DFlow. While the cut approach is able to solve all Grid instances with up to 3600 nodes to optimality within 1 hour, the largest instances which can be completely solved by DFlow in 2 hours contain 900 nodes. We see that, due to their high number of customers, DC is highly advantageous even for the smallest graphs of PCSTLib$^+$: for the P group with 100 nodes, it is 2000–3000 times faster than DFlow.

For the T$^+$ instances, the findings are consistent with the ones just reported, as only DC was able to solve instances. Interestingly, DC solves all instances with up to 1000 nodes within two hours in the context of 2RPCSN, while some of these instances turn out to be more difficult for the 2RSN and 2NCON settings. Figure 12.1 shows the respective running times of DC for the different problem settings.

A further common measure to assess and compare ILP formulations is to look at the lower bounds resulting from their LP relaxations, i.e., the solution at the root node of the branch-and-bound tree (LP$_r$). In our case, these values are identical as the corresponding polytopes are equivalent, cf. Theorem 10.12.

DC also outperforms DF in terms of running times needed to solve the (equivalent) LP relaxation. When DF is not able to solve the given instance to optimality within 2 hours, it is due to a large size of the LP and the most part of the computation time is needed to solve the root relaxation. In contrast to this, when branching is required, DC uses only a comparably small percentage of the total running time to solve the root relaxation. For the Grid$^+$ instances with 400 nodes, DF cannot

	100			190	400				900
	Grid$^+$	**K**	**P**	**ClgS$^+$**	**Grid$^+$**	**K***		**P**	**Grid^{+**}**
$\mathbf{t_{ILP}}$	**0.13**	**0.69**	**0.35**	**0.44**	**37.3**	**1466**	—	**51.8**	**1267** —
$\mathbf{t_{LP}}$	0.11	0.64	0.18	0.30	8.1	24.6	18.9	14.5	82.2 —
$\mathbf{gap_{lp}}$	0.16	0.09	0.18	0.12	0.35	0.21	0.61	0.34	0.26 ***
#BB	0.9	2.2	14.3	1.3	47.2	557	864	114	27.1 —
r.br.	16.7	33.3	100	42.1	100	100	100	100	100 100

Table 12.6: Results for 2NCON computed via DC, when branching was required. "*" and "**": only 66.7% and 33.7% were solved, respectively. The left/right column gives the statistics of the solved/unsolved instances after 2h. For the latter we use an upper bound (found by DC without any primal heuristic) to estimate the gap. The statistics corresponding to "***" can be found on page 139.

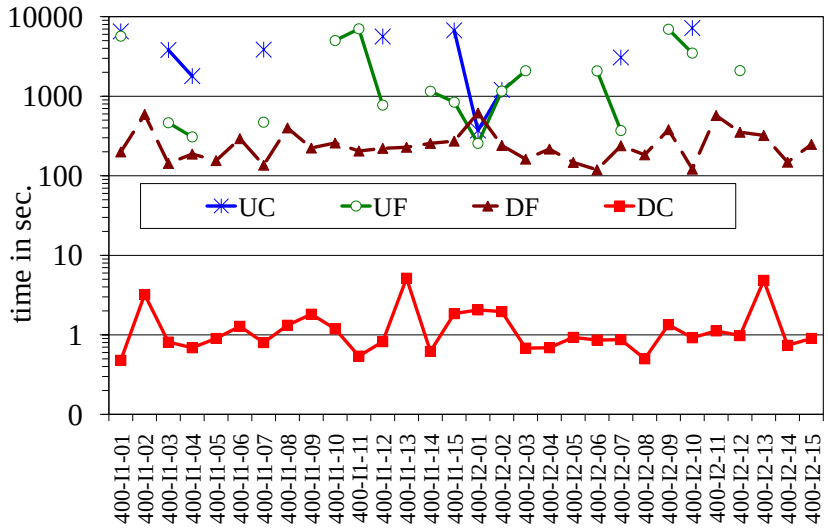
even solve the first LP relaxation within the given time bound. The DC algorithm, on the other hand, requires only 37 seconds on average to solve an ILP, whereby the corresponding LP relaxation is solved within 8 seconds. In Figure 12.2, we visualize these observations w.r.t. 2NCON and the PCSTLib$^+$ instances. The results for 2R(PC)SN are analogous.

12.4 Analysis of DCut Performance

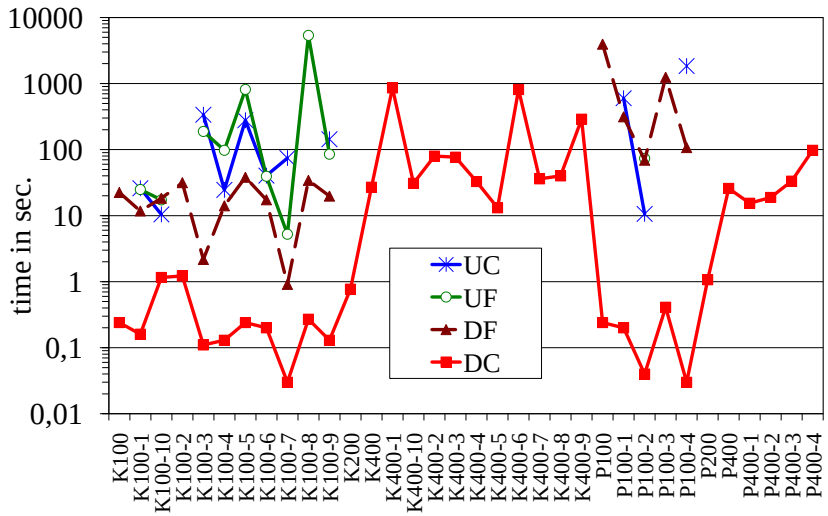
As the performance of our DC algorithm is similar for different problem settings, we only consider its performance for the most prominent 2NCON problem in the following. We observe that the LP relaxation of our ILPs usually give strong lower bounds. For many instances—i.p. all Grid and ClgS instances—the relaxation already gives an integral and thus optimal solution. In Table 12.6, we report on the quality and time (t_{LP}) of the solutions at the root node, i.e., the LP relaxation of the full model. For each set we compute the average relative $\mathbf{gap_{lp}} := \frac{(\mathrm{OPT}-\mathrm{LP}_r)}{\mathrm{OPT}}$ in percent, whereby OPT denotes the optimal objective value of the ILP. Additionally, we give the average total runtime t_{ILP}, the average percentage of instances that require branching (r.br.), and the average number of branch-and-bound nodes (#BB). Note that, within 2 hours, DC solves 4/6 of the K instances with 400 nodes. However, DC is able to find an integral upper bound (UB) for all unsolved instances, without the use of a heuristic. Hence, we also give an average relative $\mathbf{gap^*_{lp}} = \frac{(\mathrm{UB}-\mathrm{LP}_r)}{\mathrm{UB}}$. We optimally solve 11 out of 30 Grid$^+$ instances with 900 nodes within 2 hours. The other 19 instances cannot be solved as we ran out of memory after 2555 seconds on average. For 17 unsolved instances, DC manages to compute the LP relaxation and for 5 instances also integral upper bounds are found. For the latter instances, we can hence compute the above defined $\mathrm{gap}^*_{\mathrm{lp}} = 0.5\%$.

