Robust Design of Single-Commodity Networks

Inaugural-Dissertation
zur
Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Universität zu Köln

vorgelegt von
Daniel Rainer Schmidt geb. Plümpe
aus Unna

Köln 2014

Berichterstatter:	Prof. Dr. Michael Jünger Prof. Dr. Frauke Liers
Tag der mündlichen Prüfung:	1. Dezember 2014

Zusammenfassung

Die vorliegende Dissertation ist im Rahmen einer Kooperation mit Valentina Cacchiani, Frauke Liers, Andrea Lodi und Michael Jünger entstanden, an der in Teilen auch Eduardo Álvarez-Miranda, Tim Dorneth und Tiziano Parriani beteiligt waren.

Das Thema der Arbeit ist ein robustes Netzwerkdesignproblem, d.h. ein Problem der Art „*dimensioniere ein Netzwerk so, dass in allen realistischen Szenarien ausreichend Kapazität vorhanden ist.*“ In dem konkreten Problem wird das Netzwerk durch einen ungerichteten Graphen modelliert, in dem jedes Szenario einen Angebots- oder Bedarfswert für jeden Knoten definiert. Ein Fluss im Netzwerk heißt zulässig, wenn er alle Angebote und Bedarfe ausgleicht. Welche Szenarien beachtet werden müssen, bestimmt das Szenarienpolytop $\mathfrak{B}$. Gesucht sind *ganzzahlige* Kapazitäten, die zwar die insgesamten Installationskosten minimieren, aber in jedem Szenario $b \in \mathfrak{B}$ die Existenz eines zulässigen Flusses garantieren. Dieses *Single-Commodity Robust Network Design Problem* (`sRND`) geht zurück auf eine Arbeit von Buchheim, Liers und Sanità (INOC 2011) und enthält als Spezialfall das *Steinerbaumproblem* (gegeben einen ungerichteten Graphen und eine Terminalmenge, finde einen Teilbaum mit minimalen Kosten, der alle Terminale verbindet). Es ist daher $\mathcal{NP}$-schwierig. Außerdem ist es eine natürliche Erweiterung von Minimumkostenflüssen.

Die Literatur unterscheidet die Fälle, dass $\mathfrak{B}$ als endliche Menge seiner Extremalpunkte (endlicher Fall) oder als Lösungsmenge eines linearen Ungleichungssystems gegeben ist (polyedrischer Fall). Diese beiden Fälle sind zwar äquivalent, können jedoch im Allgemeinen nicht effizient ineinander überführt werden.

Buchheim, Liers und Sanità (INOC 2011) schlagen einen Branch-and-Cut-Algorithmus für den endlichen Fall vor. In diesem Fall existiert eine kanonische Formulierung des Problems als gemischt-ganzzahliges lineares Programm (MIP). Sie enthält Flussvariablen für jedes Szenario. Um bessere Schranken zu erhalten, fügen Buchheim, Liers und Sanità zusätzlich allgemeine Schnittebenen, sog. Target Cuts, ein.

Der erste Teil der Dissertation beschäftigt sich nun mit der Variante des Problems, in der jedes Szenario genau zwei Terminalknoten besitzt. Ist das Netzwerk zusätzlich ein ungewichteter vollständiger Graph, so handelt es sich um das *Network Synthesis Problem* im Sinne von Chien (IBM Journal of R&D, 1960). Für diesen Fall existieren polynomielle Algorithmen von Gomory und Hu (SIAM J. of Appl. Math. 1961) und Kabadi, Yan, Du und Nair (SIAM J. on Discr. Math. 2009), die jedoch darauf beruhen, dass vollständige Graphen hamiltonsch sind. Wir entwickeln unter der Voraussetzung, dass die Angebote und Bedarfe einer bestimmten Verteilung folgen, einen ähnlicher Algorithmus für Hyperwürfelgraphen. Diese Graphen sind ebenfalls hamiltonsch.

Im zweiten Teil der Arbeit geht es um die Struktur des `sRND`-Lösungspolyeders. Hier ist das erste Ergebnis eine neue MIP-Kapazitätsformulierung des `sRND`-Problems, deren Größe unabhängig von der Anzahl der Extremalpunkte von $\mathfrak{B}$ ist. Damit ist die Formulierung auch für den polyedrischen Fall geeignet. Die Formulierung verwendet sogenannte Cut-Set-Ungleichungen, die in ähnlicher Form auch bei anderen Netzwerkdesignproblemen

verwendet werden. Wir wandeln einen Beweis von Mattia (Computational Optimization and Applications 2013) ab und so zeigen so, dass die Cut-Set-Ungleichungen Facetten des Lösungpolyeders sind. Um eine bessere Relaxierung des ganzzahligen linearen Programms zu erreichen, interpretieren wir bestimmte generische Schnittebenen als sogenannte 3-Partitionsungleichungen und zeigen, dass auch diese Ungleichungen Facetten des Lösungspolyeders definieren.

Da die Kapazitätsformulierung exponentielle Größe hat, benötigen wir einen Separierungsalgorithmus für Cut-Set-Ungleichungen. Im endlichen Fall führen wir das Separierungsproblem auf ein Minimum-Schnitt-Problem zurück, das in Polynomialzeit gelöst werden kann. Im polyedrischen Fall ist das Separierungsproblem hingegen $\mathcal{NP}$-schwierig, selbst wenn man davon ausgeht, dass das Szenarienpolytop im Wesentlichen ein Quader – das sogenannte Hose-Polytop – ist. Dennoch kann das Separierungsproblem in der Praxis gelöst werden: Wir zeigen hierzu einen MIP basierten Separierungsansatz für Hose-Szenarienpolytope. Weiterhin stellt die Arbeit zwei Separierungsalgorithmen für 3-Partitionsungleichungen vor, die unabhängig von der Codierung des Szenarienpolytops funktionieren. Zusätzlich führen wir einige Rundungsheuristiken ein.

Das Ergebnis ist ein Branch-and-Cut-Algorithmus für die Kapazitätsformulierung, den wir im letzten Teil der Dissertation untersuchen. Dort weisen wir experimentell die praktische Leistungsfähigkeit des Branch-and-Cut-Algorithmus für den endlichen und den polyedrischen Fall nach. Als Referenzpunkt dient dazu eine `CPLEX`-Implementierung der Flussformulierung und die Rechenergebnisse von Buchheim, Liers und Sanità. Die Experimente zeigen, dass unser neuer Branch-and-Cut-Algorithmus eine Verbesserung des bisherigen Ansatzes darstellt. Dabei eignet er sich vorallem für Probleminstanzen mit vielen Szenarien. Es zeigt sich insbesondere, dass die MIP-Separierung der Schnittungleichungen im polyedrischen Fall auch praktisch anwendbar ist.

Abstract

The results in the present work were obtained in a collaboration with Eduardo Álvarez-Miranda, Valentina Cacchiani, Tim Dorneth, Michael Jünger, Frauke Liers, Andrea Lodi and Tiziano Parriani.

The subject of this thesis is a robust network design problem, i.e., a problem of the type "dimension a network such that it has sufficient capacity in all likely scenarios." In our case, we model the network with an undirected graph in which each scenario defines a supply or demand for each node. We say that a flow in the network is feasible for a scenario if it can balance out its supplies and demands. A scenario polytope $\mathfrak{B}$ defines which scenarios are relevant. The task is now to find integer capacities that minimize the total installation costs while allowing for a feasible flow in each scenario. This problem is called *Single-Commodity Robust Network Design Problem* (`sRND`) and was introduced by Buchheim, Liers and Sanità (INOC 2011). The problem contains the Steiner Tree Problem (given an undirected graph and a terminal set, find a minimum cost subtree that connects all terminals) and therefore is $\mathcal{NP}$-hard. The problem is also a natural extension of minimum cost flows.

The network design literature treats the case that the scenario polytope $\mathfrak{B}$ is given as the finite set of its extreme points (finite case) and that it is given as the feasible region of finitely many linear inequalities (polyhedral case). Both descriptions are equivalent, however, an efficient transformation is not possible in general.

Buchheim, Liers and Sanità (INOC 2011) propose a Branch-and-Cut algorithm for the finite case. In this case, there exists a canonical problem formulation as a mixed integer linear program (MIP). It contains a set of flow variables for every scenario. Buchheim, Liers and Sanità enhance the formulation with general cutting planes that are called target cuts.

The first part of the dissertation considers the problem variant where every scenario has exactly two terminal nodes. If the underlying network is a complete, unweighted graph, then this problem is the Network Synthesis Problem as defined by Chien (IBM Journal of R&D 1960). There exist polynomial time algorithms by Gomory and Hu (SIAM J. of Appl. Math 1961) and by Kabadi, Yan, Du and Nair (SIAM J. on Discr. Math.) for this special case. However, these algorithms are based on the fact that complete graphs are Hamiltonian. The result of this part is a similar algorithm for hypercube graphs that assumes a special distribution of the supplies and demands. These graphs are also Hamiltonian.

The second part of the thesis discusses the structure of the polyhedron of feasible `sRND` solutions. Here, the first result is a new MIP-based capacity formulation for the `sRND` problem. The size of this formulation is independent of the number of extreme points of $\mathfrak{B}$ and therefore, it is also suited for the polyhedral case. The formulation uses so-called cut-set inequalities that are known in similar form from other network design problems. By adapting a proof by Mattia (Computational Optimization and Applications 2013), we show that cut-set inequalities induce facets of the `sRND` polyhedron. To obtain a better linear programming relaxation of the capacity formulation, we interpret certain general mixed integer cuts as 3-partition inequalities and show that these inequalities induce facets as well.

The capacity formulation has exponential size and we therefore need a separation algorithm for cut-set inequalities. In the finite case, we reduce the cut-set separation problem to a minimum cut problem that can be solved in polynomial time. In the polyhedral case, however, the separation problem is $\mathcal{NP}$-hard, even if we assume that the scenario polytope is basically a cube. Such a scenario polytope is called Hose polytope. Nonetheless, we can solve the separation problem in practice: We show a MIP based separation procedure for the Hose scenario polytope. Additionally, the thesis presents two separation methods for 3-partition inequalities. These methods are independent of the encoding of the scenario polytope. Additionally, we present several rounding heuristics.

The result is a Branch-and-Cut algorithm for the capacity formulation. We analyze the algorithm in the last part of the thesis. There, we show experimentally that the algorithm works in practice, both in the finite and in the polyhedral case. As a reference point, we use a `CPLEX` implementation of the flow based formulation and the computational results by Buchheim, Liers and Sanità. Our experiments show that the new Branch-and-Cut algorithm is an improvement over the existing approach. Here, the algorithm excels on problem instances with many scenarios. In particular, we can show that the MIP separation of the cut-set inequalities is practical.

Acknowledgments

My thanks go to all those who have supported me in writing this thesis. I am very grateful to Prof. Dr. Michael Jünger for having me in his Cologne group and I could hardly imagine a more ideal place for working on a PhD. During my time in Cologne, I have had much freedom, but also a place to find advice and support whenever I needed it. You gave me the opportunity to travel with my ideas and to discuss with fellow researchers. Thank you for your trust in me. I also thank Prof. Dr. Frauke Liers for her support, the pleasant collaboration, for many discussions, for listening and for sharing ideas. I always enjoyed my visits to Erlangen. I thank Prof. Andrea Lodi for taking time for our discussions and for asking the right questions. Thank you also for always welcoming me in Bologna and for all the support that you gave me. Large parts of this thesis were developed in a joint project in the VIGONI program funded by the German-Italian University Centre and the German Academic Exchange Service whose financial support I gratefully acknowledge. I thank Prof. Dr. Michael Jünger, Prof. Dr. Frauke Liers and Prof. Andrea Lodi for including me in this great project.

I thank my friends and colleagues Frank Baumann, Michael Belling, Diana Fanghänel, Dustin Feld, Martin Gronemann, Thomas Lange, Sven Mallach, Francesco Mambelli, Gregor Pardella, Andreas Schmutzer, Christiane Spisla, Lena Tepaße, Göntje Teuchert and Käte Zimmer for the harmonic working atmosphere in Cologne, for many discussions, enjoyable coffee breaks and many joint travels to conferences and workshops. In particular, I thank three of you who took many of the every-day troubles from me: Michael for the excellent library service, Thomas by maintaining a most reliable and efficient computer system and Göntje by helping with all the organizational and administrative woes. I thank Christiane for being such a kind and pleasant-natured office-mate and for bringing a lot of fun into our shared office. I trust you enjoyed our research discussions and chats as much as I did.

I also thank my friends from Bologna: Eduardo Álvarez-Miranda, Valentina Cacchiani and Tiziano Parriani made me feel very welcome in Italy. I thank you for a successful project and an enjoyable collaboration. In particular, I thank Valentina for countless kind emails, good suggestions and ideas – but also for showing me Bologna and the Italian cuisine.

To Martin Groß, Jan-Philipp Kappmeier and Melanie Schmidt I am grateful for many fun days of research on network flows. I thank Prof. Dr. Martin Skutella and his group for welcoming us in Berlin for this collaboration.

I thank Jan-Philipp Kappmeier, Magdalena Schmidt, Melanie Schmidt, Christiane Spisla and Lena Tepaße for proof-reading parts of this thesis and giving invaluable feedback. Special thanks go to Melanie who managed to read the entire thesis while writing her own.

However, I could not have written this thesis without the love and support of my family. I thank all of you, in particular my parents and my brother Sven, for being there, but also my parents-in-law and my *belles-sœurs* Katharina and Magdalena for all their kindness. To my wife Melanie I am grateful for more reasons than I could possibly express here.

Contents

Introduction

Much of the present day infrastructure for transportation, supply or telecommunication is organized in networks that connect a set of nodes via links of limited capacity. Before such a network is built or expanded, plans need to be made: Keeping in mind the traffic that will later flow through the network, we must decide where to build its links and what capacities they should have. To keep the access to the network affordable, its costs should be as small as possible. The resulting optimization problem is known as *network design*.

The interest in the design of networks started in the 1950's and 1960's. By now, entire research communities and journals are devoted to its study; their topics ranging from abstract studies of base problems to practical works that incorporate the physical and technical properties of state-of-the-art networks.

The difficulty in designing a network that is both reliable and affordable is the fact that the traffic through a typical network is constantly changing. The fluctuations not only concern the raw amount of traffic in the network, but also the areas of the network that are in use at a given point in time. Consider for instance the traffic in a road network of a big city: In the morning, mostly those streets that lead into the city will be used. Then, during the day, traffic concentrates on the city center until in the afternoon, those streets that lead out of the city are filled with commuters. Thus, when designing a network, different usage situations – we call them scenarios – must be taken into account.

However, classical optimization only works if the problem parameters are known exactly and in particular, if there is only one usage scenario. In order to still use these classical methods, we could optimize for the *average* traffic, risking that the capacities will be insufficient to handle traffic peaks. Unfortunately, most networks break down when the traffic flow exceeds their capacity (think of a traffic jam or a black-out) and therefore, this strategy is unsafe. Alternatively, we can build a network that can handle all the peaks at the same time. While this strategy guarantees that the network will not be used beyond its capacity, it is also very expensive. As an example, consider again the road network from the previous paragraph: If the peak traffic during rush hour requires a four lane road, then building four lanes per direction is certainly a feasible solution. However, by realizing that there will never be peak traffic *into* the city and *out of* the city at the same time, we can find a cheaper solution: We build a road with five lanes and switch the direction of four lanes according to the current traffic. This is why *robust* optimization is especially important in network design.

This thesis focuses on an abstract model for robust network design, neglecting the physical properties of the traffic that is transported through the network. In particular, we do not model properties like gas pressure, resistances of the links or telecommunication protocols. Instead, we merely assume that no goods are lost or created during the transport (a property known as flow conservation) and that no link can be used above its capacity. In principle, this yields a base model for a wide range of applications; however, we focus on telecommunication networks as an application from now on.

In more detail, we consider the following problem. In each scenario, every node can be a source (i.e., supplying goods to the network), a sink (i.e., consuming goods from the network) or a hub (i.e., forwarding all goods). Each sink has a demand that states the amount of goods that it wants to receive. However, the sink cannot specify which source should supply those goods. Likewise, each source specifies the amount of goods that it will supply to the network without declaring which sink will receive them. The capacities of the network must be installed in multiples of a base unit and must be such that in all scenarios, all goods can be sent through the network. Our task is to find such capacities at minimum cost.

We distinguish two different ways to specify the scenarios: In the first problem variant, we explicitly list all possible situations. In the second variant, the situations are described implicitly using linear side constraints. Such constraints can model that a given area of the network receives a certain amount of traffic in all cases or they could limit the amount of traffic sent to a given node in any scenario. In both cases, we call the resulting problem the *Single-Commodity Robust Network Design Problem* (`sRND`). A typical application that is described by the `sRND` model are video-on-demand networks where distributed server farms deliver movies to paying customers. Here, it is irrelevant to the customer which server delivers the movie as long as the entire movie arrives.

The above network design problem is a problem of the type "*Find a combination of elementary decisions that obey certain side constraints while incurring minimum cost.*" As such, it is a problem from *Combinatorial Optimization*. Many combinatorial optimization problems – and in particular our network design problem – are $\mathcal{NP}$-hard. It is commonly believed that this means that there can be no algorithm that efficiently finds a best possible solution on all problem instances. Still, to find reasonably good solutions, we could relax the quality assumption and require that our algorithm must efficiently find a solution whose costs are within a constant factor of the optimum costs, on all problem instances. Such an algorithm is an approximation algorithm. On the other hand, we can relax the requirement that the algorithm must be efficient on all problem instances, as long as it runs in reasonable time on those instances that we actually want to solve. One class of algorithms that makes the latter choice are *Branch-and-Bound* algorithms.

In 2011, Buchheim, Liers and Sanità [BLS11] proposed a Branch-and-Cut algorithm for the `sRND` problem. The algorithm is based on formulating the `sRND` problem as an Integer Linear Program. However, it can only handle the case that the scenario set is given as a finite list and the size of the Integer Linear Program depends on the number of scenarios. In this thesis, we propose a different Branch-and-Cut algorithm whose underlying integer linear programming formulation is independent of the number of scenarios. Moreover, it also works if the scenario set is given by certain linear inequalities.

Outline of the Thesis

Chapter 1 contains a brief introduction into combinatorial optimization and sets the notation for the later chapters. It covers linear programming, Branch-and-Bound algorithms and other fundamental concepts of combinatorial optimization, most notably graphs, network flows and cuts. We formally define the `sRND` problem in *Chapter 2*. Additionally, the chapter

covers the related literature on network design and gives a brief introduction into robust linear programming. In *Chapter 3*, we cover a special case of the `sRND` problem where each scenario has a unique source node and a unique sink node. For this case, there exists a combinatorial algorithm by Gomory and Hu [GH61] that assumes that the network is a complete graph with unit costs. We propose a similar algorithm for a family of unit-cost hypercube instances and show that hypercube instances with binary scenarios are hard to solve for Branch-and-Bound algorithms. *Chapter 4* contains the theoretical foundation for our Branch-and-Cut algorithm. There, we develop an integer linear programming formulation for the `sRND` problem. This formulation is based on so-called cut-set inequalities. We then strengthen our formulation with 3-partition inequalities and show that these inequalities can be obtained as $\{0, \frac{1}{2}\}$-cuts. Both the cut-set inequalities and the 3-partition inequalities are facet-inducing. Our integer linear programming formulation contains an exponential number of cut-set inequalities. We show in *Chapter 5* how to separate the inequalities. This allows us to solve the linear programming relaxation. If the scenario set is given as a finite list, the separation is possible in polynomial time. If the scenario set is given by linear inequalities, the separation problem becomes $\mathcal{NP}$-hard, even if the linear inequalities are so-called box-constraints. We propose an integer linear programming based separation algorithm for this case. In *Chapter 6*, we describe the Branch-and-Bound algorithm and analyze it computationally. The experiments show that the algorithm improves on the previous approach by Buchheim, Liers and Sanità [BLS11]. If the number of scenario is large, it performs better than a commercial solver for integer linear programs and it provides better root bounds than the commercial solver in all cases. For both descriptions of the scenario set, the algorithm is able to solve medium sized instances in reasonable time.