12.5 Directed vs. Undirected Models for 2RPCSN

For the 2RPCSN problem we compared our results for DF and DC with the running times of the algorithms based on the undirected formulations published

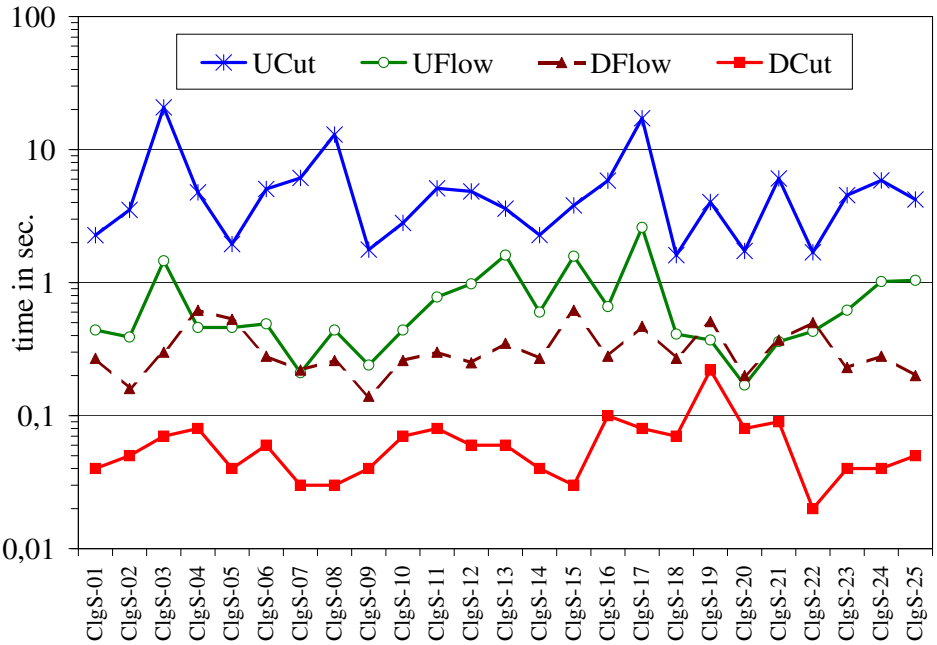


(a) Grid instances with 400 nodes



(b) PCSTLib$^+$ instances

Figure 12.3: Diagrams comparing UCut and UFlow approaches to our DCut and DFlow formulations.



(c) *ClgS* instances

Figure 12.3: Continued.

in [WRP+06, WRP+07]. Note that although the latter algorithms were run on a stronger Intel Xeon 3.6 GHz machine with the new version CPLEX 10.0.1 and LEDA 5.1.1, DC clearly outperforms them on all tested instances, cf. Figure 12.3. DC solves all Grid instances with up to 3600 nodes and most of the instances with 4900 nodes to provable optimality. For the previous approaches, the largest solvable Grid instance has 400 nodes and the required running times are much longer, cf. Figure 12.3(a). This is of particular interest, as one expects flow formulations to perform quite well on instances with such a small percentage of customers.

12.6 Analysis of DC on TSPLIB+

Our main goal was to develop a fast exact algorithm able to solve large real-world instances. As we mentioned before, the underlying graphs of such instances are typically sparse. Nevertheless, we also tested the performance of DC for 2NCON on complete graphs with different customer settings in order to evaluate the overall effectiveness of our algorithm. We therefore used TSPLIB+ instances, as such similar instances were already used in the literature in the context of 2CON problems [KMN04, MS89]. Furthermore, random complete graphs were used in [Cho92, GR02, CA95]. Since the previously used customer choices are not available to us, we cannot fairly compare our results to other published results.[3] Testing different

[3]We hence encourage others to use TSNDLib in the future, to avoid problems with random instances without a published deterministic generator based on pseudo-random numbers.

parameter settings on a small sample set of TSPLIB$^+$ instances, we found out that using mc-cuts never pays of, which is the main difference to the previously tested sparse instances with a small number of customers. Furthermore, based on the above test results, we decided to use the I2 setting for the choice of the initial LP and not to use any primal heuristic within our algorithm.

Table 12.7 summarizes the results of DC on TSPLIB$^+$ instances. We can observe that the total running time increases with the overall ratio of customers. On the other hand the ratio of customers being $\mathcal{R}_2$ customers does not clearly impact the algorithm's performance: all three instance types with 100% customers behave similarly. Interestingly, we observe that the LP relaxations for the 100% customer instances are typically obtained faster than for the (25%,25%) instances. Yet, the weaker LP gap and the consequently larger branch-and-bound tree reverse this potential advantage. Overall we note that even complete instances with many (or all) nodes being customers can be solved consistently up to the size of 150 nodes.

Note that since our approach is designed for the {0,1,2}-SND problems, it can also be used to solve the special cases where the vector connectivity requirements ρ are restricted, cf. Section 9. Indeed, except for complete Euclidean problems, our DCut formulation is the strongest known formulation also for the minimum 2-node-connected Steiner network and minimum 2-node-connected spanning subgraph problems. However, in these cases, it can be advantageous to use algorithms that are tailored for the special problem structure.

Table 12.7: Results for the TSPLIB$^+$ with many customers.