Chapter 1

Preliminaries

This chapter contains all basic definitions that will be used throughout the thesis. Its purpose is to set a consistent notation for the thesis. In addition, the text cites some fundamental theorems that are explicitly or implicitly used later, aiming – within reasonable bounds – to make the thesis self-contained. All definitions and results here are standard and can be found, along with their proofs, in various text books on combinatorial optimization, in particular the ones by Schrijver [Sch86], Nemhauser and Wolsey [NW88], Korte and Vygen [KV12], and the extensive lecture notes by Grötschel [Grö04]. The textbook by Ahuja, Magnanti and Orlin [AMO93] is a good additional reference for the part on network flow theory. All of these references were used as a source for the present text.

1.1 Basic Linear Algebra

The linear algebra part of the chapter mostly follows the exposition in [Grö04].

1.1.1 Vectors and Functions

Throughout this thesis, we assume that vectors $x = (x_1, \ldots, x_n)^{\mathrm{T}} \in \mathbb{R}^n$ are column vectors and denote them with lowercase letters. If we want to stress the geometric interpretation of x, we call x a **point**. We interpret functions $\phi : S \to \mathbb{R}$ over some finite, ordered set $S := \{s^1, \ldots, s^n\}$ as vectors $\phi = (\phi(s^1), \ldots, \phi(s^n))^{\mathrm{T}} \in \mathbb{R}^n$ and write $\mathbb{R}^S := \{\phi \mid \phi : S \to \mathbb{R}\}$ for the set of all functions from S to $\mathbb{R}$. Following this notation, the image of $s_i \in S$ under ϕ is $\phi_i \in \mathbb{R}$. We write $x^{\mathrm{T}}y = \sum_{i=1}^n x_i \cdot y_i$ for the **inner product** of $x, y \in \mathbb{R}^n$. For two sets $X, Y \subseteq \mathbb{R}^n$ the **Minkowski sum** $X + Y$ denotes the set $\{x + y \mid x \in X, y \in Y\}$. If $Y = \{y\}$ consists of a single vector y, we write $X + y$ as a shorter form for $X + \{y\}$.

1.1.2 Matrices

A matrix $A \in \mathbb{R}^{m,n}$ has m rows and n columns. We denote matrices with capital letters. Consequently, we write the entry in the i-th row and the j-th column of A as a_{ij}. We use $a_{i*} \in \mathbb{R}^n$ to refer to the (transposed) i-th row of A and $a_{*j} \in \mathbb{R}^m$ to refer to its j-th column. For a set $I \subseteq \{1, \ldots, m\}$, we denote the submatrix that consists of the rows $(a_{i*})_{i \in I}$ as A_{I*} and we write A_{*J} to denote the columns $(a_{*j})_{j \in J}$ indexed by a set $J \subseteq \{1, \ldots, n\}$.

1.1.3 Combinations of Vectors, Subspaces and Convexity

We say that a vector $x^{k+1} \in \mathbb{R}^n$ is a **linear combination** of k vectors $x^1, \ldots, x^k \in \mathbb{R}^n$ if there exists a $\lambda \in \mathbb{R}^k$ such that $\lambda_1 x^1 + \cdots + \lambda_k x^k = x^{k+1}$. Moreover, the vector x^{k+1} is a

- **conic combination** of $x^1, \ldots, x^k$ if additionally $\lambda \geq 0$.
- **affine combination** of $x^1, \ldots, x^k$ if additionally $\sum_{i=1}^k \lambda_i = 1$.
- **convex combination** of $x^1, \ldots, x^k$ if it is both a conic and an affine one.

The **(linear) subspace** of $\mathbb{R}^n$ induced by all linear combinations of $x^1, \ldots, x^k \in \mathbb{R}^n$ is the **span**

$$\operatorname{span}(\{x^1, \ldots, x^k\}) := \Big\{ \sum_{i=1}^k \lambda_i x^i \;\Big|\; \lambda \in \mathbb{R}^k \Big\} \subseteq \mathbb{R}^n$$

of $\{x^1, \ldots, x^k\}$. Likewise, we call the set of all conic combinations of $x^1, \ldots, x^k \in \mathbb{R}^n$ the **conic subspace** (or **cone**)

$$\operatorname{cone}(\{x^1, \ldots, x^k\}) := \Big\{ \sum_{i=1}^k \lambda_i x^i \;\Big|\; \lambda \in \mathbb{R}^k_{\geq 0} \Big\} \subseteq \mathbb{R}^n$$

spanned by $x^1, \ldots, x^k$, we say that the set of all affine combinations of $x^1, \ldots, x^k \in \mathbb{R}^n$ is the **affine subspace** spanned by $x^1, \ldots, x^k$

$$\operatorname{aff}(\{x^1, \ldots, x^k\}) := \Big\{ \sum_{i=1}^k \lambda_i x^i \;\Big|\; \lambda \in \mathbb{R}^k \wedge \sum_{i=1}^k \lambda_i = 1 \Big\} \subseteq \mathbb{R}^n$$

and call the set of all convex combinations of $x^1, \dots, x^k \in \mathbb{R}^n$ the **convex subspace** spanned by $x^1, \dots, x^k$

$$\operatorname{conv}(\{x^1, \dots, x^k\}) := \left\{ \sum_{i=1}^{k} \lambda_i x^i \;\middle|\; \lambda \in \mathbb{R}^k_{\geq 0} \wedge \sum_{i=1}^{k} \lambda_i = 1 \right\} \subseteq \mathbb{R}^n.$$

We also refer to the latter set as the **convex hull** of $x^1, \dots, x^k$.

Any vector $x = \sum_{i=1}^{k} \lambda_i x^i \in \operatorname{aff}(x^1, \dots, x^k) \subseteq \mathbb{R}^n$ can be equivalently written as $x = x^1 + \sum_{i=2}^{k} \lambda_i (x^i - x^1)$ because it holds that

$$\begin{aligned} x &= x^1 + \sum_{i=2}^{k} \lambda_i (x^i - x^1) = x^1 + \sum_{i=2}^{k} \lambda_i x^i - x^1 \cdot \sum_{i=2}^{k} \lambda_i = x^1 + \sum_{i=2}^{k} \lambda_i x^i - x^1 (1 - \lambda_1) \\ &= x^1 + \sum_{i=2}^{k} \lambda_i x^i - x^1 + \lambda_1 x^1 = \sum_{i=1}^{k} \lambda_i x^i. \end{aligned}$$

It follows that $\operatorname{aff}(x^1, \dots, x^k) = x^1 + \operatorname{span}(x_2 - x^1, \dots, x^k - x^1)$, i.e., the affine subspace spanned by $x^1, \dots, x^k$ is the linear subspace spanned by $x^2 - x^1, \dots, x^k - x^1$ translated by the x^1 vector.

1.1.4 Independence of Vectors

A non-empty set $X = \{x^1, \dots, x^k\} \subseteq \mathbb{R}^n$ of vectors is **linearly/affinely** independent if for all $i = 1, \dots, k$, the element x^i does not lie in the linear/affine subspace induced by $X \setminus \{x^i\}$. We define that the empty set is linearly dependent, but affinely *in*dependent. By choosing $\lambda = 0$, we observe that $0 \in \operatorname{span}(\{x^1, \dots, x^k\})$ and that therefore, any set of vectors containing the zero vector is linearly dependent. Thus, the vectors in X are linearly independent if and only if $\sum_{i=1}^{k} \lambda_i x^i \neq 0$ for all $\lambda \in \mathbb{R}^k \setminus \{0\}$. We can check if a set of vectors is affinely independent by moving the subspace that it spans into the origin: The vectors $x^1, \dots, x^k$ are affinely independent if and only if $x^2 - x^1, \dots, x^k - x^1$ are linearly independent. Linear independence implies affine independence, but the converse is only true if the space spanned by the affinely independent vectors is a proper affine space, i.e., if it does not contain the origin. This is a standard result in linear algebra.

Lemma 1.1. *Let $x^1, \dots, x^k \in \mathbb{R}^n$, $k \leq n$, be affinely independent vectors. Then, the $x^1, \dots, x^k$ are* linearly *independent if and only if $0 \notin \operatorname{aff}(x^1, \dots, x^k)$*

Proof. If $0 \in \operatorname{aff}(x^1, \dots, x^k)$, there is some $\lambda \in \mathbb{R}^k$ with $0 = \sum_{i=1}^{k} \lambda_i x^i$ and $\sum_{i=1}^{k} \lambda_i = 1$. In particular, $\lambda \neq 0$ and the $x^1, \dots, x^k$ are linearly dependent.

For the other direction, assume that $0 \notin \operatorname{aff}(x^1, \dots, x^k)$ and that $x^1, \dots, x^k$ are linearly dependent, i.e., $\sum_{i=1}^{k} \lambda_i x^i = 0$ for some $\lambda \neq 0$. If $\sum_{i=1}^{k} \lambda_i = 0$, we have $\lambda_1 = -\sum_{i=2}^{k} \lambda_i$ and it follows

$$0 = \sum_{i=1}^{k} \lambda_i x^i = \sum_{i=2}^{k} \lambda_i x^i + \lambda_1 x^1 = \sum_{i=2}^{k} \lambda_i x^i - \sum_{i=2}^{k} \lambda_i x^1 = \sum_{i=2}^{k} \lambda_i (x^i - x^1).$$

Now, the $\lambda_2, \ldots, \lambda_k$ cannot all be zero, as otherwise we would have $\lambda_1 = 0$, too, and then $\lambda = 0$. Consequently, $x^1, \ldots, x^k$ are affinely dependent – a contradiction.

Otherwise, set $\mu := \sum_{i=1}^{k} \lambda_i \neq 0$ and then

$$\sum_{i=1}^{k} \lambda_i x^i = 0 \iff \sum_{i=1}^{k} \frac{\lambda_i}{\mu} x^i = 0 \iff \sum_{i=1}^{k} \lambda_i' x^i = 0$$

for $\lambda_i' = \lambda_i / \mu$. This is in contradiction to our assumption, because together with the fact that $\sum_{i=1}^{k} \lambda_i' = 1$ it implies that $0 \in \operatorname{aff}(x^1, \ldots, x^k)$. □

1.1.5 Dimension

The **dimension** $\dim X$ of a set $X \subseteq \mathbb{R}^n$ is the cardinality of a largest affinely independent subset of X, minus one. In particular, the dimension of $\operatorname{span}(x^1, \ldots, x^k)$ is n if and only if the cardinality of a largest *linearly* independent subset of $\operatorname{span}(x^1, \ldots, x^k)$ is n: Let $\{y^1, \ldots, y^n\}$ be such a set. Then $y^i \neq 0$ for all $i = 1, \ldots, n$ and $\{0, y^1, \ldots, y^n\}$ is a subset of $\operatorname{span}(x^1, \ldots, x^k)$ that contains $n + 1$ affinely independent vectors (we observe that $y^i - 0$, $i = 1, \ldots, n$, are linearly independent). On the other hand, the existence of $n' > n + 1$ affinely independent vectors would imply the existence of $n' - 1 > n$ linearly independent vectors in $\operatorname{span}(x^1, \ldots, x^k)$. As we have previously observed, the affine subspace induced by $x^1, \ldots, x^k$ can be written as $x^1 + \operatorname{span}(\{x^2 - x^1, \ldots, x^k - x^1\})$. Its dimension is exactly the dimension of $\operatorname{span}(x^2 - x^1, \ldots, x^k - x^1)$. We say that $X \subseteq \mathbb{R}^n$ is full-dimensional if $\dim X = n$.

1.1.6 Hyperplanes and Half-Spaces

A **hyperplane** in $\mathbb{R}^n$ is an affine subspace of dimension $n - 1$. Hyperplanes are exactly the subspaces that can be written as

$$H_{a,b} := \{x \in \mathbb{R}^n \mid a^{\mathrm{T}} x = b\} \subseteq \mathbb{R}^n$$

for some $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. A **halfspace** in $\mathbb{R}^d$ is a set of the form

$$\{x \in \mathbb{R}^n \mid a^{\mathrm{T}} x \geq b\} \subseteq \mathbb{R}^n$$

and its boundary is the hyperplane $H_{a,b}$.

1.2 Polyhedral Theory

The presentation of polyhedral theory and linear programming is mainly based on the textbooks by Nemhauser and Wolsey [NW88] and Schrijver [Sch86].

1.2.1 Convex Sets, Extreme Points and Rays

We say that a set $X \subseteq \mathbb{R}^n$ is **convex** if for any two points $x, y \in X$ the line between x and y is contained in X, i.e., if $\lambda x + (1 - \lambda) y \in X$ for any $\lambda \in [0, 1]$. The linear, the conic, the

affine and the convex subspace spanned by any $x^1, \dots, x^k \in \mathbb{R}^n$ are all convex sets, as is any half-space.

Let now $X \subseteq \mathbb{R}^n$ be any convex set. A point $x \in X$ is an **extreme point** of X if x does not lie strictly between two points in X, i.e., if there are no $y, z \in X \setminus x$ such that $x = \frac{1}{2}y + \frac{1}{2}z$. Moreover, $x \in X$ is an **interior point** of X if there is an $\varepsilon > 0$ such that the open ball of radius ε around x completely lies in X, i.e., if $x + \{y \in R^n \mid \|y\| < \varepsilon\} \subseteq X$. Lastly, $x \in X$ is an **inner point** of X if it is neither an extreme nor an interior point of X.

A convex set $X \subseteq \mathbb{R}^n$ is unbounded if there are vectors $x, r \in \mathbb{R}^n$ such that $x + \lambda r \in X$ for all $\lambda \geq 1$. We call r a **unbounded direction** or **ray** of X. Analogously to extreme points, a ray r is an **extreme ray** if it cannot be obtained as a convex combination of other rays, i.e., if there are no rays $r^1, r^2 \in \mathbb{R}^n \setminus r$ such that $r = \frac{1}{2}r^1 + \frac{1}{2}r^2$.

1.2.2 Polyhedra

A **polyhedron** is the intersection of finitely many half-spaces, i.e., a set of the form

$$\mathfrak{P}_{A,b} = \{x \in \mathbb{R}^n \mid Ax \geq b\}$$

where $A \in \mathbb{R}^{m,n}$ is a matrix and $b \in \mathbb{R}^m$ is a vector. Sets of the form $\{x \in \mathbb{R}^n_{\geq 0} \mid Ax \leq b\}$ are polyhedra as well; they can be rewritten as $\{x \in \mathbb{R}_{\geq 0} \mid (-A)x \geq -b\}$. The intersection of finitely many polyhedra is a polyhedron. In particular, a set $\{x \in \mathbb{R}^n_{\geq 0} \mid Ax = b\}$ is a polyhedron because it is the intersection of $\{x \in \mathbb{R}^n_{\geq 0} \mid Ax \geq b\}$ and $\{x \in \mathbb{R}^n_{\geq 0} \mid Ax \leq b\}$. We use the Fraktur typeface to denote polyhedra. Being the intersection of convex sets, $\mathfrak{P}_{A,b}$ is convex, too. We say that a polyhedron $\mathfrak{P}$ is **rational** if there exists a matrix $A \in \mathbb{Q}^{m,n}$ and a vector $b \in \mathbb{Q}^m$ such that $\mathfrak{P} = \{x \in \mathbb{R}^n \mid Ax \geq b\}$. Throughout this thesis, we only consider rational polyhedra and given that we want to represent polyhedra in a computer, this is a reasonable assumption.

If $\mathfrak{P}$ is **bounded**, we say that $\mathfrak{P}$ is a **polytope**. In this case, there exists a $B \in \mathbb{R}$ such that $\|x\| \leq B$ for all $x \in \mathfrak{P}$.

We say that the inequality $a^{\mathrm{T}}x \geq b$ is **valid** for a polyhedron $\mathfrak{P}$ if it is satisfied by all $x \in \mathfrak{P}$ or, equivalently, if $\mathfrak{P} \subseteq \{x \in \mathbb{R}^n \mid a^{\mathrm{T}}x \geq b\}$. A subset $\mathfrak{F} \subseteq \mathfrak{P}$ is a **face** of $\mathfrak{P}$ if there is a valid inequality $a^{\mathrm{T}}x \geq b$ for $\mathfrak{P}$ such that $\mathfrak{F} = \{x \in \mathfrak{P} \mid a^{\mathrm{T}}x = b\}$. In this case, we say that $a^{\mathrm{T}}x \geq b$ **induces the face** $\mathfrak{F}$. We observe that $\mathfrak{F}$ is a polyhedron. If $\mathfrak{F}$ is non-empty, we say that $a^{\mathrm{T}}x \geq b$ **supports** $\mathfrak{P}$. A non-empty face $\mathfrak{F}$ of $\mathfrak{P}$ is a **facet** of $\mathfrak{P}$ if $\dim \mathfrak{F} = \dim \mathfrak{P} - 1$. It is a **vertex** of $\mathfrak{P}$ if $\dim \mathfrak{F} = 0$. A vector $x \in \mathfrak{P}$ is an extreme point of $\mathfrak{P}$ if and only if $\{x\}$ is a 0-dimensional face of $\mathfrak{P}$. We therefore use *vertex* and *extreme point* interchangeably in the sequel. We can determine the dimension of any face $\mathfrak{F}$ of $\mathfrak{P}_{A,b}$ by checking the rank of the rows of $Ax \geq b$ that are exactly satisfied by $\mathfrak{F}$.

Lemma 1.2 (Proposition 2.4 in [NW88])**.** *Let $\mathfrak{P}_{A,b} = \{x \in \mathbb{R}^n \mid Ax \geq b\} \subseteq \mathbb{R}^n$ be a polyhedron and let $\mathfrak{F}$ be a face of $\mathfrak{P}$. If we define the set $I := \{i \mid a^{\mathrm{T}}_{i*}x = b \text{ for all } x \in \mathfrak{F}\}$, then*

$$\dim \mathfrak{F} = n - \operatorname{rank}(A_{I*} \,|\, b_I)$$

where $(A_{I*} \,|\, b_I)$ is the subsystem of $Ax \geq b$ that consists of the rows indexed by I. □

Our definition of a polyhedron characterizes polyhedra with a finite number of linear inequalities, its **linear description**. The importance of facets lies in the fact that they are necessary for the linear description of $\mathfrak{P}$ in the following sense.

Theorem 1.3 (Theorem 3.5 in [NW88])**.** *Any polyhedron $\mathfrak{P}$ has a (inclusion-wise) minimal description by linear inequalities that is unique up to scalar multiplication and there is a one-to-one correspondence between the facets of $\mathfrak{P}$ and the inequalities in the minimal linear description.* □

In particular, any polyhedron $\mathfrak{P}_{A,b}$ has a finite number of facets, namely at most as many as there are rows in A. Equivalently, we can characterize a polyhedron by its vertices and extreme rays.

Theorem 1.4 (Theorem 4.8 in [NW88])**.** *A set $\mathfrak{P} \subseteq \mathbb{R}^n$ is a polyhedron if and only if it can be written as*

$$\mathfrak{P} = \operatorname{conv}(x^1, \dots, x^k) + \operatorname{cone}(y^1, \dots, y^l)$$

where $x^1, \dots, x^k \in \mathbb{R}^n$ are its vertices and $y^1, \dots, y^\ell \in \mathbb{R}^n$ are its extreme rays. □

In particular, the number of vertices and extreme rays of any polyhedron is finite.

Corollary 1.5. *Any extreme point x of $\mathfrak{P}_{A,b} \subseteq \mathbb{R}^n$ is the intersection of n linearly independent hyperplanes defined by the rows of $Ax \geq b$. Consequently, the polyhedron $\mathfrak{P}_{A,b}$ can have at most $\binom{m}{n}$ vertices, where m is the number of rows in A.* □

If a polyhedron is bounded, i.e., if it is a polytope, we do not need the conic part of the description.

Corollary 1.6 (Theorem 3.31 and Corollary 3.32 in [KV12])**.** *A set $\mathfrak{P} \subseteq \mathbb{R}^n$ is a polytope if and only if it can be written as*

$$\mathfrak{P} = \operatorname{conv}(x^1, \dots, x^k)$$

where $x^1, \dots, x^k \in \mathbb{R}^n$ are the vertices of $\mathfrak{P}$. □

It is possible to convert between a linear description and a description by extreme points and rays, but only at the expense of the size of the description.

Example 1.7. The *n-dimensional unit hypercube polytope* defined by

$$\mathfrak{C}_n = \{x \in \mathbb{R}^n \mid 0 \leq x_i \leq 1 \text{ for all } i = 1, \dots, n\}$$

has a linear description of size $\Theta(n)$. Yet, any vector $x \in \{0,1\}^n$ satisfies n linearly independent inequalities with equality and is thus a vertex of $\mathfrak{C}_n$. Thus, the hypercube polytope has 2^n vertices.

In practice, we use software like PORTA [CL08] or Polymake [GJ00] for the conversion. Due to the exponential worst-case running time of the algorithms and the huge output size, however, the conversion is only feasible for small polytopes.

1.3 Linear Programming

Linear programming is the task to optimize a linear objective function over a polyhedron and a **linear program** (LP) is given by a system

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j && && (1.1)\\ \text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i && \text{for all } i = 1, \dots, m \\ & x_j \geq 0 && \text{for all } j = 1, \dots, n \end{aligned}$$

of linear inequalities and a linear **objective function** $\sum_{j=1}^{n} c_j x_j$. We call the i-th row $a_{i*}^{\mathrm{T}} x \geq b_i$ of the system the i-th **constraint** and call x_j the j-th **variable**. The vector b defines the **right-hand side** of the constraints.

The above form of the linear program is the standard form that we will use throughout this thesis. Still, it is straight-forward to include different types of constraints in the program: Constraints of the form $a^{\mathrm{T}} x \leq b$ can be included by multiplying both sides with -1 and we can include constraints of the form $a^{\mathrm{T}} x = b$ by adding both the constraints $a^{\mathrm{T}} x \geq b$ and $a^{\mathrm{T}} x \leq b$. Likewise, the assumption that $x \geq 0$ is without loss of generality, as two variables $y - z = x$ with $y, z \in \mathbb{R}_{\geq 0}$ model a general variable $x \in \mathbb{R}$. In order to maximize the objective function $\sum_{j=1}^{n} c_j x_j$, one can minimize $-\sum_{j=1}^{n} c_j x_j$ instead.

Given a vector $x \in \mathbb{R}^n$, we say that the **slack** of the inequality $a^{\mathrm{T}} x \geq b$ at x is

$$\operatorname{slack}(a, b, x) = b - a^{\mathrm{T}} x.$$

If the slack of $a^{\mathrm{T}} x \geq b$ at x is zero, we say that the inequality is **tight** at x. We say that $a^{\mathrm{T}} x \geq b$ is **satisfied** by x if $\operatorname{slack}(a, b, x) \geq 0$ and that $a^{\mathrm{T}} x \geq b$ is **violated** by x if $\operatorname{slack}(a, b, x) < 0$. In the latter case, let

$$\operatorname{viol}(a, b, x) = -\operatorname{slack}(a, b, x) = a^{\mathrm{T}} x - b$$

be the **violation** of $a^{\mathrm{T}} x \geq b$ at x.

1.3.1 Optimum Solutions

The **feasible region** $\{x \in \mathbb{R}^n_{\geq 0} \mid Ax \geq b\}$ of $\min\{c^{\mathrm{T}} x \mid x \in \mathbb{R}^n_{\geq 0} \text{ s.t. } Ax \geq b\}$ is a polyhedron $\mathfrak{P}_{A,b}$. If $\mathfrak{P}_{A,b}$ is empty, we say that the program is **infeasible**. If on the other hand $\min\{c^{\mathrm{T}} x \mid x \in \mathbb{R}^n_{\geq 0} \text{ s.t. } Ax \geq b\} = -\infty$, we say that the program is **unbounded**.

Theorem 1.8 (Theorem 1.8 in [KV12])**.** *If a linear program is neither infeasible nor unbounded, then the objective function attains the optimum.* □

We use this fact as a justification to not distinguish minima and infima in this case. If a linear program has an optimum solution, i.e., if it is neither infeasible nor unbounded, it is well-known that this optimum solution can be attained in a vertex of the polyhedron.

Theorem 1.9 (Theorem 4.5 in [NW88])**.** *If the linear objective function $c^{\mathrm{T}} x$ attains a minimum/maximum on a polyhedron $\mathfrak{P}_{A,b}$, then it attains a minimum/maximum in a vertex of $\mathfrak{P}_{A,b}$, respectively.* □

1.3.2 Duality

Duality provides certificates for optimality in the following way. Any conic combination

$$\sum_{i=1}^{m} \lambda_i \sum_{j=1}^{n} a_{ij} x_j \geq \sum_{i=1}^{m} \lambda_i b_i \qquad \text{with } \lambda \geq 0$$

of the inequalities in A is a valid inequality for $\mathfrak{P}_{A,b}$ and we can rewrite that inequality equivalently as

$$\sum_{j=1}^{n} x_j \sum_{i=1}^{m} a_{ij} \lambda_i \geq \sum_{i=1}^{m} \lambda_i b_i \qquad \text{with } \lambda \geq 0.$$

Thus, if $\lambda \geq 0$ is chosen such that $\sum_{i=1}^{m} a_{ij}\lambda_i \leq c_j$ for all $j = 1, \ldots, n$, we obtain

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{j=1}^{n} (\sum_{i=1}^{m} \lambda_i a_{ij}) x_j = \sum_{i=1}^{m} \lambda_i \sum_{j=1}^{n} a_{ij} x_j \geq \sum_{i=1}^{m} \lambda_i b_i.$$

This proves that for any $\lambda \geq 0$ with $\sum_{i=1}^{m} a_{ij}\lambda_i \leq c_j$ for all $j = 1, \ldots, n$, the sum $\sum_{i=1}^{m} \lambda_i b_i$ is a lower bound on the objective function. We will see that maximizing this sum over all valid choices of λ provides the best possible lower bound. The maximization corresponds to the following program.

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} b_i \lambda_i & & (1.2) \\ \text{s.t.} \quad & \sum_{i=1}^{m} a_{ij} \lambda_i \leq c_j & & \text{for all } j = 1, \ldots, n \\ & \lambda_i \geq 0 & & \text{for all } i = 1, \ldots, m \end{aligned}$$

Program (1.2) is again a linear program. It has a variable for each constraint in (1.1) and a constraint for each variable in (1.1). We call it the **dual** of the **primal** program (1.1) and the dual program of (1.2) is again the program (1.1). In matrix notation, the dual reads $\max\{b^{\mathrm{T}}\lambda \mid \lambda \in \mathbb{R}^m \text{ s.t. } A^{\mathrm{T}}\lambda \leq c\}$. By the above argumentation, it is true that $b^{\mathrm{T}}\lambda \leq c^{\mathrm{T}}x$ for all feasible solutions x, λ of (1.1) and (1.2), respectively, and this observation is called the *weak duality principle*. However, these feasible solutions do not necessarily exist.

Theorem 1.10 (Proposition 2.2 and Corollary 2.5 in [NW88])**.** *Consider a pair of a primal linear program* $\min\{c^{\mathrm{T}}x \mid Ax \geq b,\ x \in \mathbb{R}^n_{\geq 0}\}$ *and its dual* $\max\{b^{\mathrm{T}}\lambda \mid A^{\mathrm{T}}\lambda \leq c,\ \lambda \in \mathbb{R}^m_{\geq 0}\}$*. It holds:*

1. *If the primal program is infeasible, then the dual program is either infeasible as well or it is unbounded.*
2. *If the primal program is unbounded, then the dual program is infeasible.*
3. *The primal program is feasible and* bounded *if and only if the dual program is feasible and bounded. In this case, we have* $b^{\mathrm{T}}\lambda \leq c^{\mathrm{T}}x$ *for any pair* λ, x *of feasible solutions.*

□

And of course, the converse is also true: If the dual is infeasible, then the primal is either infeasible or unbounded. An unbounded dual program implies an infeasible primal one. A useful criterion for feasibility is Farkas' Lemma, see Theorem 2.7 in [NW88].

Lemma 1.11 (Farkas' Lemma)**.** *A primal problem in standard form is feasible (i.e.,* $\{x \in \mathbb{R}^n_{\geq 0} \mid Ax \geq b\} \neq \emptyset$*) if and only if* $b^{\mathrm{T}}\lambda \leq 0$ *for all* $\lambda \in \mathbb{R}^m_{\geq 0}$ *with* $A^{\mathrm{T}}\lambda \leq 0$*.* □

Equivalently, the primal program given by $\{x \in \mathbb{R}^n_{\geq 0} \mid Ax \geq b\}$ is feasible if and only if $\max\{b^{\mathrm{T}}\lambda \mid A^{\mathrm{T}}\lambda \leq 0, \lambda \in \mathbb{R}^m_{\geq 0}\} \leq 0$. If both the primal and the dual program have a feasible solution, however, the bound provided by the dual program is tight.

Theorem 1.12 (Theorem 2.4 and Proposition 2.6 in [NW88])**.** *Consider a pair of a primal linear program* $\min\{c^{\mathrm{T}}x \mid Ax \geq b,\ x \in \mathbb{R}^n_{\geq 0}\}$ *and its dual* $\max\{b^{\mathrm{T}}\lambda \mid A^{\mathrm{T}}\lambda \leq c,\ \lambda \in \mathbb{R}^m_{\geq 0}\}$*. Let* $x \in \mathbb{R}^n_{\geq 0}$ *be a feasible solution for* $Ax \geq b$ *and let* $\lambda \in \mathbb{R}^m_{\geq 0}$ *be a feasible solution for* $A^{\mathrm{T}}\lambda \leq c$*. Then the following statements are equivalent.*

1. *The vectors* x *and* λ *are optimum for their respective programs.*
2. *We have* $c^{\mathrm{T}}x = b^{\mathrm{T}}\lambda$*.*
3. *For all* $j = 1, \ldots, n$ *with* $x_j > 0$ *we have* $\sum_{i=1}^m a_{ij}\lambda_i = c_j$ and *for all* $i = 1, \ldots, m$ *with* $\lambda_i > 0$*, we have* $\sum_{j=1}^n a_{ij}x_j = b_i$*. This means that if the inequality defined by the* j*-th row of* (A, b) *is not tight, then the corresponding dual variable must be zero; and that if the dual inequality given by the* i*-th row of* (A^T, c) *is not tight, then the corresponding primal variable must be zero.*

□

Property 2 is often referred to as the *strong duality principle* and Property 3 is called *complementary slackness.*

1.3.3 Cutting Plane Algorithms

Linear Programs can be solved in a time that is polynomial in the size of the constraint system $Ax \geq b$. For some problems, however, there are linear programming formulations whose size is exponential in the input size of the problem. Thus, it is inefficient to solve the entire linear program with a linear programming solver.

Still, in order to optimize a fixed objective function $c^{\mathrm{T}}x$ over the system, we do not need a full description of $\mathfrak{P}_{A,b}$. Theoretically, it is enough to know n rows of (A, b) that define an optimum vertex of $\mathfrak{P}_{A,b}$ with respect to $c^{\mathrm{T}}x$. However, finding these n constraints is as hard as solving the original problem. It still shows that if we fix the objective function, we can omit certain constraints without changing the optimum solution.

Cutting plane algorithms exploit this idea. Such an algorithm starts by solving a small subproblem $\min\{c^{\mathrm{T}}x \mid A^1x \geq b^1\}$, obtaining a first solution x^1. It then needs to solve the **separation problem**: Find a row of row $a_{i*}^{\mathrm{T}}x \geq b_i$ of $Ax \geq b$ such that $a_{i*}^{\mathrm{T}}x^1 < b_i$ or prove that none such row exists. If a new row is found, the algorithm adds it to the current subsystem and since this new row cuts off the current solution x^1, it is called a **cutting plane** or simply a **cut**. Otherwise, no more cutting planes can be found and the algorithm stops with a solution x^*. This solution x^* is feasible for $Ax \geq b$, because if it were not, the algorithm would have found another cutting plane. It is also optimum for $Ax \geq b$, because any solution for $Ax \geq b$ is feasible for all subsystems that the algorithm considers.

It can happen that the separation problem is $\mathcal{NP}$-hard to solve or that the algorithm needs to perform an exponential number of iterations to solve the linear program. We will see examples for both cases later. However, a famous theorem tells us that (up to certain technical conditions) the separation problem is polynomial time solvable if and only if the original problem is polynomially solvable. This theorem is generally attributed to Grötschel, Lovász and Schrijver [GLS81; GLS84] and, according to Korte and Vygen [KV12], it was independently discovered by Karp and Papadimitriou [KP80; KP82] as well as Padberg and Rao [PR81]. It not only holds for polyhedra, but also for general convex sets.

If an algorithm can find *some* cutting planes but cannot prove that no more cutting planes exist, we say that the algorithm is a **heuristic separation algorithm**. Otherwise, the algorithm is an **exact separation algorithm**.

Generating a violated cutting plane for the dual problem corresponds to generating a variable that improves the objective value for the primal problem. This method is called **column generation**. The problem of finding an improving variable is called **pricing problem**.

1.3.4 Integer Linear Programs

Many optimization problems can be modeled more easily by restricting a subset of the variables to integer values. Such variables can model binary decisions or indivisible goods that only exist in multiples of some base unit. We a resulting problem of the form

$$\min\{c^{\mathrm{T}}x + d^{\mathrm{T}}y \mid (x,y) \in \mathbb{Z}^{n_1}_{\geq 0} \times \mathbb{R}^{n_2}_{\geq 0} \text{ such that } Ax + By \geq b\} \tag{1.3}$$

a **Mixed Integer Linear Program** (or **MIP**) if both integer and general variables are present. If the program only has integer variables, we say that it is an **Integer Linear Program** (or **IP**). The convex hull

$$\mathfrak{P}_{A,B,b} = \operatorname{conv}\{(x,y) \in \mathbb{Z}^{n_1}_{\geq 0} \times \mathbb{R}^{n_2}_{\geq 0} \mid Ax + By \geq b\}$$

of the set of **integer feasible solutions** is again a polyhedron. We say that $(x,y) \in \mathfrak{P}_{A,B,b}$ is an **integer solution** or **integer feasible** if x is integer. All vertices of $\mathfrak{P}_{A,B,b}$ are integer feasible.

Therefore, there is an equivalent formulation of (1.3) as a linear program without integer variables. In general, however, the size of this formulation can be exponential in the size of (1.3) and in fact, optimizing a general MIP is $\mathcal{NP}$-hard [Kar72].

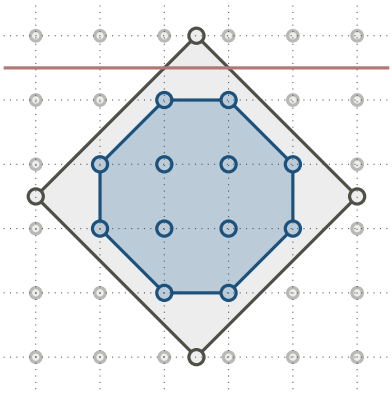
A classical approximation of this mixed integer linear program is the **linear programming relaxation** (LP relaxation) of (1.3)

$$\min\{c^{\mathrm{T}}x + d^{\mathrm{T}}y \mid (x,y) \in \mathbb{R}^{n_1}_{\geq 0} \times \mathbb{R}^{n_2}_{\geq 0} \text{ such that } Ax + By \geq b\} \tag{1.4}$$

by removing the integrality condition on the x variables. Its feasible region

$$\widetilde{\mathfrak{P}}_{A,B,b} = \{(x,y) \in \mathbb{R}^{n_1}_{\geq 0} \times \mathbb{R}^{n_2}_{\geq 0} \mid Ax + By \geq b\} \tag{1.5}$$

is an outer approximation of $\mathfrak{P}_{A,B,b}$, i.e., it holds that $\mathfrak{P}_{A,B,b} \subseteq \widetilde{\mathfrak{P}}_{A,B,b}$. We say that $(x,y) \in \widetilde{\mathfrak{P}} \setminus \mathfrak{P}$ is a **fractional solution** or **fractionally feasible**. By cutting off fractional



$$\begin{array}{rcrcr} -x & & +y & \leq & 2.5 \\ x & & +y & \leq & 3.5 \\ x & & -y & \leq & 2.5 \\ -x & & -y & \leq & 1.5 \\ & & x, y & \in & \mathbb{Z} \end{array}$$

Figure 1.1: The picture shows a 2-dimensional lattice. The blue points are the feasible solutions of the integer linear program on the right; the blue region depicts their convex hull. The feasible region of the corresponding linear programming relaxation is drawn in gray. Its vertices are not lattice points. In order to equivalently rewrite the integer linear program as a linear program without explicit integrality conditions, we would need to find the eight blue facets of the convex hull of its feasible solutions. In red: A valid cutting-plane that would strengthen the linear programming relaxation. The picture was inspired by the famous book cover of Schrijver's textbook on "*The Theory of Linear and Integer Programming*" [Sch86].

solutions of $\widetilde{\mathfrak{P}}_{A,B,b}$ with cutting planes, we can bring the LP relaxation closer to the original MIP (see next subsection). The situation is illustrated by Figure 1.1.

If all vertices (x, y) of the LP relaxation $\widetilde{\mathfrak{P}}_{A,B,b}$ are integer feasible, we have $\mathfrak{P}_{A,B,b} = \widetilde{\mathfrak{P}}_{A,B,b}$. We can thus remove the integrality conditions without changing the problem. This happens independently of the choice of any integer $b \in \mathbb{Z}^m$ if and only if the determinant of each square submatrix of the constraint matrix A is 1, -1 or 0. In this case, the matrix A is **totally unimodular**.