$(\mathcal{R}_1, \mathcal{R}_2)$	(10,10)				(25,25)				(25,75)				(50,50)				(75,25)			
inst, $\lvert V\rvert$	o	**ILP**	LP	g_{lp}	o	**ILP**	LP	g_{lp}	o	**ILP**	LP	g_{lp}	o	**ILP**	LP	g_{lp}	o	**ILP**	LP	g_{lp}
eil51	5	**0.52**	0.52	0.00	5	**0.57**	0.35	0.18	5	**17.94**	0.43	1.00	5	**0.92**	0.45	0.21	5	**0.29**	0.27	0.03
st70	5	**3.12**	3.12	0.00	5	**1.73**	1.73	0.00	5	**8.03**	1.70	0.68	5	**6.87**	2.09	0.50	5	**4.21**	1.88	0.33
eil76	5	**3.80**	3.74	0.04	5	**12.37**	3.46	0.45	5	**117.87**	4.46	0.47	5	**37.45**	3.24	0.35	5	**14.76**	2.34	0.21
pr76	5	**6.06**	5.54	0.07	5	**24.13**	3.29	0.59	5	**138.02**	2.65	0.97	5	**118.44**	2.44	0.95	5	**91.80**	1.59	0.81
rat99	5	**33.18**	33.18	0.00	5	**38.16**	11.71	0.40	5	**55.10**	5.00	0.42	5	**32.96**	6.58	0.25	5	**17.29**	6.53	0.16
kroA100	5	**18.78**	18.78	0.00	5	**18.27**	16.11	0.04	5	**143.61**	8.63	0.94	5	**131.19**	7.03	0.66	4	**12.80**	7.61	0.23
kroB100	5	**36.45**	36.45	0.00	5	**14.67**	12.97	0.05	5	**240.68**	7.29	0.90	5	**46.03**	7.29	0.58	5	**20.69**	7.84	0.17
kroC100	5	**21.42**	21.42	0.00	5	**14.54**	12.04	0.11	5	**70.54**	6.73	1.05	5	**47.39**	9.07	0.82	5	**20.75**	7.81	0.43
kroD100	5	**27.35**	27.35	0.00	5	**15.93**	13.15	0.09	5	**50.68**	8.06	0.55	5	**46.18**	7.32	0.62	5	**8.97**	6.51	0.13
kroE100	5	**30.75**	30.60	0.01	5	**10.29**	9.58	0.03	5	**443.43**	8.39	1.01	5	**95.15**	8.91	0.57	5	**23.41**	8.32	0.29
rd100	5	**27.63**	26.36	0.01	5	**101.87**	9.35	0.37	5	**16.82**	7.58	0.33	5	**68.84**	8.06	0.55	5	**11.34**	6.80	0.08
lin105	5	**63.46**	63.46	0.00	5	**32.83**	16.56	0.22	5	**10.27**	9.17	0.07	5	**12.22**	9.32	0.12	5	**12.62**	10.61	0.09
pr107	5	**32.67**	32.67	0.00	5	**19.64**	19.06	0.02	5	**11.18**	10.54	0.00	5	**8.05**	8.05	0.00	5	**5.99**	5.99	0.00
pr124	5	**176.80**	115.66	0.07	5	**66.03**	50.46	0.10	5	**100.61**	16.28	1.31	5	**52.22**	18.46	0.96	5	**38.27**	16.70	0.41
bier127	5	**89.61**	85.24	0.06	5	**157.07**	78.46	0.15	5	**86.91**	33.57	0.36	5	**211.49**	35.19	0.29	5	**102.33**	29.48	0.13
ch130	5	**110.24**	107.78	0.02	4	**80.96**	52.73	0.10	5	**261.28**	35.87	0.37	5	**315.64**	32.23	0.26	5	**50.12**	28.44	0.16
pr144	5	**165.75**	165.75	0.00	4	**81.14**	85.57	0.00	5	**292.94**	44.65	0.58	5	**854.61**	48.18	0.84	5	**347.02**	41.57	0.56
ch150	5	**134.71**	134.71	0.00	5	**968.74**	76.00	0.26	2	**2168.74**	59.25	0.35	1	**99.36**	46.66	0.26	5	**1043.69**	55.07	0.57
pr152	3	**760.79**	520.79	0.10	0	—	207.49	—	4	**893.08**	47.70	0.61	5	**1466.53**	53.61	0.65	5	**1560.69**	69.07	0.66
rat195	4	**706.69**	589.33	0.07	2	**237.61**	220.53	0.14	0	—	151.71	—	0	—	171.21	—	4	**1231.29**	163.14	0.13
d198	5	**3251.52**	3200.65	0.02	4	**1799.87**	1132.92	0.07	2	**1528.01**	198.98	0.09	4	**1247.43**	211.85	0.18	5	**707.11**	243.52	0.12
kroB200	4	**2091.73**	1676.50	0.09	1	**452.11**	475.83	0.00	0	—	190.41	—	5	**877.62**	227.91	0.28	4	**545.33**	193.10	0.17
ts225	3	**1001.33**	998.46	0.00	1	**2198.20**	432.60	0.13	0	—	163.89	—	0	—	322.68	—	0	—	338.00	—
pr264	5	**4636.08**	4636.08	0.00	1	**1904.72**	1825.91	0.04	3	**4856.05**	484.83	0.31	5	**2125.86**	468.38	0.21	5	**681.01**	585.56	0.06
pr299	4	**5367.26**	5239.67	0.00	0		3200.01	—	0	—	944.56	—	0	—	1302.13	—	0	—	1259.68	—
lin318	2	**5966.71**	6707.46	0.00	1	**5917.81**	5619.80	0.01	0	—	2049.47	—	0	—	2360.41	—	0	—	2461.59	—
u574	0	—	7237.05	—	0	—	7234.46	—	0	—	7292.77	—	0	—	7294.54	—	0	—	7271.86	—

The number in the instance names give the number of nodes. "o" denotes the number of optimally solved instances out of 5 (per instance name and customer setting). LP times are averaged over all 5 instances.

Part IV

Epilogue

Chapter 13

Conclusions and Outlook

In this thesis we designed exact algorithms for topological network design problems such as the k-cardinality tree and $\{0,1,2\}$-survivable network design problems. Therefore, we derived new strong ILP models based on orientation properties of feasible solutions and used them within a branch-and-cut scheme. Our ILP models are the strongest known models for the considered problems: our extensive polyhedral analysis shows that, whereas there exists an undirected formulation for the k-cardinality tree problem which is equivalently strong (but has some disadvantages in practice), our model for the $\{0,1,2\}$-survivable network problems with node-disjointness is strictly stronger than all previously known models. Thus, we answered the long open question whether there exists an orientation property of 2-node-connected graphs that can be used to derive stronger ILPs for the 2NCON problem.