Lemma 1.13 (Proposition 2.1 in [NW88])**.** *The following statements are equivalent.*

1. *The matrix $A \in \mathbb{R}^{m,n}$ is totally unimodular.*
2. *The transpose A^{T} of A is totally unimodular.*
3. *A matrix obtained by deleting a row of A that is a unit vector is unimodular.*
4. *A matrix obtained by multiplying a row of A by -1 is totally unimodular.*

□

Chvátal-Gomory Cuts

Parts of this subsection are based on the concise but very useful introduction to Chvátal-Gomory cuts by Fischetti and Lodi [FL07] and the one by Caprara and Fischetti [CF96]. These cutting planes provide one possibility to derive a complete linear description of the convex hull of integer feasible solutions. For any given linear program $Ax \geq b$ with a constraint matrix $A = (a_{ij}) \in \mathbb{Z}^{m \times n}$ and vectors $x \in \mathbb{R}^n, b \in \mathbb{Z}^m$ one can generate a valid

inequality for $Ax \geq b$ by computing an arbitrary conic combination with coefficients $\lambda \geq 0$

$$\sum_{i=1}^{m} \lambda_i \sum_{j=1}^{n} a_{ij} x_j \geq \sum_{i=1}^{m} \lambda_i b_i \tag{1.6}$$

of the rows of $Ax \geq b$. By rounding up the left-hand side coefficients, we know that the value $\sum_{j=1}^{n} \lceil \sum_{i=1}^{m} \lambda_i a_{ij} \rceil x_j$ of the left-hand side will be integer for any integer x. The inequality therefore remains valid if we round up the right-hand side of (1.6) to the next integer as well. We thus obtain

$$\sum_{j=1}^{n} \Big\lceil \sum_{i=1}^{m} \lambda_i a_{ij} \Big\rceil x_j \geq \Big\lceil \sum_{i=1}^{m} \lambda_i b_i \Big\rceil \tag{1.7}$$

as a valid inequality for $Ax \geq b, x \in \mathbb{Z}$. If the right-hand side of (1.6) is not integer, the inequality (1.7) can be used to cut off fractional solutions of a linear program. It is generally referred to as a *Chvátal-Gomory Cut* and is due to Gomory [Gom58] and Chvátal [Chv73].

We define the linear description that arises from taking all Chvátal-Gomory cuts over the original formulation $Ax \geq b$ as the first Chvátal-Gomory closure

$$\mathfrak{G}^1(A,b) := \Big\{ x \in \mathbb{R}^n_{\geq 0} \;\Big|\; Ax \geq b \wedge \lceil \lambda^{\mathrm{T}} A \rceil x \geq \lceil \lambda^{\mathrm{T}} b \rceil \text{ for all } \lambda \in \mathbb{R}^m_{\geq 0} \Big\}.$$

Chvátal [Chv73] showed that even though the definition of $\mathfrak{G}^1$ includes an inequality for every $\lambda \in \mathbb{R}^m_{\geq 0}$, a finite number of inequalities suffice to define $\mathfrak{G}^1$. Thus, the first Chvátal-Gomory closure $\mathfrak{G}^1(A,b)$ is a polyhedron. By iterating the process, we define the i-th Chvátal-Gomory closure as the first Chvátal-Gomory closure of the $(i-1)$-th, i.e., we set $\mathfrak{G}^i(A,b) := \mathfrak{G}^1(\mathfrak{G}^{i-1}(A,b))$ which is again a polyhedron by induction. We say that a Chvátal-Gomory cut is of rank i if the i-th Chvátal-Gomory closure is the first that contains the cut.

If all variables of $Ax \geq b$ are integer and bounded by some value B, then for some finite $i \in \mathbb{Z}_{\geq 0}$, the i-th Chvátal-Gomory closure exactly describes the convex hull of all integer feasible solutions, i.e.,

$$\operatorname{conv}\{x \in [0,B]^n \cap \mathbb{Z}^n \mid Ax \geq b\} = \mathfrak{G}^i(A,b)$$

for some finite $i \in \mathbb{Z}_{\geq 0}$. However, the correct value of i can be very large. Moreover, the separation problem for Chvátal-Gomory cuts is $\mathcal{NP}$-hard [Eis99]. This is why practical algorithms usually separate Chvátal-Gomory cuts heuristically.

The Special Case of Zero-Half Cuts

Caprara and Fischetti [CF96] consider the special case that $\lambda_i \in \{0, \frac{1}{2}\}$ for all $i = 1, \ldots, m$ and refer to the resulting cuts as $\{0, \frac{1}{2}\}$-Chvátal-Gomory cuts (or, simply $\{0, \frac{1}{2}\}$-cuts). This special case will be of importance in the later chapters and we take some time to find a necessary condition for a $\{0, \frac{1}{2}\}$-cut to be violated. Assume for the analysis that λ is chosen such that the left-hand side coefficients are integer and do not need to be rounded. Then, building a $\{0, \frac{1}{2}\}$-cut is equivalent to selecting some subset $I \subseteq \{1, \ldots, m\}$ of the constraints and then computing the inequality

$$\frac{1}{2} \cdot \sum_{i \in I} \sum_{j=1}^{n} a_{ij} x_j \geq \Big\lceil \frac{1}{2} \sum_{i \in I} b_i \Big\rceil.$$

The **violation** of a $\{0, \frac{1}{2}\}$-cut with respect to some fractional solution x^* is

$$\operatorname{viol}(I, x^*) := \sum_{i \in I} \operatorname{viol}(a_{i*}, b_i, x^*) = \left\lceil \frac{1}{2} \cdot \sum_{i \in I} b_i \right\rceil - \sum_{i \in I} \frac{1}{2} \sum_{j=1}^{n} a_{ij} x_j^*.$$

In order to use $\{0, \frac{1}{2}\}$-cuts in a cutting plane algorithm, we often wish to maximize the violation or, at the very least, to find a $\{0, \frac{1}{2}\}$-cut with $\operatorname{viol}(I, x^*) > 0$. It is well-known that $\operatorname{viol}(I, x^*)$ is inversely proportional to the sum of the slacks of the participating inequalities

$$\operatorname{slack}(I, x^*) := \sum_{i \in I} \operatorname{slack}(a_{i*}, b_i, x^*) = \sum_{i \in I} \Big(\sum_{j=1}^{n} a_{ij} x_j^* - b_i \Big)$$

due to the following analysis.

$$\begin{aligned} \operatorname{viol}(I, x^*) &= \left\lceil \frac{1}{2} \cdot \sum_{i \in I} b_i \right\rceil - \frac{1}{2} \sum_{i \in I} \sum_{j=1}^{n} a_{ij} x_j^* \\ &= \left\lceil \frac{1}{2} \cdot \sum_{i \in I} b_i \right\rceil - \frac{1}{2} \sum_{i \in I} b_i - \frac{1}{2} \sum_{i \in I} \operatorname{slack}(a_{i*}, b_i, x^*) \end{aligned}$$

The resulting $\{0, \frac{1}{2}\}$-cut is never violated by any fractionally feasible x^* if $\sum_{i \in I} b_i$ is even; in that case, the rounding does not increase its right-hand side and the cut is a linear combination of the rows in A. This is consistent with the above analysis: If x^* is fractionally feasible for $Ax \geq b$, the slack $\operatorname{slack}(a_{i*}, b_i, x^*)$ of all rows $i \in I$ is non-negative. This implies that $\operatorname{viol}(I, x^*) \leq 0$ if $\sum_{i \in I} b_i$ is even.

If otherwise $\sum_{i \in I} b_i$ is odd, then the violation of the resulting cut is exactly

$$\frac{1}{2} - \frac{1}{2} \sum_{i \in I} \operatorname{slack}(a_{i*}, b_i, x^*).$$

In particular, the violation is maximum if the inequalities in I are tight at x^* and decreases with increasing slacks $\operatorname{slack}(a_{i*}, b_i, x^*)$ of the rows $i \in I$. Moreover, if the rows in I have a total slack of at least 1, the resulting $\{0, \frac{1}{2}\}$-cut is redundant.

Summarizing, it is desirable to use inequalities with low slack, preferably tight inequalities, to generate a $\{0, \frac{1}{2}\}$-cut. In general, the $\{0, \frac{1}{2}\}$-cut separation problem is $\mathcal{NP}$-hard as well [CF96].

Branch-and-Bound Algorithms

Modeling combinatorial optimization problems as an MIP is of no use if the resulting MIP cannot be solved. Still, solving the linear programming relaxation does not always provide sufficient bounds, even if we strengthen the linear programming relaxation with additional cutting planes. Thus, we can be stuck with a fractional solution at some point. Branch-and-Bound algorithms solve this problem by *Branching*, i.e., they divide the original problem into several disjoint subproblems such that any integer optimum solution is still

feasible in at least one of the subproblems while the current fractional solution is cut off in all of them. The branching is generally done by inserting additional constraints into the problem or by fixing variables. The algorithm then recurses on the subproblems. We call the resulting search tree the **Branch-and-Bound tree**. The root of the tree corresponds to the original problem while all other nodes correspond to recursively created subproblems. An extensive introduction to Branch-and-Bound algorithms can be found in [EGJR01] and [Mar01]. Assuming that the algorithm solves a minimization problem, the *Bound* component of a Branch-and-Bound algorithm maintains:

- A local lower bound (LLB), i.e., a minimum value for any integer feasible solution of a given subproblem.
- A global lower bound (GLB), i.e., a minimum value for any integer feasible solution of the original problem.
- A global upper bound (GUB), i.e., a maximum value for any integer feasible solution of the original problem.

The algorithm can obtain locally valid lower bounds by solving any relaxation of the problem. Typically, computationally easy relaxations are chosen for this part, most notably the LP relaxation. A global lower bound is given by the minimum of all lower bounds in any subproblem. Global upper bounds can be obtained by a problem heuristic and by constructing a feasible solution based on the solution of the relaxation. This latter step is called **exploiting**. We call the currently best known integer feasible solution the **incumbent solution**.

Whenever the local lower bound in any subproblem is not strictly smaller than the global upper bound, we know that this subproblem cannot produce a better solution than the one that we already have. In this case, the subproblem is **fathomed** and can be closed. The same is true if the relaxation is infeasible, i.e., if the feasible region of the subproblem is empty.

A **Branch-and-Cut** algorithm is a Branch-and-Bound algorithm in which the LP relaxation is solved with the cutting plane method. The additional cutting planes can stem from the original problem formulation, i.e., the LP relaxation itself, they can be problem specific cuts or they can be general MIP cuts like the Chvátal-Gomory cuts from the previous subsection. In more detail, a Branch-and-Cut algorithm works as follows. At the root node of the Branch-and-Bound tree, the algorithm runs a heuristic to initialize the global upper bound and builds an initial relaxation of the problem. It then adds the root node to a list of open subproblems.

While the list is not empty, the algorithm selects the next subproblem from the list. The selection is governed by the **selection strategy**. The algorithm then starts by solving the initial LP relaxation of the subproblem If the relaxation is infeasible, the algorithm concludes that no solution exists, marks the subproblem as fathomed and closes it. Otherwise, the algorithm has obtained a local lower bound. It now checks if the value of the incumbent solution is less or equal to this bound. If so, we have proven that the incumbent solution is best possible for the subproblem, and the subproblem can be fathomed. Otherwise, it might happen that the solution obtained from the LP relaxation is integer feasible; if so, we have

found a new integer feasible solution. The algorithm can then fathom the subproblem and possibly update the global upper bound. Otherwise, we run an additional heuristic that tries to build an integer feasible solution from the result of the LP relaxation and possibly update the global lower bound again. Finally, the algorithm tries to add cutting planes to the LP relaxation in the hope that the relaxation will produce a better bound or even an integer solution in the next iteration. If no cutting planes can be found or if the algorithm discovers that adding new cutting planes does not yield a significant improvement (**tailing off**), the algorithm has no choice but to branch, thus creating new subproblems according to a predefined **branching strategy**. The algorithm inserts the new subproblems into the list of open subproblems. A flow chart of the algorithm is depicted in Figure 1.2.

We review some of the algorithmic choices in the algorithm.

Subproblem Selection. The Branch-and-Bound tree is traversed in an order that depends on how the algorithm progresses through the list of open subproblems. Usually, the algorithm can traverse the tree in a breadth-first fashion (which stresses the quick generation of good lower bounds), a depth-first fashion (which stresses the quick generation of feasible solutions) or in *best*-first fashion. In the latter case, the algorithm will heuristically select those subproblems first that are expected to yield good bounds.

Branching Strategy. The easiest way to branch is to **branch by fixing variables**. This strategy first selects branching variable candidates according to a **variable selection rule**, ranks them and then proceeds to branch on the variable x_i of highest rank. In the case of binary variables, this will create one subproblem where x_i is fixed to zero and another subproblem where x_i is fixed to one. If x_i is an integer variable and has a value of x_i^* in the current solution of the LP relaxation, we create one subproblem where the upper bound of x_i is changed to $\lfloor x_i^* \rfloor$ and another subproblem where the lower bound of x_i is changed to $\lfloor x_i^* \rfloor + 1$. **Branching on constraints** is a generalization of this strategy. Suppose that the solution x^* of the current LP relaxation obeys $a^{\mathrm{T}} x^* = b$ for some $a \in \mathbb{Z}^n$ and $b \in \mathbb{R} \setminus \mathbb{Z}$. Then we remove no integer point by creating one subproblem with the additional constraint $a^{\mathrm{T}} x \leq \lfloor b \rfloor$ and another one with the additional constraint $a^{\mathrm{T}} x \geq \lfloor b \rfloor + 1$.

Branching Variable Selection and Ranking. Apart from special cases, it only makes sense to branch on variables that have a fractional value in the solution of the current LP relaxation. An easy way to rank these variables is by *fractionality*, i.e., a variable has a higher rank if the fractional part of its value in the current LP relaxation is close to 0.5. Ties can be broken by ranking variables with large objective function coefficient higher. More sophisticated ranking strategies simulate the outcome of a possible branching by tentatively fixing a variable according to a branching rule and then solving the resulting LP relaxation. In order to decrease the running time of this procedure, the solution algorithm for the LP relaxation is often stopped after a constant number of iterations. The procedure yields two estimate values for every branching variable candidate and proceeds by selecting one of the variables whose estimates are not dominated by the others. We call this process **Strong Branching**.

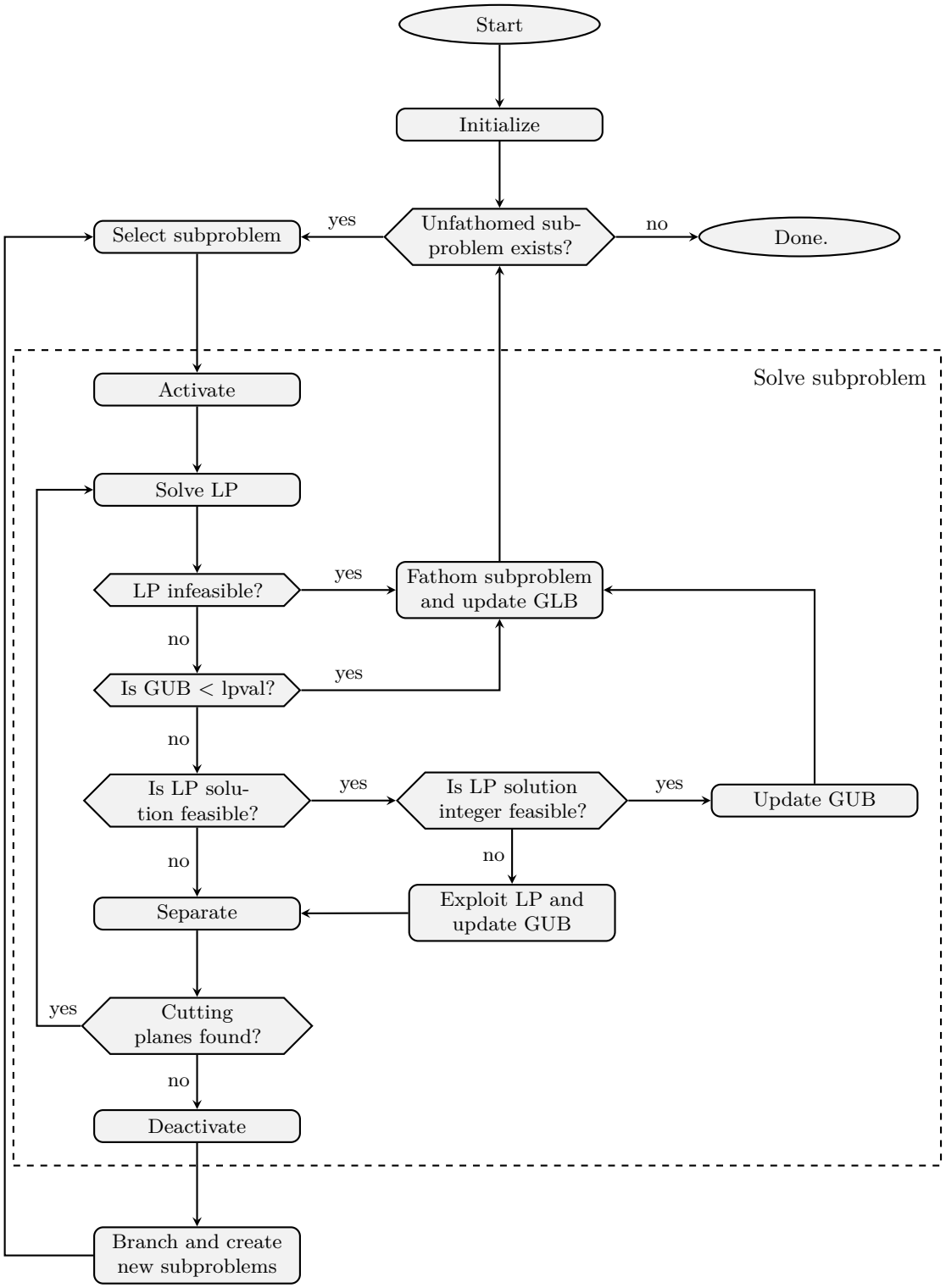


Figure 1.2: A flow-chart of a typical Branch-and-Cut algorithm. The chart is a simplified version of the one in [EGJR01, Figure 9]

Constraint Elimination. In order to keep the LP relaxation small, the cutting plane method can decide to remove constraints that are no longer useful. Typically, a constraint is considered useful as long as it has small slack and thus constraints are removed if they have had large slack for a given number of iterations of the cutting plane method (**aging**).

Tailing off. The Branch-and-Cut algorithm can compare the upper and lower bounds of a subproblem before and after the insertion of cutting planes. If neither value changes significantly after a given number of iterations, it makes sense to stop the cutting plane generation prematurely and to branch instead.

1.4 Graphs and networks

We use the ideas from the previous section to solve network design problems. To that aim, we model networks as finite **graphs**. Following common convention, we denote a graph by the letter G and use the letter V for its set of **nodes**. If G is an undirected graph, we write $\{i, j\}$ to denote an **undirected edge**[1] connecting the nodes i and j. We use the letter E for the multi-set of edges of G. The multi-set E may contain several edges between a given pair of nodes. We call such edges **parallel**. If G is directed, we write (i, j) for an edge (in this case, we also say **arc**) from i to j, stressing that the edge has an orientation. We then denote the multi-set of arcs of G by the letter A. As before, we say that two arcs (i, j) and (k, l) are parallel if $i = k$ and $j = l$ and they are **antiparallel** if $i = l$ and $j = k$. We shall also convert directed graphs into undirected ones by replacing all arcs (i, j) of a directed graph by an edge $\{i, j\}$ and by subsequently removing all parallel edges. We refer to the resulting graph as the **underlying (simple) undirected graph**.