Our computational studies show that our algorithms solve almost all known instances to provable optimality within reasonable time bounds. For EKCT it optimally solves all instances of KCTLIB and is even faster than the state-of-the-art metaheuristics for instances with up to 1000 nodes. The results of this thesis show that recent advances in computational power and ILP solvers, if used in conjunction with strong ILP formulations, allow exact algorithms to become feasible or even preferable alternatives to other, e.g., metaheuristic, approaches. Our results can therefore be seen as an example that we should not easily give up on exact algorithms simply because the problem is NP-hard or because old approaches did not show their practical applicability.

For most of the known instances of KCT and $\{0,1,2\}$-SNDP problems, our algorithms are the first to compute provably optimal solutions. Hence, using these optimal solutions it is now possible to better evaluate current and future heuristic methods and lower bounding procedures than before.

Further constraints. For the KCT problem, we found out that grid instances with small cardinalities k are generally harder to solve than other instances. Furthermore, node-weighted instances are harder than their edge-weighted counterparts which seems to be be due to the symmetries in the resulting KCA instance as all arcs with the same target node have identical costs. It would be worthwhile to investigate whether this structure can be exploited via additional constraints, further

strengthening the Kca′-DCut formulation.

Our results for KCT furthermore show that usually most of the CPU time is spent for computing the LP relaxation and that the relative gap between this lower bound and the optimum solution is quite small. However, this is not the case for small values of k. On the other hand, for such instances the solution is significantly smaller than the original graph. It may be that special preprocessing routines or strengthening inequalities can be designed to identify parts of the graph that can be safely excluded from the search space as they cannot be contained in the optimal solution.

In contrast to this, for the 2NCON problem we can observe that the LP relaxations are solved quickly compared to the time needed to solve the corresponding ILP. Hence, many branch-and-bounds nodes are required to solve the instances to optimality. At this point, our algorithm may be improved by identifying further classes of strengthening inequalities and/or improving the branching strategy.

Orientations for further problems. For a special variant of a 2NCON problem where there exists an edge $\{s, t\}$ that has to be contained in every optimal solution, we can use the s, t-orientations in order to derive an even stronger ILP formulation: We can transform such a problem into a problem of finding feasible orientations, w.r.t. two roots s and t, requiring the arc (s, t) to be in the solution, all nodes v of the solution network to have a (s, v)-path, and all nodes $v \in \mathcal{R}_2$ two node-disjoint directed paths (s, v) and (v, t). Furthermore, we can require resulting orientations to be cycle-free by adding additional constraints, thus strengthening the formulation.

Furthermore, our orientation-based approaches can also be applied to other network design problems where feasible solutions are trees, 2-connected, or mixture of both. For example it could be applied to problems with degree-constraints or group-generalizations of Steiner problems (given multiple groups of nodes, pick one representative per group and connect these representatives).

Another promising field where our research can be successively reused is summarized as *connected facility location*. Thereby, we search for a cost-minimal solution where we have to select facilities, assign customers to these facilities, and subsequently interconnect the facilities. The latter interconnection may be required not only to be a tree, but to satisfy further survivability constraints. Arborescence based CFL models have recently been presented by my co-author [LG10, GL11]. Considering certain 2-node-connectivity requirements, we presented and investigated the *2-interconnected facility location* problem (*2iCFL*) both from the complexity theory and the mathematical programming point of view [CKM09]. For the latter we applied our orientation theorems and obtained a practically worthwhile algorithm.

Approximation algorithms. Both the k-cardinality tree and {0,1,2}-survivable network design problems have been subject of extensive research in the field of approximation algorithms. In contrast to other (meta-)heuristic algorithms, such approximations have only been considered in a theoretical setting and, to the best of our knowledge, there are no published implementations or studies that would evaluate their practical performance and applicability. We hope that at some point the research in this interesting field will be taken into practice to experimentally

evaluate its advantages and drawbacks. For the {0,1,2}-SND problem, this thesis suggests TSNDLib, a collection of benchmark instances, that can be used for this purpose.

For both considered problem classes, the best known approximation algorithms are based on undirected ILP formulations that are based on undirected graphs. The best known approximation factor for the related Steiner tree problem was given recently in [BGRS10]. In contrast to the previously known approximations, it uses a directed ILP formulation for the first time. It would be worthwhile to see, whether our orientation-based ILPs can be used to derive better approximation factors for KCT or {0,1,2}-SND problems.

Bibliography

[ABV95] B. Awerbuch, Y. Azar A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. In *Proc. of STOC '95*, pages 277–283. ACM, 1995.

[ACPS93] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. *Journal of the ACM*, 40(5):1134–1164, 1993.

[ADNP99] V. Auletta, Y. Dinitz, Z. Nutov, and D. Parente. A 2-approximation algorithm for finding an optimum 3-vertex-connected spanning subgraph. *Journal of Algorithms*, 32(1):21–30, 1999.

[AK00] S. Arora and G. Karakostas. A $(2 + \epsilon)$-approximation algorithm for the k-MST problem. In *Proc. SODA '00*, pages 754–759. SIAM, 2000.

[Bac05] P. Bachhiesl. The OPT- and the SST-problems for real world access network design – basic definitions and test instances. Working Report NetQuest 01/2005, Carinthia Tech Institute, Klagenfurt, Austria, 2005.

[BB03] C. Blum and M. Blesa. KCTLIB – a library for the edge-weighted k-cardinality tree problem. `http://iridia.ulb.ac.be/~cblum/kctlib/`, 2003.

[BB05a] C. Blum and M. Blesa. Combining ant colony optimization with dynamic programming for solving the k-cardinality tree problem. In *Proc. IWANN'05*, volume 3512 of *LNCS*. Springer, 2005.

[BB05b] C. Blum and M. J. Blesa. New metaheuristic approaches for the edge-weighted k-cardinality tree problem. *Computers & Operations Research*, 32:1355–1377, 2005.

[BBM90] D. Bienstock, E. F. Brickell, and C. L. Monma. On the structure of minimum-weight k-connected spanning networks. *SIAM Journal on Discrete Mathematics*, 3(3):320–329, 1990.

[BCV95] A. Blum, P. Chalasani, and S. Vempala. A constant-factor approximation for the k-MST problem in the plane. In *Proc. STOC '95*, pages 294–302. ACM, 1995.

[BE03] C. Blum and M. Ehrgott. Local search algorithms for the k-cardinality tree problem. *Discrete Applied Mathematics*, 128(2–3):511–540, 2003.

[BGRS10] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. An improved lp-based approximation for steiner tree. In *Proc. STOC '10*, pages 583–592. ACM, 2010.