In a directed graph, we define the **reverse** arc of an arc (i, j) as the arc (j, i). The class of **bidirected** graphs is a subclass of the directed graphs. For each of its arcs (i, j), a bidirected graph must contain the reverse arc (j, i). Given an undirected graph $G = (V, E)$, we define the underlying bidirected graph of G as $G^{\leftrightarrow} = (V, \{(i, j), (j, i) \mid \{i, j\} \in E\})$.

Unless otherwise stated, we consider **undirected** graphs in this thesis.

If $\phi : A \to X$ is a function that assigns a value from an arbitrary set X to each arc of a directed graph (V, A), then we write $\phi_{ij} = \phi_{(i,j)} = \phi((i, j))$ to denote the image of (i, j) under ϕ. Likewise, if (V, E) is an undirected graph instead, we write ϕ_{ij} to denote the image of an edge $\{i, j\} \in E$ under $\phi : E \to X$.

1.4.1 Adjacency, Paths and Cycles

If an undirected graph contains an edge $\{i, j\}$, we say that i and j are **adjacent** and that they are **neighbors**. We use the symbol $\delta(i) := \{j \in V \mid \{i, j\} \in E\}$ to denote the **neighborhood** of i. Likewise, we say that two undirected edges $e, f \in E$ are adjacent if they share a common node, i.e., if $e \cap f \neq \emptyset$.

A sequence $(e_1, \ldots, e_k)$ of $k \geq 0$ undirected edges from E is called an **edge progression of length** k in this thesis. If an edge progression P visits an edge $e \in E$, then we write $e \in P$. An edge progression is a **walk** if e_i and e_{i+1} are adjacent in G for all $i = 1, \ldots, k-1$.

[1] We also allow the slightly sloppy notation $\{i, i\}$ to denote a **loop** connecting node i with itself.

In this case, the edge progression can be written as $(\{i_1, i_2\}, \{i_2, i_3\}, \ldots, \{i_k, i_{k+1}\})$ and we can equivalently define it by the sequence $P = (i_1, \ldots, i_{k+1})$ of nodes that it **visits**. If $i_1, \ldots, i_{k+1}$ are pairwise different, we say that P is a **path** from its **start node** i_1 to its **end node** i_{k+1}. We call a walk P a **cycle** if $i_1 = i_{k+1}$ and we say that P is a **simple cycle** if additionally $i_2, \ldots, i_k$ are pairwise different, i.e., a simple cycle is a simple path with $i_1 = i_{k+1}$. A simple cycle that visits all nodes $i \in V$ is a **Hamiltonian cycle**.

We say that i and j are **adjacent** (or **neighbors**) in a directed graph $G = (V, A)$ if G contains an arc (i, j). We use $\delta^{\text{in}}(i) := \{j \in V \mid (j, i) \in A\}$ and $\delta^{\text{out}} := \{j \in V \mid (i, j) \in A\}$ to denote the **incoming** and the **outgoing neighborhood** of node i, respectively. We set $\delta(i) = \delta^{\text{in}}(i) \cup \delta^{\text{out}}(i)$ also in this case.

Analogously to the undirected case, we call two arcs $(i_1, j_1), (i_2, j_2)$ **adjacent** if $j_1 = i_2$ and say that a sequence $(e_1, \ldots, e_k)$ of $k \geq 0$ adjacent arcs is a **directed walk**. Any directed walk has the form $((i_1, i_2), (i_2, i_3), \ldots, (i_k, i_{k+1}))$ and we can identify it by the nodes $P = (i_1, \ldots, i_{k+1})$ that it visits. If no node is contained twice in P, we say that P is a **directed path** and if P starts and ends in the same node, we say that P is a **directed cycle**. We call a directed cycle **simple** if $i_2, \ldots, i_k$ are pairwise different.

1.4.2 Connectivity

Given an undirected graph $G = (V, E)$ and a subset $X \subseteq V$ of its nodes, we say that the graph $G[X] := (X, \{\{i, j\} \in E \mid i, j \in X\})$ is the **induced subgraph** of the set X. We say that an undirected graph $G = (V, E)$ is **connected**, if for any pair $i, j \in V$ of nodes, there is a path from i to j in G. A connected component of G is a maximal subset $X \subseteq V$ such that $G[X]$ is connected. A connected graph that does not contain any cycles is a **tree**. If for any $i \in V$, a graph G is connected, but $G[V \setminus \{i\}]$ is not, then $i \in V$ is a **cut vertex** of G. Furthermore, if G is connected, but removing the edge $\{i, j\}$ makes the remaining graph $(V, E \setminus \{\{i, j\}\})$ disconnected, then $\{i, j\}$ is a **bridge**. We call a connected graph $G = (V, E)$ **biconnected** if $|V| \geq 2$ and G contains no cut vertex. A **biconnected component** of G is a maximal subset X of V such that $G[X]$ is biconnected.

1.5 Cuts and flows in networks

A **network** is a triple $N := (V, E, u)$ consisting of a graph $G := (V, E)$ that has been augmented by a capacity function $u : E \to \mathbb{Z}_{\geq 0}$ for the edges. The underlying graph G can be directed or undirected; we speak of a **directed or undirected network** accordingly.

1.5.1 Cuts

We extend the notion of neighborhoods to sets. In a directed graph $G = (V, A)$, any **cut-set** $S \subseteq V$ induces a **cut**

$$\delta^{\text{out}}(S) := \{(i, j) \in A \mid i \in S \wedge j \in V \setminus S\}.$$

It consists of all the edges that leave S. Likewise, the set of edges that enter S is

$$\delta^{\text{in}}(S) := \{(j, i) \in A \mid i \in S \wedge j \in V \setminus S\}.$$

Observation 1.14. *For any $S \subseteq V$ it holds that $\delta^{out}(S) = \delta^{in}(V \setminus S)$ and vice-versa.*

A **proper cut-set** is a cutset $\emptyset \subsetneq S \subsetneq V$. A proper cut-set S is **strong** if the subgraph $G[S]$ induced by S and the subgraph $G[V \setminus S]$ induced by $V \setminus S$ are both connected. In a directed network $G = (V, A, u)$, we say that a cut induced by $S \subseteq V$ has a weight of $\sum_{(i,j)\in\delta^{\text{out}}(S)} u_{ij}$. The sets $\delta^{\text{in}}(S)$ and $\delta^{\text{out}}(S)$ coincide in an undirected network $N = (V, E, u)$ and any **cut-set** $S \subseteq V$ induces the undirected cut

$$\delta(S) := \{\{i, j\} \in E \mid i \in S \wedge j \in V \setminus S\}.$$

of weight $\sum_{\{i,j\}\in\delta(S)} u_{ij}$.

Observation 1.15. *For any $S \subseteq V$ it holds that $\delta(S) = \delta(V \setminus S)$ and vice-versa.*

There is a one-to-one correspondence between undirected cuts $\delta(S)$ in an undirected network $N = (V, E, u)$ and directed cuts $\delta^{\text{out}}(S)$ in the underlying bi-directed network $N^{\leftrightarrow} = (V, A, \bar{u})$ where $\bar{u}_{ij} = \bar{u}_{ji} = u_{ij}$ for all $\{i, j\} \in E$. Moreover, both cuts have the same weight. We therefore only consider directed cuts in this chapter.

We also consider partial cuts, i.e., collections of arcs between sets $S, T \subseteq V$ that do not necessarily span the entire node set. In a directed graph, we write $(S : T) = \delta^{\text{out}}(S) \cap \delta^{\text{in}}(T)$ to denote the set of arcs that start in S and end in T. If G is an undirected graph, then $(S : T) = \delta(S) \cap \delta(T)$ is the set of edges that have one node in S and one node in T.

The **minimum cut problem** asks for a proper cut-set S of minimum weight in a directed network (V, A, u). It admits a formulation as an *integer* linear program with a variable x_i for all nodes $i \in V$ and a variable y_{ij} for all arcs $(i, j) \in A$.

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in A} u_{ij} y_{ij} && && (1.8)\\
\text{s.t.} \quad & x_i - x_j \le y_{ij} && \text{for all } (i,j) \in A && (1.8a)\\
& \sum_{i\in V} x_i \ge 1 && && (1.8b)\\
& \sum_{i\in V} x_i \le |V| - 1 && && (1.8c)\\
& x_i \in \{0,1\} && \text{for all } i \in V && (1.8d)\\
& y_{ij} \in \{0,1\} && \text{for all } (i,j) \in A && (1.8e)
\end{aligned}$$

The interpretation of the variables is that $x_i = 1$ if and only if $i \in S$; likewise, we have $y_{ij} = 1$ if and only if $(i, j) \in \delta^{\text{out}}(S)$. Constraint (1.8a) ensures that the y variables are set such that $y_{ij} = 1$ if and only if $x_i = 1$ and $x_j = 0$. Constraints (1.8a) and (1.8b) enforce that the set S is a proper cut-set. This formulation depends on our assumption that u is non-negative – if $u_{ij} < 0$ for some arc $(i, j) \notin \delta(S)$, then any optimum solution must set $y_{ij} = 1$, even if $(i, j) \notin \delta(S)$. Moreover, the integrality requirement on the variables is necessary as the constraint matrix is not totally unimodular. Still, the problem can be solved with in polynomial time with an algorithm by Gomory and Hu [GH61] or a simplified version by Gusfield [Gus90].

For two nodes $s, t \in V$, a s-t-cut-set is a cut-set S such that $s \in S$ and $t \in V \setminus S$. The corresponding cut $\delta^{\text{out}}(S)$ is a s-t-cut and the **minimum s-t-cut problem** is the problem

of finding an s-t-cut with minimum weight. In this case, the formulation (1.8) simplifies to the following one with the same interpretation of the variables.

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in A} u_{ij} y_{ij} && && (1.9)\\
\text{s.t.} \quad & x_i - x_j \leq y_{ij} && \text{for all } (i,j) \in A && (1.9a)\\
& x_s = 1 && && (1.9b)\\
& x_t = 0 && && (1.9c)\\
& x_i \in [0,1] && \text{for all } i \in V && (1.9d)\\
& y_{ij} \in [0,1] && \text{for all } (i,j) \in A && (1.9e)
\end{aligned}$$

By forcing that $x_s = 1$ and $x_t = 0$ in the constraints (1.9b) and (1.9c), we can guarantee that $s \in S$ and $t \in V \setminus S$ in any feasible solution. Without constraints (1.8b) and (1.8c), the constraint matrix is now totally unimodular and this is why we could remove the integrality requirement for the variables (formulation (1.9) is a linear program). Still, for the same reason as before, the formulation only works if u is non-negative.

The complexity of both the minimum cut and the minimum s-t-cut problem depends on the choice of the weights. If the weights are non-negative, both problems can be solved in polynomial time while both problems are $\mathcal{NP}$-hard if arbitrary weights are allowed. McCormick, Rao and Rinaldi give a extensive overview over the intermediate cases [MRR03]. In particular, they show how to compute a minimum cut in polynomial time in the *star-negative* case, i.e., if all edges with negative weights are incident to a unique node s. We briefly repeat their demonstration here because the star negative case will be useful in the later chapters where we will use it for undirected graphs.

Theorem 1.16 (McCormick, Rao and Rinaldi [MRR03])**.** *Let $G = (V, E)$ be an undirected graph with a distinct node $s \in V$. If $u \in \mathbb{R}^E$ is such that $u_{ij} \geq 0$ for all $\{i, j\}$ with $i \neq s$, then a minimum cut on (V, E, u) can be computed in polynomial time.*

Proof. If s is the only node of G, the unique cut of cost zero is induced by $S = \{s\}$. We can therefore assume that $|V| \geq 2$ and we let $U = \min\{u_{ij} \mid \{i, j\} \in E \wedge\ u_{ij} < 0\}$ be the smallest negative weight. We can also assume without loss of generality that the edge $\{s, i\}$ is present for all $i \in V \setminus \{s\}$, because if it is not, we can insert it with zero weight. Now, build an auxiliary graph G' by inserting an additional node t along with additional edges $\{i, t\}$ of weight $|U|$ for all $i \in V \setminus \{s\}$. Then, again for all $i \in V \setminus S$, increase the weight of $\{s, i\}$ by $|U|$. Any cut in G' must either cut $\{s, i\}$ or $\{i, t\}$ for all $i \in V \setminus \{s\}$ and therefore, a set $S \subseteq V$ induces a cut of weight C in G if and only if it induces a cut of weight of $C + |U| \cdot (|V| - 2)$ in G'. Thus, it suffices to compute a minimum s-t-cut in G' with respect to the modified, non-negative weights and this can be done in polynomial time. □

1.5.2 Single-Commodity Flows

The dual to minimum s-t-cuts are maximum s-t-flows. In a directed network (V, A, u) with two distinct nodes s and t, a directed **s-t-flow** is a function $f : A \to \mathbb{R}_{\geq 0}$ that satisfies the following two properties.

- **Flow balance.** At each node except s and t, the incoming flow equals the outgoing flow, i.e.,

$$\sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji} = 0 \qquad \text{for all nodes } j \in V \setminus \{s,t\}.$$

- **Capacity compliance.** The flow respects the capacity of each arc, i.e.,

$$f_{ij} \leq u_{ij} \qquad \text{for all arcs } (i,j) \in E$$

Here, the value f_{ij} is the value that f assigns to the arc (i, j). We call s and t the network's **source** and **sink**, respectively, and think of s as a node that is used to feed goods *into* the network while t is used to extract goods *from* the network. For simplicity, we also write (V, A, u, s, t) to quickly specify a network with a source s and a sink t. For any node $i \in V$ we call $\operatorname{ex}_f(i) := \sum_{(i,j)\in A} f_{ji} - \sum_{(j,i)\in A)} f_{ij}$ the **excess** of node i with respect to f. We write $|f| := -\operatorname{ex}_f(s)$ and say that $|f|$ is the **value** of the flow f.

In order to define a flow on an undirected graph $G = (V, E)$ with capacities $u \in \mathbb{R}^E_{\geq 0}$, we replace G the underlying bi-directed graph $G^{\leftrightarrow} = (V, E^{\leftrightarrow})$ and then define a flow f on $G^{\leftrightarrow}$. For each edge $\{i, j\} \in E$ of G, the flow f consists of two values f_{ij} and f_{ji} that define the amount of flow passing over $\{i, j\}$ from i to j and from j to i, respectively. We require that $f_{ij} + f_{ji} \leq u_{ij}$ for all edges $\{i, j\}$ of G and that f satisfies the flow conservation constraints.

The maximum s-t-flow problem is the problem to find a flow f of maximum value $|f|$ on a given directed network (V, A, u, s, t). It has a straight forward formulation as a linear program with an arc flow variable f_{ij} for all arcs $(i, j) \in A$.

$$\begin{aligned} \max \quad & \sum_{(s,j)\in A} f_{sj} - \sum_{(i,s)\in A} f_{it} && && (1.10)\\ \text{s.t.} \quad & \sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji} = 0 && \text{for all } i \in V \setminus \{s,t\} \\ & 0 \leq f_{ij} \leq u_{ij} && \text{for all } (i,j) \in A \end{aligned}$$

This program is the dual of the minimum s-t-cut program (1.9) and therefore, its constraint matrix is totally unimodular. This means that, as long as the capacities u are integer, there always exists an integer maximum s-t-flow. Another fact follows from the duality of the two problems: A maximum s-t-flow has the same value as a minimum s-t-cut. This is Ford and Fulkerson's MaxFlow-MinCut-Theorem.

Theorem 1.17 (MaxFlow-MinCut-Theorem [FF54])**.** *Let $N = (V, A, u)$ be a directed network and let $s, t \in V$. Then the value of a maximum s-t-flow in N is the same as the value of a minimum s-t-cut with respect to u in N.* □

The reason why network flows are often associated with shipping goods is that they can be decomposed into source-sink paths.

Lemma 1.18 (Theorem 8.8 in [KV12]). *Let $N = (V, A, u)$ be a directed network and let f be a s-t-flow on N, $s, t \in V$. Then there is a path-cycle decomposition of f, i.e., a finite set of simple s-t-paths P with a flow assignment $\pi : P \to \mathbb{R}_{>0}$ and a finite set of simple cycles C with a flow assignment $\zeta : C \to \mathbb{R}_{>0}$ such that for all $(i, j) \in A$*

$$f_{ij} \quad = \sum_{p \in P \,:\, (i,j) \in P} \pi(p) \quad + \sum_{c \in C \,:\, (i,j) \in C} \zeta(c)$$

Moreover, there is a maximum s-t-flow on N such that C is empty in any decomposition. □

Using Lemma 1.18 there is a linear program with path flow variables for the maximum s-t-flow problem. Let $\mathcal{P}(s, t)$ be the set of all directed paths from s to t in N.

$$\begin{aligned} \max \quad & \sum_{p \in \mathcal{P}(s,t)} x_p && && (1.11) \\ \text{s.t.} \quad & \sum_{p \in \mathcal{P}(s,t) \,:\, (i,j) \in p} x_p \le u_{ij} && \text{for all } (i,j) \in A \\ & x_p \ge 0 && \text{for all } p \in \mathcal{P}(s,t) \end{aligned}$$

Here, variable x_p models the amount of flow that is assigned to the s-t-path p. As the number of variables in this formulation can be exponential, it needs to be solved by column generation in practice.

There are many combinatorial algorithms for the maximum flow problem, as is seen in the large overview in [AMO93]. Due to the duality of the problems, these algorithms also compute minimum s-t-cuts: Given any maximum s-t-flow f, a residual edge with respect to f is an edge whose capacity is not fully used by f. Using complementary slackness, the set of nodes $S \subseteq V$ that can reach t with a path of residual edges is a minimum cut if and only if f is maximum. The same holds for the set of nodes that can be reached from s with residual edges. Thus, any maximum s-t-flow algorithm can be used to compute a minimum s-t-cut with non-negative weights.

If the goal is not to maximize the flow but rather to ship a prescribed amount of goods, we can generalize the flow conservation constraint. Given a **supply and demand vector** or a **balance vector** $b \in \mathbb{R}^V$ that has an entry b_i for each $i \in V$, a **b-flow** is a function $f : A \to \mathbb{R}_{\ge 0}$ that satisfies properties similar to those of an s-t-flow:

- **Flow balance**

$$\sum_{(i,j) \in A} f_{ij} - \sum_{(j,i) \in A} f_{ji} = b_i \qquad \text{for all nodes } i \in V$$

- **Capacity compliance**

$$f_{ij} \le u_{ij} \qquad \text{for all edges } \{i, j\} \in E$$

If for some $i \in V$ we have $b_i > 0$, then we say that the node i has a **supply** of b_i and that i is a **source**. If on the other hand $b_i < 0$, we say that node i has a **demand** of $|b_i| = -b_i$ and that i is a **sink**. We also call source and sink nodes **terminals** and all other nodes **intermediate nodes**. Lastly, we say that $b \in \mathbb{R}^V$ is **balanced** if $\sum_{i \in V} b_i = 0$. The resulting problem is a feasibility problem: Given a network (V, A, u) and a balance vector $b \in \mathbb{R}^V$, does there exist a b-flow? The following necessary condition is standard.

Lemma 1.19. *Let $N = (V, A, u)$ be a directed network and let $b \in \mathbb{R}^V$. If there exists a b-flow on N, then b is balanced.*

Proof. Let f be a b-flow on N. From the flow balance conditions, we obtain that

$$\sum_{i \in V} b_i = \sum_{i \in V} [\sum_{(i,j) \in A} f_{ij} - \sum_{(j,i) \in A} f_{ji}] = \sum_{(i,j) \in A} f_{ij} - \sum_{(i,j) \in A} f_{ij} = 0$$

□

Thus, if b is not balanced, we know that no b-flow can exist, independently of the arcs of the network and the choice of capacities. If b is balanced, however, we can decide the existence of a b-flow with a single maximum flow computation.

Theorem 1.20 (Section 9.1 in [KV12])**.** *Let $N = (V, A, u)$ be a directed network and let $b \in \mathbb{R}^V$ be balanced. Construct an auxiliary network $N' = (V', A', u')$ by introducing an additional source s^* and an additional sink t^*, connecting all original sources s to s^* with an arc (s^*, s) of capacity b_s and all original sinks t to t^* with an arc (t, t^*) of capacity $-b_t$:*

$$\begin{aligned} V' &:= V \cup \{s^*, t^*\} \\ A' &:= A \cup \{(s^*, i) \mid i \in V : b_i > 0\} \cup \{(i, t^*) \mid i \in V : b_i < 0\} \\ u'_{ij} &:= u_{ij} \quad \text{for all } (i,j) \in A \\ u'_{s^*i} &:= b_i \quad \text{for all } i \in V : b_i > 0 \\ u'_{it^*} &:= -b_i \quad \text{for all } i \in V : b_i < 0 \end{aligned}$$

Then, there exists a b-flow in N if and only if a maximum s^*-t^*-flow in the auxiliary network $N' = (V', A', u')$ has a value of $\sum_{i \in V,\ b_i > 0} b_i$. □

A similar construction can be used to polynomially reduce the maximum s-t-flow problem to a b-flow: Set the flow balance of s and t to an arbitrary value β and $-\beta$, respectively. All other flow balances are set to zero. Then, by doing a binary search on β, the correct value for the maximum flow can be found. In that sense, the two problems are equivalent. We can also characterize the existence of b-flows in terms of cuts.

Lemma 1.21. *Let $N = (V, A, u)$ be a directed network and let $b \in \mathbb{R}^V$. If there exists a b-flow in N, then for all $S \subseteq V$ it holds that*

$$\sum_{(i,j) \in \delta^{out}(S)} u_{ij} \geq \sum_{i \in S} b_i$$

Proof. Let f be a b-flow and let $S \subseteq V$. We obtain from the flow balance conditions that

$$\sum_{i\in S} b_i = \sum_{i\in S} \Big[\sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji}\Big] = \underbrace{\sum_{i\in S} \Big[\sum_{\substack{(i,j)\in A\\ j\in S}} f_{ij} - \sum_{\substack{(j,i)\in A\\ j\in S}} f_{ji}\Big]}_{=0} + \sum_{i\in S} \Big[\sum_{\substack{(i,j)\in A\\ j\in V\setminus S}} f_{ij} - \sum_{\substack{(j,i)\in A\\ j\in V\setminus S}} f_{ji}\Big]$$

$$= \sum_{(i,j)\in\delta^{\text{out}}(S)} f_{ij} - \sum_{(i,j)\in\delta^{\text{in}}(S)} f_{ij} \leq \sum_{(i,j)\in\delta^{\text{out}}(S)} u_{ij}.$$

□

The following theorem is an extension of Ford and Fulkerson's MaxFlow-MinCut-Theorem by Gale.

Theorem 1.22 (Gale; Ford and Fulkerson [Gal57; FF54])**.** *Let $N = (V, A, u)$ be a directed network and let $b \in \mathbb{R}^V$ be balanced. There exists a b-flow on N if and only if for all $S \subseteq V$ it holds that*

$$\sum_{(i,j)\in\delta^{out}(S)} u_{ij} \geq \sum_{i\in S} b_i. \tag{1.12}$$

Proof. Necessity follows from Lemma 1.21. For sufficiency, we claim that in the auxiliary network N' from Theorem 1.20, the set $\{s^*\}$ is a minimum s^*-t^*-cut. To see this, consider some cut-set $\{s^*\} \cup S \subseteq V'$ in N'. The cut induced by $\{s^*\} \cup S$ in N' has a weight of

$$\text{weight}(\{s^*\} \cup S) = \sum_{i\in V\setminus S,\ b_i>0} b_i - \sum_{i\in S,\ b_i<0} b_i + \sum_{(i,j)\in\delta^{\text{out}}(S)} u_{ij}. \tag{1.13}$$

On the other hand, the cut induced by $\{s^*\}$ has a weight of

$$\text{weight}(\{s^*\}) = \sum_{i\in V, b_i>0} b_i = \sum_{i\in S, b_i>0} b_i + \sum_{i\in V\setminus S, b_i>0} b_i. \tag{1.14}$$

Subtracting (1.13) from (1.14) we obtain

$$\begin{aligned}\text{weight}(\{s^*\}) - \text{weight}(\{s^*\} \cup S) &= \sum_{i\in S, b_i>0} b_i + \sum_{i\in S,\ b_i<0} b_i - \sum_{(i,j)\in\delta^{\text{out}}(S)} u_{ij}\\ &= \sum_{i\in S} b_i - \sum_{(i,j)\in\delta^{\text{out}}(S)} u_{ij} \overset{(1.12)}{\leq} 0.\end{aligned}$$

It follows that $\{s^*\}$ is indeed a minimum s^*-t^*-cut cut and therefore, a flow of value $\text{weight}(\{s^*\}) = \sum_{i\in V, b_i>0} b_i$ exists in N'. This is equivalent to the existence of a b-flow by Theorem 1.20. □

We get the analogous theorem for an undirected network $G = (V, E, u)$ by considering (as before) the underlying bidirected network where every edge $\{i, j\} \in E$ is replaced by two arcs (i, j) and (j, i) with shared capacity. This gives us two flow values f_{ij} and f_{ji} for each edge $\{i, j\}$ and f is a valid flow if $f_{ij} + f_{ji} \leq u_{ij}$.

Corollary 1.23 (Gale; Ford and Fulkerson [Gal57; FF54]). *Let $N = (V, E, u)$ be an undirected network and let $b \in \mathbb{R}^V$. There exists a b-flow on N if and only if for all $S \subseteq V$ it holds that*

$$\sum_{\{i,j\}\in\delta(S)} u_{ij} \geq |\sum_{i\in S} b_i|.$$

□

Given a network $N = (V, A, u)$, a balance vector $b \in \mathbb{R}^V$ and a cost vector $c \in \mathbb{R}^A$ the cost of a b-flow f is $\sum_{(i,j)\in A} c_{ij} f_{ij}$. The problem of finding a b-flow with minimum costs (or deciding that none such flow exists) is the **minimum cost flow** problem. Its formulation as a linear program has an arc-flow variable f_{ij} for each arc $(i, j) \in A$ that models the amount of flow on (i, j).

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij} f_{ij} && && (1.15)\\ \text{s.t.} \quad & \sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji} = b_i && \text{for all } i \in V \\ & 0 \leq f_{ij} \leq u_{ij} && \text{for all } (i,j) \in A \end{aligned}$$

While the right-hand sides differ, the constraint matrix of formulation 1.15 is the same as the one for the maximum s-t-flow formulation (1.10) – apart from the fact that it contains a flow balance constraint for *all* nodes. Thus, the matrix is totally unimodular and therefore, if for a given *integral* balance vector $b \in \mathbb{Z}^V$ an optimum b-flow exists, then there also exists an integral one. Notice here that there is no canonical way to write (1.15) with path-flow variables (short of using the transformation in Theorem 1.20) because there is no fixed mapping of sources to sinks.

The minimum cost flow problem is well-studied and, as for the maximum s-t-flow problem, there exist many standard solution algorithms. We refer to [AMO93] for an overview and more details.

1.6 Multi-Commodity Flows

In a standard network flow as defined in the previous section, any source can serve any sink. This is not true in applications where several distinct goods (or, commodities) need to be shipped through the network. In this case, the suitable modeling choice is to use **multi-commodity flows**. Given a directed network $N = (V, A, u)$ and K commodities $C := \{(s^1, t^1, d^1), \ldots, (s^K, t^K, d^K)\}$, a C-multi-commodity flow is a flow that simultaneously sends exactly $d^k \geq 0$ units of flow from s^k to t^k for all $k = 1, \ldots, K$ while respecting the capacities u. The value d^k is called the **demand** of the k-th commodity. Thus, a C-multi-commodity flow is composed of s^k-t^k-flow flows f^k, $k = 1, \ldots, K$, that share the capacities of the network. More formally, a C-multi-commodity flow is a function $f : A \times \{1, \ldots, K\} \to \mathbb{R}_{\geq 0}$ that assigns to each arc in N a flow of commodity $k = 1, \ldots, K$

such that the following linear constraints are satisfied.

$$\sum_{(i,j)\in A} f_{ij}^k - \sum_{(j,i)\in A} f_{ji}^k = \begin{cases} d^k, & \text{if } i = s^k \\ -d^k, & \text{if } i = t^k \\ 0, & \text{otherwise} \end{cases} \qquad \begin{array}{l}\text{for all } k = 1,\dots,K \\ \text{and for all } i \in V\end{array} \tag{1.16}$$

$$\sum_{k=1}^{K} f_{ij}^k \leq u_{ij} \qquad \text{for all } (i,j) \in A$$

$$f_{ij}^k \geq 0 \qquad \begin{array}{l}\text{for all } k = 1,\dots,K \\ \text{and for all } (i,j) \in A\end{array}$$

Rather than specifying the commodities of a multi-commodity flow on a given network $N = (V, A, u)$ as a list $\{(s^k, t^k, d^k)\}_{k=1}^K$ it is sometimes convenient to write them as a matrix $D = (d_{ij})_{i,j\in V} \in \mathbb{R}_{\geq 0}^{V\times V}$ instead. Here, for each pair $i, j \in V, i \neq j$, the entry d_{ij} defines a commodity (i, j, d_{ij}) with source i, sink j and a demand of d_{ij}. We assume that $d_{ii} = 0$ for all $i \in V$. A multi-commodity flow given in this notation is called a multi-commodity D-flow. The **multi-commodity flow problem** is the problem to find a feasible solution for (1.16). In contrast to the single-commodity flow arc-flow formulation (1.10) from the previous section, the constraint matrix of (1.16) is not totally unimodular and for that reason, a feasible multi-commodity flow in a network with integer capacities and demands is not necessarily integer itself (see Figure 1.3). Indeed, Even, Itai and Shamir show that finding a integer multi-commodity flow is $\mathcal{NP}$-hard in both the directed and the undirected case [EIS76]. Their proof holds even if only two commodities are involved and if at the same time the network has unit capacities[2]. The fractional multi-commodity flow problem is polynomially solvable – it suffices to solve formulation (1.16). Still, at the time of the writing no combinatorial algorithm that solves the problem in polynomial time is known to the author.

Formulation (1.16) consists of K arc-flow formulations of the s-t-flow problem that are coupled only by the capacity constraints. We can couple the path-flow formulations for the s-t-flow problem in the same way and obtain a path-flow formulation for the multi-commodity flow problem. As before, we use $\mathcal{P}(s^k, t^k)$ to denote the set of all directed s^k-t^k-paths for $k = 1,\dots,K$.

$$\sum_{p\in\mathcal{P}(s^k,t^k)} x_P^k = d^k \qquad \text{for all } k = 1,\dots,K \tag{1.17}$$

$$\sum_{k=1}^{K} \sum_{\substack{p\in\mathcal{P}(s^k,t^k):\\ (i,j)\in p}} x_p^k \leq u_{ij} \qquad \text{for all } (i,j) \in A$$

$$x_p^k \geq 0 \qquad \begin{array}{l}\text{for all } p \in \mathcal{P}(s^k, t^k) \\ \text{and } k = 1,\dots,K\end{array}$$

[2]Even, Itai and Shamir refer to [Kar75] for the first proof of the $\mathcal{NP}$-hardness of the multi-commodity flow problem. This proof requires as many commodities as there are clauses, however.

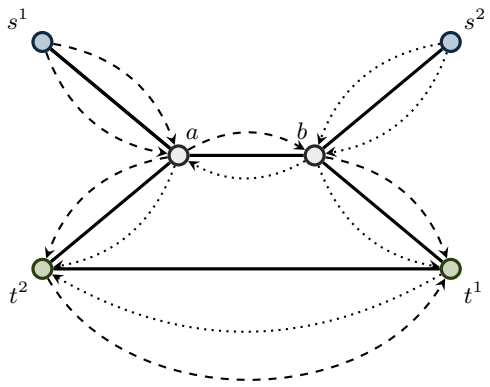


Figure 1.3: An example originating from [AMO93] where no integer multi-commodity flow exists even though all input data is integer. In the example, all edges have a capacity of 1 and both commodities have a demand of 1. A feasible flow is attained by sending half a unit of commodity 1 along both dashed paths and half a unit of commodity 2 along both dotted paths. On the other hand, any flow that sends a full unit of flow on any source sink path disconnects the network. Therefore, no feasible integer flow can exist.

The aim in the following is to derive a commonly-known multi-commodity equivalent of Gale's existence criterion (Theorem 1.22) for single-commodity flows. The first (standard) observation is that the cut-condition from the previous section is still a necessary condition.

Lemma 1.24. *Let $N = (V, A, u)$ be a directed network and let $C := \{(s^k, t^k, d^k)\}_{k=1}^K$ be a set of K commodities. For any set $S \subseteq V$ let $T(S)$ be the set of commodities whose source and sink are separated by the cut $\delta(S)$, i.e., let $T(S) := \{k \mid s_k \in S \wedge t_k \in V \setminus S\}$. If there is a feasible C-multi-commodity flow in N, then*

$$\sum_{(i,j)\in\delta^{out}(S)} u_{ij} \;\geq\; \sum_{k\in T(S)} d^k$$

for all cut-sets $S \subseteq V$. □

For undirected networks, it is well-known that the condition translates to

Lemma 1.25. *Let $N = (V, E, u)$ be an undirected network and let $C := \{(s^k, t^k, d^k)\}_{k=1}^K$ be a set of K commodities. For any set $S \subseteq V$ let $T(S)$ be the set of commodities whose source and sink are separated by the cut $\delta(S)$, i.e., let $T(S) := \{k \mid s_k \in S \wedge t_k \in V \setminus S$ or $s_k \in V \setminus S \wedge t_k \in S\}$. If there is a feasible C-multi-commodity flow, then*

$$\sum_{(i,j)\in\delta(S)} u_{ij} \;\geq\; \sum_{k\in T(S)} d^k$$

for all cut-sets $S \subseteq V$. □

In contrast to single-commodity flows, however, the above condition is not sufficient for the existence of a multi-commodity flow, see Figure 1.4 for an example. Instead, the proper translation of Theorem (1.17) was given by Onaga and Kakusho [OK71] as the so-called

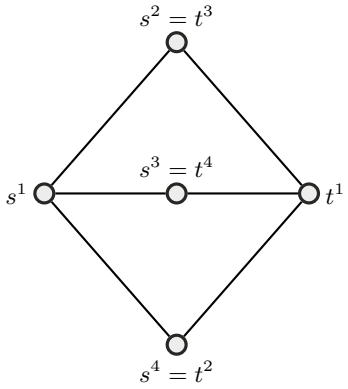


Figure 1.4: A counterexample by Okamura and Seymour [OS81]. All edges in the graph have a capacity of 1 and we are to ship one unit of each commodity. The instance satisfies the cut condition of Lemma 1.25, but no feasible multi-commodity flow exists (at most 3 commodities can be fully sent at the same time).

Japanese Theorem. By Farkas' Lemma[3], the path-flow formulation (1.17) has a feasible solution if and only if

$$\sum_{(i,j)\in A} \mu_{ij} u_{ij} + \sum_{k=1}^{K} \lambda^k d^k \geq 0$$

for all vectors $\mu \in \mathbb{R}^A_{\geq 0}$ and $\lambda \in \mathbb{R}^K$ that satisfy

$$-\sum_{(i,j)\in p} \mu_{ij} \leq \lambda^k$$

for all $p \in \mathcal{P}(s^k, t^k)$ and all $k = 1, \ldots, K$. This is true if and only if the optimum solution value of

$$\min \quad \sum_{(i,j)\in A} \mu_{ij} u_{ij} - \sum_{k=1}^{K} \sigma^k d^k \tag{1.18}$$

$$\text{s.t.} \quad \sum_{(i,j)\in p} \mu_{ij} \geq \sigma^k \qquad \text{for all } k = 1, \ldots, K \text{ and all } p \in \mathcal{P}(s^k, t^k) \tag{1.18a}$$

$$\mu_{ij} \geq 0 \qquad \text{for all } (i,j) \in A \tag{1.18b}$$

is non-negative (substitute σ for $-\lambda$). Now, constraint (1.18a) means that σ^k is at most the length $\text{dist}_\mu(s^k, t^k)$ of a shortest s^k-t^k-path with respect to the edge lengths given by μ. Setting $\sigma^k = \text{dist}_\mu(s^k, t^k)$ for all $k = 1, \ldots, K$ can never decrease the solution value and therefore, there is always an optimum solution where σ^k is exactly the length of a shortest s^k-t^k-path. This is why there is a feasible multi-commodity flow if and only if $\sum_{(i,j)\in A} \mu_{ij} u_{ij} \geq \sum_{k=1}^{K} d^k \cdot \text{dist}_\mu(s^k, t^k)$ for all choices of non-negative edge lengths μ. It turns out that it suffices to check the feasibility condition for all metrics, i.e., for all $\mu \in \mathbb{R}^{V\times V}_{\geq 0}$ such that $\mu_{ih} + \mu_{hj} \geq \mu_{ij}$ for all $i, j, h \in V$.

[3]The full computation in matrix notation is e.g. given in [BCGT98].

Theorem 1.26 (Japanese Theorem [OK71; Iri70])**.** *Let $N = (V, A, u)$ be a network with K commodities $C := \{(s^k, t^k, d^k)\}_{k=1}^K$. Then there is a feasible C-multi-commodity flow in N if and only if*

$$\sum_{(i,j)\in A} \mu_{ij} u_{ij} \geq \sum_{k=1}^{K} d^k \cdot \operatorname{dist}_\mu(s^k, t^k)$$

for all metrics $\mu \in \mathbb{R}_{\geq 0}^{V\times V}$. □

This is a generalization of Gale's existence criterion for b-flows (Theorem 1.22) in the following sense. A **cut-metric** is a metric μ with

$$\mu_{ij} = \begin{cases} 1, & \text{if } i \in S \text{ and } j \in V \setminus S \\ 0, & \text{otherwise} \end{cases}$$

for a cut-set $S \subseteq V$. If a flow has a single commodity, checking the condition of Theorem 1.26 for all cut metrics translates exactly to checking the condition of Theorem 1.22 for all cut-sets $S \subseteq V$. A different interpretation of Gale's Theorem 1.22 is thus that the existence of a single-commodity b-flow can be asserted by checking the condition from the Japanese Theorem for all cut-metrics.