[BK04] F. Barahona and H. Kerivin. Separation of partition inequalities with terminals. *Discrete Optimization*, 1(2):129–140, 2004.

[BKM08] M. Didi Biha, H. Kerivin, and A. R. Mahjoub. On the polytope of the (1,2)-survivable network design problem. *SIAM Journal on Discrete Mathematics*, 22(4):1640–1666, 2008.

[Blu06] C. Blum. A new hybrid evolutionary algorithm for the huge k-cardinality tree problem. In *Proc. GECCO'06*, pages 515–522. ACM, 2006.

[Blu07] C. Blum. Revisiting dynamic programming for finding optimal subtrees in trees. *European Journal of Operational Research*, 177(1):102–115, 2007.

[BLW87] M. W. Bern, E. L. Lawler, and A. L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *Journal of Algorithms*, 8(2):216–235, 1987.

[BM96] M. Didi Biha and A. R. Mahjoub. k-edge connected polyhedra on series-parallel graphs. *Operations Research Letters*, 19(2):71–78, 1996.

[BM97] M. Baïou and A. R. Mahjoub. Steiner 2-edge connected subgraph polytopes on series-parallel graphs. *SIAM Journal on Discrete Mathematics*, 10(3):505–514, 1997.

[BMM04] A. Balakrishnan, T. L. Magnanti, and P. Mirchandani. Connectivity–splitting models for survivable network design. *Networks*, 43(1):10–27, 2004.

[Bra02] U. Brandes. Eager s,t-ordering. In *Proc. ESA '02*, volume 2461 of *LNCS*, pages 247–256. Springer, 2002.

[BRV96] A. Blum, R. Ravi, and S. Vempala. A constant-factor approximation algorithm for the k-MST problem. In *Proc. STOC '96*, pages 442–448. ACM, 1996.

[BS04] T. N. Bui and G. Sundarraj. Ant system for the k-cardinality tree problem. In *Proc. GECCO '04*, volume 3102 of *LNCS*, pages 36–47. Springer, 2004.

[BUM06] J. Brimberg, D. Urošević, and N. Mladenović. Variable neighborhood search for the vertex weighted k-cardinality tree problem. *European Journal of Operational Research*, 171(1):74–84, 2006.

[BX00] M. J. Blesa and F. Xhafa. A C++ Implementation of Tabu Search for k-Cardinality Tree Problem based on Generic Programming and Component Reuse. In c/o tranSIT GmbH, editor, *Net.ObjectDsays 2000 Tagungsband*, pages 648–652, Erfurt, Germany, 2000. Net.ObjectDays-Forum.

[CA95] L. W. Clarke and G. Anandalingam. A bootstrap heuristic for designing minimum cost survivable networks. *Computers & Operations Research*, 22(9):921–934, 1995.

[CCK08] T. Chakraborty, J. Chuzhoy, and S. Khanna. Network design for vertex connectivity. In *Proc. STOC '08*, pages 167–176. ACM, 2008.

[CF70] W. Chou and H. Frank. Survivable communication networks and the terminal capacity matrix. *IEEE Transactions on Circuit Theory*, CT-17:183–192, 1970.

[CG97] B. V. Cherkassky and A. V. Goldberg. On implementing push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.

[Chi08] M. Chimani. *Computing Crossing Numbers*. PhD thesis, TU Dortmund, 2008.

[Cho92] S. Chopra. Polyhedra of the equivalent subgraph problem and some edge connectivity problems. *SIAM Journal on Discrete Mathematics*, 5(3):321–337, 1992.

[CK94] S. Y. Cheung and A. Kumar. Efficient quorumcast routing algorithms. In *Proc. INFOCOM '94*, pages 840–847. IEEE Society Press, 1994.

[CK09] J. Chuzhoy and S. Khanna. An $o(k3logn)$-approximation algorithm for vertex-connectivity survivable network design. In *Proc. FOCS '09*, pages 437–441. IEEE Computer Society, 2009.

[CKLM08a] M. Chimani, M. Kandyba, I. Ljubić, and P. Mutzel. Obtaining optimal k-cardinality trees fast. In *Proc. ALENEX'08*, pages 27–36. SIAM, 2008.

[CKLM08b] M. Chimani, M. Kandyba, I. Ljubić, and P. Mutzel. Strong formulations for 2-node-connected Steiner network problems. In *Proc. COCOA '08)*, volume 5165 of *LNCS*, pages 190–200. Springer, 2008.

[CKLM09] M. Chimani, M. Kandyba, I. Ljubić, and P. Mutzel. Obtaining optimal k-cardinality trees fast. *Journal of Experimental Algorithmics*, 14:2.5–2.23, 2009.

[CKLM10] M. Chimani, M. Kandyba, I. Ljubić, and P. Mutzel. Orientation-based models for 0,1,2-survivable network design: theory and practice. *Mathematical Programming*, 124:413–439, 2010.

[CKM07] M. Chimani, M. Kandyba, and P. Mutzel. A new ILP formulation for 2-root-connected prize-collecting Steiner networks. In *Proc. ESA '07*, volume 4698 of *LNCS*, pages 681–692. Springer, 2007.

[CKM09] M. Chimani, M. Kandyba, and M. Martens. 2-interconnected facility location: Specifications, complexity results, and exact solutions. Technical report TR09–1–008, Chair XI Algorithm Engineering, TU Dortmund, 2009. submitted to journal.

[CKP07] M. Chimani, M. Kandyba, and M. Preuss. Hybrid numerical optimization for combinatorial network problems. In *Proc. HM '07*, volume 4771 of *LNCS*, pages 185–200. Springer, 2007.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[CRR03] H. Cancela, F. Robledo, and G. Rubino. Network design with node connectivity constraints. In *Proc. LANC '03*, pages 13–20. ACM, 2003.

[CRRW93] C. R. Coullard, A. Rais, R. L. Rardin, and D. K. Wagner. Linear-time algorithm for the 2-connected Steiner subgraph problem on special classes of graphs. *Networks*, 23(3):195–206, 1993.

[CRW01] F. A. Chudak, T. Roughgarden, and D. P. Williamson. Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation. In *Proc. IPCO '01*, volume 2081 of *LNCS*, pages 60–70. Springer, 2001.

[CS89] G-R. Cai and Y-G. Sun. The minimum augmentation of any graph to a k-edge-connected graph. *Networks*, 19:151–172, 1989.

[CV07] J. Cheriyan and A. Vetta. Approximation algorithms for network design with metric costs. *SIAM Journal on Discrete Mathematics*, 21(3):612–636, 2007.