Onaga and Kakusho [OK71] give a combinatorial interpretation of why the condition in the Japanese Theorem is necessary for the existence of a multi-commodity flow. For some weights $\mu \in \mathbb{R}_{\geq 0}^A$, the total amount of weighted capacity in a network is $U := \sum_{(i,j)\in A} \mu_{ij} u_{ij}$. Sending a unit of flow along a path $p \in \mathcal{P}(s^k, t^k)$ consumes $\sum_{(i,j)\in A} \mu_{ij} u_{ij}$ weighted capacity units of the total weighted capacity U. Thus, the capacity consumption is minimum if all flow is sent along shortest paths with respect to the weights μ. Yet, in that case, the flow requires exactly $\sum_{k=1}^K d^k \operatorname{dist}_\mu(s^k, t^k)$ weighted capacity units. This implies that $\sum_{(i,j)\in A} \mu_{ij} u_{ij} \geq \sum_{k=1}^K d^k \operatorname{dist}_\mu(s^k, t^k)$ for all choices of $\mu \in \mathbb{R}_{\geq 0}^A$ is necessary for the existence of a multi-commodity flow.

Chapter 2

Network Design and Robustness

Network design is the task to build a network of links over a set of terminals, allowing them to exchange physical or abstract goods. Typically, the design should minimize the building costs of the network, maximize its potential profit and obey a wide range of side constraints. These constraints can for instance concern the network's connectivity, its capacity, the network's diameter and other topological aspects. This chapter contains a survey over *capacitated* network design problems, i.e., problems where the task is to find capacities for the links such that certain goods can be transported through the network. The focus lies on applications in *communication networks* where data packages make up the network's traffic. As the thesis aims to solve an abstract model rather than aiming for one specific application, our survey will mostly neglect modeling application dependent physical properties and technological details. Still, the traffic in real-world networks is hard to predict. Robust optimization provides the abstract tools that we need to design optimum networks that continue to work when the traffic fluctuates. We review the state-of-the-art of these tools and in particular, of robust capacitated network design. Apart from the literature review, we define the *single-commodity robust network design problem* in this chapter. This problem will be the main subject of the remaining chapters.

2.1 What Robustness Means

According to a text book by Ben-Tal, El Ghaoui and Nemirowski [BTEN09], robust optimization is a "*methodology for handling optimization problems with uncertain data*". We extend this notion of robustness and say that a solution to an optimization problem is robust if it is feasible for a prescribed range of scenarios rather than in a single situation. Two examples illustrate this concept.

- The amount of traffic generated by a client in a network fluctuates on a daily basis. One reason for this fluctuation is that the traffic depends on the clients' activities: Watching a movie online generates much more traffic than reading emails or simple surfing. Likewise, noise and other interferences on the line create transmission errors and force repeated transmissions, thus generating more traffic. This is why, even with sophisticated statistical means, traffic can only be predicted and not be *foretold*. In particular, it can never be known with arbitrary precision and in that sense, the traffic in a network is uncertain – we can only assert that it will be within certain tolerances. Consequently, describing the network's traffic with a fixed number in an optimization model can lead to solutions that are feasible in theory but not in practice.

  In the past, one solution for this problem has been to estimate the true traffic requirements, to design the network and to then add a certain amount of capacity to handle unexpected fluctuations. This approach, however, is expensive and at the same time, it does not give any guarantee that the additional safety capacity will be sufficient. Its rationale is that small fluctuations in the traffic should only induce small changes in the required capacity. Unfortunately, this is not true: Indeed, Ben-Tal, El Ghaoui and Nemirovski [BTEN09] show a practical linear programming instance from the NetLib [netlib] where changing the coefficients of the constraint matrix by 10% makes a previously feasible solution violate the perturbed constraints by 450%. A much better approach is to include the possibility that the real capacity can deviate from its nominal value in the model. Then, every possible deviation defines one scenario that should be considered in the optimization.

- Uncertain traffic is not the only problem in network design. Suppose a network includes a commercial district and a residential area. We can expect that the commercial district will mostly generate traffic during working hours while the residential area's traffic will concentrate on the evening and night hours. To get a feasible solution, we can design a network that can handle the traffic of both areas at the same time, but this solution will be overly expensive. Likewise, we could take an average over the requirements and run the risk of having insufficient capacities for traffic peaks. A more reasonable solution is to allow different scenarios in the model: One where the residential area is the main source of traffic and one where most traffic comes from commercial district. This idea goes back to Gomory and Hu [GH64] who observe that "*Actually, in a communication net problem there is no one set of requirements $R_{p,q}$ but rather a set $R_{p,q}(t)$ varying with a third index, time, that allows for a changing load on the network.*" In this case, it is not the uncertainty that makes it difficult to cast the requirements into a single nominal input, but rather the different network configurations that need to be taken into account.

Here, we consider robust *linear* programs, although the notion exists for different models as well, see for instance [BTN99] and [Soy73] for robust convex and quadratic programs.

2.2 Single-Commodity Robust Network Design

One of the most basic network design problem is the question to connect a given subset of nodes of a graph with a tree. This question is known as the *Steiner tree problem*.

Definition 2.1. *Let $G = (V, E)$ be an undirected graph and let $T \subseteq V$ be a set of nodes called* terminals*. An edge set $X \subseteq E$ induces a **Steiner tree** if the subgraph induced by X is a connected tree that contains all terminals $t \in T$.*

Problem 2.2. *Given an undirected graph $G = (V, E)$, a terminal set $T \subseteq V$ and edge costs $c \in \mathbb{R}^E_{\geq 0}$, the **Steiner tree problem** is the task to find an edge set $X \subseteq E$ that induces a Steiner tree with minimum costs $\sum_{\{i,j\} \in E} c_{ij}$.*

The main subject of this thesis is a robust single-commodity network design model that is a generalization of the Steiner tree problem (as we will see in Chapter 3). It is given by the following definitions.

Definition 2.3. *Let $G = (V, E)$ be an undirected, connected network and let $\mathfrak{B} \subseteq \mathbb{R}^V$ be a polytope. We say that $\mathfrak{B}$ is a (single-commodity) scenario set on G if all vertices of $\mathfrak{B}$ are integer and if $\sum_{i \in V} b_i = 0$ for all $b \in \mathfrak{B}$.*

Problem 2.4. *Let $G = (V, E)$ be a connected, undirected network, let $\mathfrak{B}$ be a single-commodity scenario set on G and let $c \in \mathbb{R}^E_{\geq 0}$. The single-commodity robust network design problem* (`sRND`) *is the task to find integer capacities $u \in \mathbb{Z}^E_{\geq 0}$ with minimal costs $\sum_{\{i,j\} \in E} c_{ij} u_{ij}$ such that there is a b-flow in (V, E, u) for all $b \in \mathfrak{B}$.*

The assumption that G is connected is without loss of generality: If G is not connected, then we can solve the `sRND` problem on the connected components independently. The same is true for the edge costs: If some edge e has strictly negative costs $c_e < 0$, then the problem is unbounded. Also, by Lemma 1.19, the condition that the scenarios in $\mathfrak{B}$ must be balanced is necessary. Still, the problem definition does not specify how the scenario polytope is encoded in the input. While the encoding is irrelevant for the modeling power of the problem, we will see later that it does make a difference in terms of theoretical problem complexity.

Problem 2.5. *A triple $(G, \mathfrak{B}, c)$ consisting of an undirected network $G = (V, E)$, a scenario set $\mathfrak{B} \subseteq \mathbb{R}^V$ on G and a cost function $c \in \mathbb{R}^E_{\geq 0}$ is an instance of the* polyhedral *single-commodity robust network design problem* (`sRND-P`) *if $\mathfrak{B}$ is given in a linear description $\mathfrak{B} = \{b \in \mathbb{R}^V \mid Ab \geq r\}$ with a matrix $A \in \mathbb{R}^{k \times n}$ and a right-hand side $r \in \mathbb{R}^k$, where $n = |V|$ and $k \in \mathbb{N}$ is some arbitrary positive integer.*

Problem 2.6. *A triple $(G, \mathfrak{B}, c)$ consisting of an undirected network $G = (V, E)$, a scenario polytope $\mathfrak{B} \subseteq \mathbb{R}^V$ on G and a cost function $c \in \mathbb{R}^E_{\geq 0}$ is an instance of the* finite *single-commodity robust network design problem* (`sRND-F`) *if $\mathfrak{B}$ is given as the finite list of its $K \in \mathbb{N}$ vertices $\mathfrak{B} = \operatorname{conv}\{b^1, \ldots, b^K\} \subseteq \mathbb{Z}^V$.*

The linear description can be converted into the vertex-based description and vice-versa, however, the conversion requires exponential time and space (see Chapter 1) in general. Yet, since both descriptions encode the same polytope, the resulting instances are otherwise equivalent. Independently of the description, it is sufficient to check feasibility for the vertices of $\mathfrak{B}$.

Observation 2.7. *A capacity vector $u \in \mathbb{Z}^E_{\geq 0}$ is feasible for an instance $(V, E, \mathfrak{B}, c)$ of the* sRND *problem if and only if there exists a b-flow in (G, u) for each vertex b of $\mathfrak{B}$.*

In particular, if $\mathfrak{B}$ is given as a finite list of points $\{b^1, \dots, b^K\}$, it suffices to check feasibility for each b^k, $k = 1, \dots, K$, and there is no need to compute the actual convex hull of the points. This motivates the following notation.

Convention 2.8. *In accordance with the standard network design literature, we write $\mathfrak{B} = \{b^1, \dots, b^K\}$ instead of $\mathfrak{B} = \operatorname{conv}\{b^1, \dots, b^K\}$ to denote a scenario set for an* sRND-F *instance.*

Regardless of the input format, the sRND problem is $\mathcal{NP}$-hard, but in order to see why, we first need to define the multi-commodity variant of the problem.

Definition 2.9. *Let $G = (V, E)$ be an undirected, connected network and let $\mathfrak{D} \subseteq \mathbb{R}^{V \times V}_{\geq 0}$ be a polytope. We say that $\mathfrak{D}$ is a multi-commodity scenario set on G if all vertices of $\mathfrak{D}$ are integer and if $d_{ii} = 0$ for all $i \in V$.*

Definition 2.10. *Let $G = (V, E)$ be a connected, undirected network, let $\mathfrak{D}$ be a multi-commodity scenario set on G and let $c \in \mathbb{R}^E_{\geq 0}$. The multi-commodity robust network design problem* (mRND) *is the task to find integer capacities $u \in \mathbb{Z}^E_{\geq 0}$ with minimal costs $\sum_{\{i,j\} \in E} c_{ij} u_{ij}$ such that there is a multi-commodity D-flow in (G, u) for all $D \in \mathfrak{D}$.*

Analogously to the sRND problem, we define the variants mRND-P and mRND-F of mRND for the cases that the scenario polytope is given as a linear description or a finite list of points, respectively. We observe that our remarks about the relationship of those two encodings remain valid for the mRND problem; this is why we use the same convention regarding vertex descriptions of uncertainty sets.

Sanità [San09] shows that already the *single-source* case of the mRND-F problem is $\mathcal{NP}$-hard. In this special case, we assume that there is a fixed node s such that in all scenarios $(d_{ij})_{i,j \in V} \in \mathfrak{D}$ we have $d_{ij} = 0$ for all $i \in V \setminus \{s\}$ and all $j \in V$. The proof is by a reduction from 3-dimensional matching and requires one scenario per matching dimension.[1]

Theorem 2.11 (Sanità [San09])**.** *The* mRND-F *problem on an undirected graph $G = (V, E)$ is $\mathcal{NP}$-hard even if the scenario set $\mathfrak{D} = \{D^1, D^2, D^3\}$ has only three scenarios, if all demands are 0 or 1 and if a fixed node $s \in V$ is the unique source node for all commodities in all scenarios.* □

To show that the problem is a special case of the single-commodity network design problem, consider a fixed scenario $d \in \mathfrak{D}$ from this special variant of the mRND problem

[1] More recently, Oriolo, Sanità and Zenklusen [OSZ13] showed that mRND-F with unit costs and demands is $\mathcal{NP}$-hard even if $\mathfrak{D} = \{D_1, D_2\}$. However, the proof requires more than one source per scenario.

where all commodities originate from the same source $s \in V$. Then, we can equivalently aggregate all commodities (s, j) with $j \in V$ into a single commodity that has

$$b_i := \begin{cases} \sum_{j \in V} d_{sj}, & \text{if } i = s \\ -d_{sj}, & \text{if } j \in V \setminus \{s\} \end{cases} \tag{2.1}$$

as its balance vector. However, the aggregation destroys the zero-one demand property.

Corollary 2.12 (Buchheim, Liers and Sanità [BLS11])**.** *The* `sRND-F` *problem on an undirected graph $G = (V, E)$ is $\mathcal{NP}$-hard even if the scenario set $\mathfrak{B} = \{b^1, b^2, b^3\}$ has only three scenarios and if a fixed node $s \in V$ is the unique source node in all three scenarios.* □

Corollary 2.13. *The* `sRND` *problem is $\mathcal{NP}$-hard.* □

If we relax the integrality requirement, the `sRND-F` problem is polynomial time solvable (see Chapter 4) with linear programming techniques. This is not true for the polyhedral case. We will see in Chapter 5 that the `sRND-P` problem is $\mathcal{NP}$-hard even if the uncertainty set is (in essence) given by box constraints and even if we do not require integer capacities. It will turn out that even deciding if a vector $u \in \mathbb{R}^E$ is feasible for an `sRND-P` instance $(V, E, \mathfrak{B}, c)$ is co-$\mathcal{NP}$-complete. If there is a single scenario, however, even the integral case can be solved in polynomial time.

Problem 2.14. *Given an undirected graph $G = (V, E)$, a cost function $c \in \mathbb{R}^E_{\geq 0}$ and a balance vector $b \in \mathbb{Z}^V$, the single-commodity network design problem* (`sND`) *is the task to find integer capacities $u \in \mathbb{Z}^E_{\geq 0}$ with minimum costs $\sum_{\{i,j\} \in E} c_{ij} u_{ij}$ such that there is a feasible b-flow in (G, u).*

The problem is polynomial time solvable by computing a minimum cost b-flow f in (G, ∞). This flow then induces an optimum solution $u_{ij} = f_{ij} + f_{ji}$ for all $\{i, j\} \in E$ and this solution is integral because an integral balance vector b implies the existence of an integral b-flow with minimum cost.

Observation 2.15. *The single-commodity network design problem* (`sND`) *is polynomial time solvable.*

Surprisingly, the same is true for the multi-commodity network design problem on an undirected graph $G = (V, E)$ with a single scenario $D \in \mathbb{Z}^{V \times V}_{\geq 0}$ (`mND`), even though the problem requires integer capacities – as long as all demands are integer. This is in contrast to the fact that integer multi-commodity flows with integer demands are $\mathcal{NP}$-hard to compute (see Chapter 1). The reason for this discrepancy is that the capacities are not bounded; we can increase them until all commodities can flow independently of each other. Thus, we can compute an *uncapacitated* integer multi-commodity D-flow f on (G, ∞). This is possible in polynomial time: For each pair $i, j \in V, i \neq j$, we simply send d_{ij} units of flow on a shortest i-j-path[2]. To obtain an optimum integer capacity vector $u \in \mathbb{Z}^E$, we set u_{ij} to the total amount of flow that f sends on $\{i, j\}$ for all $\{i, j\} \in E$.

[2] If some commodity had more than one source and sink, we would need to compute a single-commodity b-flow here, but this flow would still be integer.

Observation 2.16. *The multi-commodity network design problem with a single* integer *scenario* (mND) *is polynomial time solvable.*

The above shortest-path algorithm may produce a fractional solution if some of the demands are fractional and it is not clear how to round the solution in an optimum way. This is because rounding up the capacities on a shortest path for a commodity produces slack and this slack may be used by the routing of another capacity. In this way, the problem essentially becomes an $\mathcal{NP}$-hard knapsack problem.

Theorem 2.17 (Bienstock, Chopra, Günlük and Tsai [BCGT98])**.** *The (integer)* mND *problem with a single scenario is $\mathcal{NP}$-hard.* □

2.3 Non-Robust Capacitated Network Design

Apart from a few special cases (see next chapter), there are no combinatorial algorithms that solve the mRND or the sRND problem exactly. Still, a whole body of literature proves that network design problems can be solved with linear programming methods. We give an overview over the literature after discussing common terminology.

2.3.1 Communication Network Design Terminology

The sRND problem from the previous section is just one possibility to realize the task "find minimum cost capacities for the links in the network such that all traffic requests can be handled." Depending on the application, the model of the links and their capacities may change; as may the meaning of their costs and of the traffic requests.

Orientation of the Links Virtually all communication network design models use graphs to model the network that exists in the real world. They also agree on using the graph's *edges* and *arcs* to model the real-world *links*. However, whether we use edges or arcs to model the links can make a difference. Generally, traffic on a link between to network nodes i and j can travel in two different directions: from i to j or from j to i. If the links allow for both, the situation can be modeled by an *undirected* graph with (undirected) edges or by two anti-parallel arcs in a *bi-directed* graph. Otherwise, a *directed* graph with arcs can be used. The sRND model uses *undirected* links, but it insists that the underlying network flow is *directed*. This is a common modeling choice and describes what we would expect from most real-world networks.

Integer Capacities, Buy-at-Bulk and Facilities In reality, we cannot install arbitrarily small amounts of capacity on a link. Instead, we install integer multiples of a *unit* capacity and model this restriction by requiring the capacities to be integer. As a variation, the *Buy-at-Bulk* model allows only binary decisions: Each link is either bought at the full cost at its full, fixed capacity or not bought at all. This can be modeled with binary decision variables. The Buy-at-Bulk model is generalized by links that consist of several *facilities*, each having a fixed capacity and cost. There, the task is to decide which facility to buy on each link. In some publications, facilities are called *modules*.

Multi-Layer-Networks Real-world networks use several layers of abstraction and network design problems occur on each of them. However, designing the layers independently can produce suboptimal solutions. The k-layer network design model aims to optimize k layers at the same time to resolve this problem.

Unsplittable Flow-Paths In a standard multi-commodity flow with a fixed source-sink assignment, any number of paths may be used for sending flow from some source to some sink. In the unsplittable path model, only a single path may be used for each source-sink pair; the entire demand of that pair must be sent over the unique path.

2.3.2 Non-Robust Formulations for the sND Problem

Using the the arc-flow formulation (1.15) for the minimum cost b-flow problem, it is straight-forward to find a linear programming formulation for the `sND` problem.

$$\begin{aligned} \min \quad & \sum_{\{i,j\}\in E} c_{ij}u_{ij} && && (2.2)\\ \text{s.t.} \quad & \sum_{\{i,j\}\in E}(f_{ij}-f_{ji}) = b_i && \text{for all } i \in V \\ & 0 \le f_{ij}+f_{ji} \le u_{ij} && \text{for all } \{i,j\} \in E \\ & u_{ij} \ge 0 && \text{for all } \{i,j\} \in E \end{aligned}$$

Since the resulting constraint matrix is still totally unimodular, we do not need to enforce integrality of u and the formulation remains solvable in polynomial time.