[CVV02] J. Cheriyan, S. Vempala, and A. Vetta. Approximation algorithms for minimum-cost k-vertex connected subgraphs. In *Proc. STOC '02*, pages 306–312. ACM, 2002.

[dFdMR95] H. de Fraysseix, P. Ossona de Mendez, and P. Rosenstiehl. Bipolar orientations revisited. *Discrete Applied Mathematics*, 56(2–3):157–179, 1995.

[DN99] Y. Dinitz and Z. Nutov. A 3-approximation algorithm for finding optimum 4,5-vertex connected spanning subgraphs. *Journal of Algorithms*, 32(1):31–40, 1999.

[Ebe83] J. Ebert. *st*-ordering the vertices of biconnected graphs. *Computing*, 30(1):19–33, 1983.

[EF96] M. Ehrgott and J. Freitag. K_TREE/K_SUBGRAPH: a program package for minimal weighted k-cardinality tree subgraph problem. *European Journal of Operational Research*, 1(93):214–225, 1996.

[EFHM97] M. Ehrgott, J. Freitag, H.W. Hamacher, and F. Maffioli. Heuristics for the k-cardinality tree and subgraph problem. *Asia Pacific Journal of Operational Research*, 14(1):87–114, 1997.

[EFS56] P. Elias, A. Feinstein, and C. E. Shannon. A note on the maximal flow through a network. *IRE Transactions on Information Theory*, IT 2:117–119, 1956.

[Epp97] D. Eppstein. Faster geometric k-point MST approximation. *Computational Geometry: Theory and Applications*, 8(5):231–240, 1997.

[ET76a] K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976.

[ET76b] S. Even and R. E. Tarjan. Computing an *st*-Numbering. *Theoretical Computer Science*, 2(3):339–344, 1976.

[ET77] S. Even and R. E. Tarjan. Corrigendum: Computing an *st*-numbering. TCS 2(1976):339-344. *Theoretical Computer Science*, 4(1):123, 1977.

[FF56] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[FH92] L. R. Foulds and H. W. Hamacher. A new integer programming approach to (restricted) facilities layout problems allowing flexible facility shapes. Technical report 1992-3, University of Waikato, Department of Management Sience, 1992.

[FHJM94] M. Fischetti, W. Hamacher, K. Jornsten, and F. Maffioli. Weighted k-cardinality trees: Complexity and polyhedral structure. *Networks*, 24:11–21, 1994.

[FJ82] G. N. Frederickson and J. JáJá. On the relationship between the biconnectivity augmentation and traveling salesman problems. *Theoretical Computer Science*, 19:189–201, 1982.

[FJW06] L. Fleischer, K. Jain, and D. P. Williamson. Iterative rounding 2-approximation algorithms for minimum-cost vertex connectivity problems. *Journal of Computer and System Sciences*, 72(5):838–867, 2006.

[FL08] J. Fakcharoenphol and B. Laekhanukit. An $O(\log^2 k)$-approximation algorithm for the k-vertex connected spanning subgraph problem. In *Proc. STOC '08*, pages 153–158. ACM, 2008.

[Fra92] A. Frank. Augmenting graphs to meet edge-connectivity requirements. *SIAM Journal on Discrete Mathematics*, 5(1):25–53, 1992.

[Gal57] D. Gale. A theorem on flows in networks. *Pacific Journal of Mathematics*, 7:1073–1082, 1957.

[Gar96] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proc. FOCS'96*, pages 302–309. IEEE Computer Society, 1996.

[Gar05] N. Garg. Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In *Proc. STOC'05*, pages 396–402. ACM, 2005.

[GGP+94] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, É. Tardos, and D. P. Williamson. Improved approximation algorithms for network design problems. In *Proc. SODA '94*, pages 223–232. SIAM, 1994.

[GH97] N. Garg and D. S. Hochbaum. An $O(\log k)$-approximation algorithm for the k minimum spanning tree problem in the plane. *Algorithmica*, 18(1):111–121, 1997.

[GL11] S. Gollowitzer and I. Ljubic. MIP models for connected facility location: A theoretical and computational study. *Computers & Operations Research*, 38(2):435–449, 2011.

[GM90] M. Grötschel and C. L. Monma. Integer polyhedra arising from certain network design problems with connectivity constraints. *SIAM Journal on Discrete Mathematics*, 3(4):502–523, 1990.

[GM93] M. X. Goemans and Y. Myung. A catalog of Steiner tree formulations. *Networks*, 23:19–28, 1993.

[GMS91] M. Grötschel, C. L. Monma, and M. Stoer. Polyhedral Approaches to Network Survivability. In *Reliability of Computer and Communication Networks, Proc. Workshop 1989*, volume 5 of *Discrete Mathematics and Theoretical Computer Science*, pages 121–141, 1991.

[GMS92a] M. Grötschel, C. L. Monma, and M. Stoer. Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. *Operatios Research*, 40(2):309–330, 1992.

[GMS92b] M. Grötschel, C. L. Monma, and M. Stoer. Facets for polyhedra arising in the design of communication networks with low-connectivity constraints. *SIAM Journal on Optimization*, 2(3):474–504, 1992.

[GR02] E. Ghashghai and R. L. Rardin. Using a hybrid of exact and genetic algorithms to design survivable networks. *Computers & Operations Research*, 29(1):53–66, 2002.

[GW95] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[Har62] F. Harary. The maximum connectivity of a graph. *Proc. of the National Academy of Sciences*, 48:1142–1146, 1962.

[Har69] F. Harary. *Graph Theory.* Addison-Wesley, 1969.

[HJM91] H. W. Hamacher, K. Joernsten, and F. Maffioli. Weighted k-cardinality trees. Technical report 91.023, Politecnico di Milano, Dipartimento di Elettronica, 1991.

[HKKN10] M. Hajiaghayi, R. Khandekar, G. Kortsarz, and Z. Nutov. Prize-collecting steiner network problems. In *Proc. IPCO'10*, volume 6080 of *LNCS*, pages 71–84. Springer, 2010.

[Jai98] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21:39–60, 1998.

[JL97] K. Joernsten and A. Lokketangen. Tabu search for weighted k-cardinality trees. *Asia Pacific Journal of Operational Research*, 14:9–26, 1997.

[JMP00] D. S. Johnson, M. Minkoff, and S. Phillips. The prize-collecting Steiner tree problem: Theory and practice. In *Proc. SODA '00*, pages 760–769. SIAM, 2000.

[Khu97] S. Khuller. Approximation algorithms for finding highly connected subgraphs. In D. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 236–265. 1997.

[KKL04] G. Kortsarz, R. Krauthgamer, and J. R. Lee. Hardness of approximation for vertex-connectivity network design problems. *SIAM Journal on Computing*, 33(3):704–720, 2004.

[KM89] C.-W. Ko and C. L. Monma. Heuristic methods for designing highly survivable communication networks. Technical report, Bellcore, Morristown, 1989.

[KM98] T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207–232, 1998.

[KM02] H. Kerivin and A. R. Mahjoub. Separation of partition inequalities for the (1,2)-survivable network design problem. *Operations Research Letters*, 30:265–268, 2002.

[KM05a] H. Kerivin and A. R. Mahjoub. Design of survivable networks: A survey. *Networks*, 46(1):1–21, 2005.

[KM05b] H. Kerivin and A. R. Mahjoub. On survivable network polyhedra. *Discrete Mathematics*, 290(2–3):183–210, 2005.

[KMN04] H. Kerivin, A. R. Mahjoub, and C. Nocq. (1,2)-survivable networks: Facets and branch-and-cut. In M. Grötschel, editor, *The Sharpest Cut, MPS-SIAM Series on Optimization*, pages 121–152. 2004.

[KMV03] T. Koch, A. Martin, and S. Voß. SteinLib – an updated library on Steiner tree problems in graphs. `http://elib.zib.de/steinlib`, 2003.

[KN03] G. Kortsarz and Z. Nutov. Approximating node connectivity problems via set covers. *Algorithmica*, 37(2):75–92, 2003.

[KN05] G. Kortsarz and Z. Nutov. Approximating k-node connected subgraphs via critical graphs. *SIAM Journal on Computing*, 35(1):247–257, 2005.

[KN07] G. Kortsarz and Z. Nutov. Approximating minimum cost connectivity problems. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*. 2007.

[KR96] S. Khuller and B. Raghavachari. Improved approximation algorithms for uniform connectivity problems. *Journal of Algorithms*, 21(2):434–450, 1996.

[KV94] S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 41(2):214–235, 1994.

[LEC67] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In P. Rosenstiehl, editor, *Theory of Graphs*, pages 215–232. 1967.

[LG10] I. Ljubic and S. Gollowitzer. Layered graph approaches to the hop constrained connected facility location problem. Technical Report 2010-08, University of Vienna, 2010.

[Lju04] I. Ljubić. *Exact and Memetic Algorithms for Two Network Design Problems*. PhD thesis, TU Vienna, 2004.

[LN09] Y. Lando and Z. Nutov. Inapproximability of survivable networks. *Theoretical Computer Science*, 410(21–23):2122–2125, 2009.

[LR03] A. Lucena and M. G. C. Resende. Strong lower bounds for the prize-collecting Steiner problem in graphs. *Discrete Applied Mathematics*, 141(1-3):277–294, 2003.

[LR08] M. Leitner and G. Raidl. Lagrangian decomposition, metaheuristics, and hybrid approaches for the design of the last mile in fiber optic networks. In *Proc. HM '08*, volume 5296 of *LNCS*, pages 158–174. Springer, 2008.

[LR10] M. Leitner and G. Raidl. Strong lower bounds for a survivable network design problem. In *Proc. ISCO '10*, volume 36 of *Electronic Notes in Discrete Mathematics*, pages 295–302, 2010.

[LRP09] M. Leitner, G. Raidl, and U. Pferschy. Accelerating column generation for a survivable network design problem. In *Proc. INOC '09*, 2009.

[LWP+06] I. Ljubić, R. Weiskircher, U. Pferschy, G. Klau, P. Mutzel, and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Mathematical Programming*, 105(2–3):427–449, 2006.

[Maf91] F. Maffioli. Finding a best subtree of a tree. Technical report 91.041, Politecnico di Milano, Dipartimento di Elettronica, 1991.

[MBCV99] J. S. B. Mitchell, A. Blum, P. Chalasani, and S. Vempala. A constant-factor approximation algorithm for the geometric k-MST problem in the plane. *SIAM Journal on Computing*, 28(3):771–781, 1999.

[Meh88] K. Mehlhorn. A faster approximation for the Steiner problem in graphs. *Information Processing Letters*, 27:125–128, 1988.

[Men27] K. Menger. Zur allgemeinen Kurventheorie. *Fundamenta Mathematicae*, 10:96–115, 1927.

[Mit99] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999.

[MMP90] C. L. Monma, B. S. Munson, and W. R. Pulleyblank. Minimum-weight two-connected spanning networks. *Mathematical Programming*, 46(2):153–171, 1990.

[MR05] T. L. Magnanti and S. Raghavan. Strong formulations for network design problems with connectivity requirements. *Networks*, 45(2):61–79, 2005.

[MS89] C. L. Monma and D. F. Shallcross. Methods for designing communications networks with certain two-connected survivability constraints. *Operations Research*, 37(4):531–541, 1989.

[NGM90] D. Naor, D. Gusfield, and Ch. Martel. A fast algorithm for optimally increasing the edge-connectivity. In *Proc. FOCS '90*, pages 698–707. IEEE Computer Society, 1990.

[Nut09a] Z. Nutov. An almost $O(\log k)$-approximation for k-connected subgraphs. In *Proc. SODA '09*, pages 912–921. SIAM, 2009.

[Nut09b] Z. Nutov. Approximating minimum cost connectivity problems via uncrossable bifamilies and spider-cover decompositions. In *Proc. FOCS '09*, pages 417–426. IEEE Computer Society, 2009.

[Nut09c] Z. Nutov. Approximating node-connectivity augmentation problems. In *Proc. APPROX '09 / RANDOM '09*, volume 5687 of *LNCS*, pages 286–297. Springer, 2009.

[Nut09d] Z. Nutov. A note on rooted survivable networks. *Information Processing Letters*, 109(19):1114–1119, 2009.

[NW99] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Discrete Mathematics and Optimization. Wiley-Interscience, 1999.

[OPTW07] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design Library. In *Proc. INOC '07*, 2007. http://sndlib.zib.de.

[PD01a] T. Polzin and S. V. Daneshmand. A comparison of steiner tree relaxations. *Discrete Applied Mathematics*, 112(1-3):241–261, 2001.

[PD01b] T. Polzin and S. V. Daneshmand. A comparison of steiner tree relaxations. *Discrete Applied Mathematics*, 112:241–261, 2001.

[PD01c] T. Polzin and S. V. Daneshmand. Improved algorithms for the Steiner problem in networks. *Discrete Applied Mathematics*, 112(1-3):263–300, 2001.

[PLZC07] S. Peng, M. Li, S. Zhang, and T. C. E. Cheng. Some new structural properties of shortest 2-connected Steiner networks. In *Proc. FAW '07*, volume 4613 of *LNCS*, pages 317–324. Springer, 2007.

[PR91] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991.

[PSK97] M. Penn and H. Shasha-Krupnik. Improved approximation algorithms for weighted 2-and 3-vertex connectivity augmentation problems. *Journal of Algorithms*, 22(1):187–196, 1997.

[PW97] H.W. Philpott and N. Wormald. On the optimal extraction of ore from an open-cast mine. Technical report, University of Auckland, New Zealand, 1997.

[QCM08] F. P. Quintão, A. S. Cunha, and G. R. Mateus. Integer programming formulations for the k-cardinality tree problem. *Electronic Notes in Discrete Mathematics*, 30:225–230, 2008.

[QCML10] F. P. Quintão, A. S. Cunha, G. R. Mateus, and A. Lucena. The k-cardinality tree problem: Reformulations and lagrangian relaxation. *Discrete Applied Mathematics*, 158(12):1305–1314, 2010.

[Rag95] S. Raghavan. *Formulations and Algorithms for the Network Design Problems with Connectivity Requirements*. PhD thesis, MIT, Cambridge, MA, 1995.

[Rag04] S. Raghavan. Low-connectivity network design on series-parallel graphs. *Networks*, 43(3):163–176, 2004.

[RdAR+01] I. Rossetti, M. P. de Aragão, C.C. Ribeiro, E. Uchoa, and R. F. Werneck. New benchmark instances for the Steiner problem in graphs. In *Extended Abstracts of the 4th Metaheuristics International Conference*, pages 557–561, 2001.

[RG77] A. Rosenthal and A. Goldner. Smallest augmentation to biconnect a graph. *SIAM Journal on Computing*, 6:55–66, 1977.

[Rob39] H.E. Robbins. A theorem on graphs with an application to a problem of traffic control. *American Mathematical Monthly*, 46:281–283, 1939.

[RSM+96] R. Ravi, R. Sundaram, M. V. Marathe, D. J. Rosenkrantz, and S. S. Ravi. Spanning trees – short or small. *SIAM Journal of Discrete Mathematics*, 9:128–200, 1996.

[RW97] R. Ravi and D. P. Williamson. An approximation algorithm for minimum-cost vertex-connectivity problems. *Algorithmica*, 18(1):21–43, 1997.

[RW02] R. Ravi and D. P. Williamson. Erratum: An approximation algorithm for minimum-cost vertex-connectivity problems. *Algorithmica*, 34(1):98–107, 2002.

[Sch98] A. Schrijver. *Theory of Linear and Integer Programming.* Discrete Mathematics and Optimization. Wiley-Interscience, 1998.

[Seg87] A. Segev. The node-weighted Steiner tree problem. *Networks*, 17(1):1–17, 1987.

[Spe08] Spec. Standard performance evaluation corporation. `http://www.spec.org/`, 2008.

[ST84] J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14:325–336, 1984.

[Sto92] M. Stoer. *Design of Survivable Networks*, volume 1531 of *Lecture Notes in Mathematics.* Springer, 1992.

[Suu74] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4:125–145, 1974.

[SWK69] K. Steiglitz, P. Weigner, and D. J. Kleitman. The design of minimum-cost survivable networks. *IEEE Transactions on Circuit Theory*, 16:455–460, 1969.

[TSN08] TSNDLib: Collection of benchmark instances for Topological {0,1,2}-Survivable Network Design problems, 2008. `http://ls11-www.cs.tu-dortmund.de/TSNDLib/`.

[TSP] TSPLIB. `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`.

[UBM04] D. Urošević, J. Brimberg, and N. Mladenović. Variable neighborhood decomposition search for the edge weighted k-cardinality tree problem. *Computers & Operations Research*, 31(8):1205–1213, 2004.

[UKW88] S. Ueno, Y. Kajitani, and H. Wada. Minimum augmentation of a tree to a k-edge-connected graph. *Networks*, 18:19–25, 1988.

[Wag07] D. Wagner. Generierung und Adaptierung von Testinstanzen für das OPT und SST Problem. Technical Report 03/2007, Carinthia Tech Institute, Klagenfurt, Austria, 2007. In german.

[WGMV95] D. P. Williamson, M. X. Goemans, M. Mihail, and V. V. Vazirani. A primal-dual approximation algorithm for generalized Steiner network problems. *Combinatorica*, 15(3):435–454, 1995.

[Win85] P. Winter. Generalized Steiner problem in outerplanar networks. *BIT*, 25(3):485–496, 1985.

[Win86] P. Winter. Generalized Steiner problem in series-parallel networks. *Journal of Algorithms*, 7(4):549–566, 1986.

[Win87a] P. Winter. Steiner problem in Halin networks. *Discrete Applied Mathematics*, 17(3):281–294, 1987.

[Win87b] P. Winter. Steiner problem in networks: A survey. *Networks*, 17(2):129–167, 1987.

[WN87] T. Watanabe and A. Nakamura. Edge-connectivity augmentation problems. *Computer and System Sciences*, 35:96–144, 1987.

[Woe92] G. J. Woeginger. Computing maximum valued regions. *Acta Cybernetica*, 10(4):303–316, 1992.

[Wol98] L. A. Wolsey. *Integer Programming*. Discrete Mathematics and Optimization. Wiley-Interscience, 1998.

[WRP+06] D. Wagner, G. R. Raidl, U. Pferschy, P. Mutzel, and P. Bachhiesl. A multi-commodity flow approach for the design of the last mile in real-world fiber optic networks. In *Proc. OR '06*, pages 197–202. Springer, 2006.

[WRP+07] D. Wagner, G. R. Raidl, U. Pferschy, P. Mutzel, and P. Bachhiesl. A directed cut for the design of the last mile in real-world fiber optic networks. In *Proc. INOC '07*, 2007.

[WZ05] P. Winter and M. Zachariasen. Two-connected Steiner networks: structural properties. *Operations Research Letters*, 33(4):395–402, 2005.

[ZL93] A. Zelikovsky and D. Lozovanu. Minimal and bounded trees. In *Tezele Cong. XVIII Acad. Romano-Americane, Kishinev*, pages 25–26, 1993.

Index