2.3.3 Non-Robust Formulations for the mND Problem

In the `mND` problem definition, each pair $s,t \in V, s \neq t$ of vertices defines a commodity. By introducing a set of arc-flow variables for every commodity, we can derive the `mND` analogon of the multi-commodity arc-flow formulation (1.16) from Chapter 1. The formulation has two continuous arc-flow variables f_{ij}^{st} and f_{ji}^{st} for each edge $\{i,j\} \in E$ and each commodity $(s,t) \in V \times V$ as well as an integer capacity variable u_{ij} for each edge $\{i,j\} \in E$.

$$\begin{aligned} \min \quad & \sum_{\{i,j\}\in E} c_{ij}u_{ij} && && (2.3)\\ \text{s.t.} \quad & \sum_{\{i,j\}\in E} f_{ij}^{st} - f_{ji}^{st} = \begin{cases} d_{st}, & \text{if } i = s \\ -d_{st}, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases} && \text{for all } i,s,t \in V \\ & \sum_{s,t\in V} f_{ij}^{st} + f_{ji}^{st} \le u_{ij} && \text{for all } \{i,j\} \in E \\ & f_{ij}^{st},\ f_{ji}^{st} \ge 0 && \text{for all } s,t \in V \\ & u_{ij} \in \mathbb{Z}^E_{\ge 0} && \text{for all } \{i,j\} \in E \end{aligned}$$

Here, the f variables are not necessarily integer – even if the u variables are. Hence, if we want integer flows, we must add the integrality requirement for the f variables. The major drawback of the formulation is that it requires $\Theta(|E| \cdot |V|^2)$ variables. By aggregating commodities with the same source, we can reduce the number of variables to $\Theta(|E| \cdot |V|)$, but even then the formulation quickly becomes too large to be solved efficiently, especially on dense graphs where the number of variables has a cubic dependency on the number of nodes.

The Japanese Theorem (Theorem 1.26) yields an alternative formulation with $|E|$ variables (we use one variable u_{ij} for each edge $\{i,j\} \in E$). As before, we define $\text{dist}_\mu(s,t)$ to be the shortest path length from s to t in G with respect to a metric μ.

$$\min \quad \sum_{\{i,j\} \in E} c_{ij} u_{ij} \tag{2.4}$$

$$\text{s.t.} \quad \sum_{\{i,j\} \in E} \mu_{ij} u_{ij} \geq \sum_{s,t \in V} d_{st} \cdot \text{dist}_\mu(s,t) \qquad \text{for all metrics } \mu \in \mathbb{R}^{V \times V}_{\geq 0} \tag{2.4a}$$

$$u_{ij} \in \mathbb{Z}_{\geq 0} \qquad \text{for all } \{i,j\} \in E \tag{2.4b}$$

The constraints (2.4a) are called **metric inequalities** (see [Min81; BCGT98; AMS04]). While there is an infinite number of metrics, it is enough to state the constraints for the extreme rays of the metric cone

$$\mathfrak{M}(G) := \{\mu \in \mathbb{R}^{V \times V}_{\geq 0} \mid \mu_{ij} \leq \mu_{ik} + \mu_{kj} \text{ for all } i,j,k \in V\}.$$

The number of extreme rays of $\mathfrak{M}(G)$ is exponential in general, but fortunately, the proof of the Japanese Theorem directly yields a separation algorithm for the metric inequalities: It suffices to solve the adaptation of the linear program (1.18) for a fixed u^*:

$$\min \quad \sum_{\{i,j\} \in E} \mu_{ij} u^*_{ij} - \sum_{s,t \in V} \sigma^{st} d_{st} \tag{2.5}$$

$$\text{s.t.} \quad \sum_{\{i,j\} \in p} \mu_{ij} \geq \sigma^{st} \qquad \text{for all } s,t \in V \text{ and } p \in \mathcal{P}(s,t) \tag{2.5a}$$

$$\mu_{ij} \geq 0 \qquad \text{for all } \{i,j\} \in E \tag{2.5b}$$

$$\sigma^{st} \in \mathbb{R}_{\geq 0} \qquad \text{for all } s,t \in V \tag{2.5c}$$

If the optimum value of the program is negative, we found a violated metric inequality, otherwise, all metric inequalities are satisfied. Program (2.5) again has an exponential number of constraints, but this time, there is a combinatorial separation algorithm: Given $(\bar{\mu}, \bar{\sigma})$, it suffices to compute a shortest s-t-path p_{st} with respect to the weights defined by $\bar{\mu}$ for all $s,t \in V$. If, for some $s,t \in V$, we find that p_{st} has a length of strictly less than $\bar{\sigma}^{st}$, we have found a violated inequality. Otherwise, all inequalities of type (2.5a) are satisfied.

Alternatively, we can reformulate program (2.5) as

$$\min \quad \sum_{\{i,j\}\in E} \mu_{ij} u^*_{ij} \tag{2.6}$$

$$\text{s.t.} \quad \left.\begin{matrix} \sigma_i^{st} - \sigma_j^{st} \\ \sigma_j^{st} - \sigma_i^{st} \end{matrix}\right\} \leq \mu_{ij} \qquad \text{for all } s,t \in V \text{ and all } \{i,j\} \in E \tag{2.6a}$$

$$\sum_{s,t\in V} \sum_{i\in V} d_{st} \sigma_i^{st} = 1 \tag{2.6b}$$

$$\mu_{ij} \geq 0 \qquad \text{for all } \{i,j\} \in E \tag{2.6c}$$

(see [Gün02; Gün07; Mir00; AMS04]) and solve it without separation. In the program, constraint (2.6b) normalizes the right-hand side of the separated inequality to 1. This is necessary because without this normalization, the program would be unbounded. Consequently, any feasible solution $(\bar{\mu}, \bar{\sigma})$ for (2.6) with an objective value of less than 1 induces a violated metric inequality $\sum_{\{i,j\}\in E} \bar{\mu}_{ij} u^*_{ij} \geq 1$. The technique to derive dual cuts by an application of Farkas' Lemma is known as Benders' decomposition [Ben62].

Branch-and-Bound algorithms that are based on the formulations in this section were for example given by Bienstock, Chopra, Günlük and Tsai [BCGT98], by Günlük [Gün99], by Avella, Mattia and Sassano [AMS04], by Costa, Cordeau and Gendron [CCG09] and by Lee, Lee and Park [LLP13]. Magnanti and Wong [MW84] give a large overview over modeling alternatives and solution algorithms for non-robust network design problems. Minoux [Min89] describes the same technique to derive metric inequalities in an extensive survey.

2.4 Tractable Worst-Case Robustness Models

The `sRND` model is *worst-case robust*: It requires that a robust solution must be feasible in all scenarios. Such a model will produce very *conservative* (i.e., safe) solutions, but this safety comes at a price: An optimum worst-case robust solution can be much more expensive than an optimum solution without robustness. Bertsimas and Sim analyze this *price of robustness* [BS04]. There are, however, situations where this model is appropriate.

- In the application, safety is critical and a failure of the optimized system is not permitted or more expensive than guarding against it. This could, for instance, be true in an energy network: There, locally driving the network over the maximum capacity can result in a global breakdown of the network, triggering costly and time-consuming repairs of vital infrastructure.

- The probability distribution of the scenarios is not known or all scenarios are equally likely.

- Or, as a variation of the previous point: We are certain that all scenarios *will* happen anyway. This is true in our second introductory example where the scenarios model the load changes of a network during the day.

Ben-Tal and Nemirovski [BTN99] describe yet another situation where worst-case robustness is useful by looking at the alternative. If we do not require that a solution is feasible in all scenarios, we accept that some scenarios violate some of the side constraints. In that sense, removing the requirement that a robust solution must work in all scenarios turns the hard side constraints into soft constraints. Worst-case robustness, however, maintains that the side constraints may not be violated under any circumstances. As an example for where hard constraints are crucial, Ben-Tal and Nemirovski propose a processing plant that uses a two-phase process to create a final product. The first phase turns a raw material into a resource needed for the second phase. However, the amount of the resource that comes out of the first phase is subject to uncertainty due to variations in the raw material and inaccuracies in the conversion process. Still, the second phase can never use up more resources than the first phase actually produced. This is a hard constraint – and modeling the situation with a soft constraint would overestimate the output of the plant.

Surprisingly, the outcome of the robustification depends on the problem *formulation*: Applying the same robustification approach with the same uncertainty set to different formulations can yield different models with different optimum solutions and different levels of protection.

2.4.1 Column-Wise Uncertainty: Soyster's Model for Robustness

To the best of the author's knowledge, Soyster [Soy73] was the first to develop a solution approach for general robust linear programs. He asks for an optimum solution of the problem

$$\begin{aligned} \inf \quad & \sum_{i=1}^{n} c_i x_i && && (2.7)\\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i \cdot \mathfrak{K}_i \subseteq \mathfrak{K} && \\ & x_i \geq 0 && \text{for all } i = 1, \ldots, n \end{aligned}$$

where $\mathfrak{K}_1, \ldots, \mathfrak{K}_n$ and $\mathfrak{K}$ are arbitrary convex sets, the multiplication $\alpha \cdot X$ of a scalar $\alpha \in \mathbb{R}$ and a set $X \subseteq \mathbb{R}^m$ multiplies all vectors in X by α and the set addition is the Minkowski sum as defined in Chapter 1. In particular, a non-robust linear program $\min\{c^{\mathrm{T}} x \mid Ax \geq b, x \in \mathbb{R}^n_{\geq 0}\}$ is the special case where $\mathfrak{K}_i = \{A_{i*}\}$ is a singleton containing the i-th column of A for $i = 1, \ldots, n$ and $\mathfrak{K} := \{x \in \mathbb{R}^n_{\geq 0} \mid x \geq b\}$ is a halfspace. The idea of Soyster is to interpret (2.7) as a linear program where the columns of the constraint matrix are not exactly known; rather, the i-th column can be any vector from the set $\mathfrak{K}_i$. Indeed, a vector $x \in \mathbb{R}^n_{\geq 0}$ is feasible for (2.7) if and only if $\sum_{i=1}^n k^i x_i \in \mathfrak{K}$ for all choices of $(k^1, \ldots, k^n) \in \mathfrak{K}_1 \times \cdots \times \mathfrak{K}_n$, i.e., all possible realizations of the columns. In the notation of the introductory examples this means that every choice of columns represents one scenario in which a robust solution of (2.7) must work.

An example from Soyster's article illustrates the idea. Consider the linear program $\min\{c^T x \mid Ax \geq b, x \in \mathbb{R}^n_{\geq 0}\}$ and let $\mathfrak{K}_i = \{a \in \mathbb{R}^m \mid \|a - A_{i*}\|_\infty \leq \varepsilon\}$ for some fixed tolerance $\varepsilon > 0$. Then, $x \in \mathbb{R}^n$ is feasible for (2.7) if and only if x is feasible for the original

linear program and all those variations of it where the coefficients of A have been changed by at most ε. Soyster shows that, more generally, the feasible region of (2.7) is a convex set.

Moreover, if $\mathfrak{K}$ is a half-space $\{x \in \mathbb{R}^n \mid x \geq b\}$, then the feasible region is a polyhedron independently of the choice of $\mathfrak{K}_1, \ldots, \mathfrak{K}_n$. Then an optimum solution for (2.7) can be found by solving the auxiliary linear program $\min\{c^{\mathrm{T}}x \mid \bar{A}x \geq b, x \in \mathbb{R}^n_{\geq 0}\}$. Here, the coefficients of $\bar{A}$ are defined as $\bar{a}_{ij} = \inf\{a \in K_i \mid a_{ij}\}$. These coefficients are well-defined: Soyster observes that we can assume $\bar{a}_{ij} > -\infty$ without loss of generality: If we have $\bar{a}_{ij} = -\infty$ for some i, j, then we know that x_i is zero and we can remove the i-th column from the problem.

2.4.2 Tractable Robust Counterparts by Ben-Tal and Nemirovski

Building on the ideas by Soyster, Ben-Tal and Nemirovski [BTN99] augment a linear program $\min\{c^T x \mid Ax \geq b, x \in \mathbb{R}^n_{\geq 0}\}$ with a convex **uncertainty set** $\mathfrak{U} \subseteq \mathbb{R}^{m \times n}$. They call the resulting problem

$$\begin{aligned} \min \quad & c^{\mathrm{T}}x & & \\ \text{s.t.} \quad & Ax \geq b & & \text{for all } A \in \mathfrak{U} \\ & x \in \mathbb{R}^n_{\geq 0} & & \end{aligned} \tag{2.8}$$

the **robust counterpart** of the original linear program. Likewise, the original problem is the **deterministic formulation** of (2.8). As before, we say that $A \in \mathfrak{U}$ is a **scenario** or a **realization** of the uncertain coefficients. Compared with Soyster's model, this allows dependencies in the rows *and* in the columns of the constraint matrix. In the model, the assumption that only the constraint matrix is uncertain is without loss of generality: If the right-hand side of the system is not known exactly, we can introduce a fixed auxiliary variable and move b to the constraint matrix. Likewise, as replacing $\mathfrak{U}$ by its closed convex hull does not change the problem, we may assume without loss of generality that $\mathfrak{U}$ is convex and closed. As the `sRND` model, this model requires that a solution x is feasible for all scenarios and in fact, Ben-Tal and Nemirovski coined the term worst-case robustness.

The first observation of Ben-Tal and Nemirovski is that the robust counterpart behaves unexpectedly. In some cases, the robust counterpart is infeasible even though for every fixed scenario $A \in \mathfrak{U}$, the problem $\min\{c^{\mathrm{T}}x \mid Ax \geq b, x \in \mathbb{R}_{\geq 0}\}$ *does* have a feasible solution. Phrased differently: Even if we can find a feasible solution $x(A) \in \mathbb{R}^n_{\geq 0}$ for every scenario $A \in \mathfrak{U}$, there is not necessarily a single solution that is feasible in *all* scenarios. Ben-Tal and Nemirovski show that this anomaly has its source in the dependencies of the rows. We say that the uncertainty set $\mathfrak{U} \subseteq \mathbb{R}^{m \times n}$ works **row-wise** if $\mathfrak{U}$ is decomposable into sets $\mathfrak{U}_1, \ldots, \mathfrak{U}_m \subseteq \mathbb{R}^n$ such that the robust counterpart (2.8) may be rewritten as

$$\begin{aligned} \min \quad & c^{\mathrm{T}}x & & \\ \text{s.t.} \quad & u^{\mathrm{T}}x \geq b & & \text{for all } u \in \mathfrak{U}_1 \\ & \vdots & & \vdots \\ & u^{\mathrm{T}}x \geq b & & \text{for all } u \in \mathfrak{U}_m \\ & x \in \mathbb{R}^n_{\geq 0}. & & \end{aligned} \tag{2.9}$$

If we have row-wise uncertainty and the feasible region of $\min\{c^{\mathrm{T}}x \mid Ax \geq b, x \in \mathbb{R}^n_{\geq 0}\}$ is bounded for all $A \in \mathfrak{U}$, then the above anomaly disappears. Moreover, in that case, the value of an optimum solution of the robust counterpart (2.8) is the one of a worst-case scenario.

Theorem 2.18 (Ben-Tal and Nemirovski [BTN99]). *Let $\mathfrak{U} \in \mathbb{R}^{m \times n}$ be a convex, closed uncertainty set that works row-wise and let $b \in \mathbb{R}^m$. If $\{x \in \mathbb{R}^n_{\geq 0} \mid Ax \geq b\}$ is bounded for all $A \in \mathfrak{U}$, then*

1. *the feasible region $\{x \in \mathbb{R}^n_{\geq 0} \mid Ax \geq b \text{ for all } A \in \mathfrak{U}\}$ of the robust counterpart is non-empty if and only if the feasible regions $\{x \in \mathbb{R}^n_{\geq 0} \mid Ax \geq b\}$ are non-empty for all scenarios $A \in \mathfrak{U}$, and*
2. *$\min\{c^{\mathrm{T}}x \mid Ax \geq b,\ x \in \mathbb{R}^n_{\geq 0},\ \text{for all } A \in \mathfrak{U}\} = \sup_{A \in \mathfrak{U}}\{c^{\mathrm{T}}x \mid Ax \geq b, x \in \mathbb{R}^n_{\geq 0}\}$.*

□

Independently of the structure of $\mathfrak{U}$, the feasible region of the robust counterpart is a closed convex set. The Separation Theorem by Grötschel, Lovász and Schrijver [GLS81; GLS84] (see the discussion in Chapter 1) tells us that, in order to optimize over the feasible region of the robust counterpart, we only need a separation algorithm. This is an algorithm that decides whether a given point $x \in \mathbb{R}^n_{\geq 0}$ is feasible for (2.8) and if not, yields a scenario $A \in \mathfrak{U}$ and a row A_{i*} of A such that $A_{i*}^{\mathrm{T}}x < b$. Ben-Tal and Nemirovski argue that a separation oracle for the robust counterpart (2.8) exists if a separation oracle for $\mathfrak{U}$ exists; indeed, to separate x^*, it suffices to solve

$$c_i^* := \min\{A_{i*}^{\mathrm{T}}x^* \mid A \in \mathfrak{U}\} \tag{2.10}$$

for all $i = 1, \ldots, m$. If for some i we have $c_i^* < b_i$, then we found a separating inequality $A_{i*}^{\mathrm{T}}x \geq b_i$. Yet, solving (2.10) is possible in polynomial time if and only if there is a separation oracle for $\mathfrak{U}$ (again due to the Theorem by Grötschel, Lovász and Schrijver).

Tractability Principle 2.19 (Ben-Tal and Nemirovski [BTN99]). *Let $\mathfrak{U} \subseteq \mathbb{R}^{m \times n}$ and let $b \in \mathbb{R}^m$. Then the program*

$$\min\{c^{\mathrm{T}}x \mid Ax \geq b,\ x \in \mathbb{R}^n_{\geq 0},\ \text{for all } A \in \mathfrak{U}\}$$

is solvable in time polynomial in m and n if there is separation algorithm for $\mathfrak{U}$ with a running time polynomial in m and n. □

Thus, we can solve worst-case robust linear programs polynomially if we can separate polynomially over the uncertainty set.

2.4.3 Gamma-Robustness: Bertsimas' and Sim's Less Conservative Model

Bertsimas and Sim [BS03; BS04] propose a model for the case where we do not need to cover entirely different scenarios, but where the input data is inexact. Analogously to the earlier example by Soyster, they assume that the coefficients of the constraint matrix are not known exactly. Each[3] coefficient a_{ij} of the constraint matrix $A \in \mathbb{R}^{m \times n}$ has a nominal

[3] We neglect the case that not all coefficients are inexact.

value $\bar{a}_{ij}$ and can deviate from that value by an amount of $\hat{a}_{ij} > 0$, i.e., it holds that the true coefficient a_{ij} lies in the interval $[\bar{a}_{ij} - \hat{a}_{ij}, \bar{a}_{ij} + \hat{a}_{ij}]$. The central observation of the article is that, in reality, it is highly unlikely that all coefficients attain the maximum deviation at the same time and that therefore, the model by Soyster can produce unnecessarily costly solutions.

In order to control the level of robustness, Bertsimas and Sim introduce a parameter vector $\Gamma \in \mathbb{Z}^m_{\geq 0}$ whose i-th entry Γ_i decides how many coefficients of the i-th constraint may deviate from their nominal value at the same time.[4] A solution is feasible in this model if and only if it is feasible regardless of the amount of deviation, as long as at most Γ_i coefficients of the i-th constraint, $i \in \{1, \dots, m\}$, deviate. Thus, the model is worst-case robust in the sense of Ben-Tal and Nemirovski with the parametrized uncertainty set

$$\mathfrak{U}(\Gamma) := \left\{ (a_{ij})_{i,j=1}^{m,n} \in \mathbb{R}^{m \times n} \;\middle|\; \begin{array}{ll} S_1, \dots, S_m \subseteq \{1, \dots, n\} & \\ \quad |S_i| \leq \Gamma_i \text{ for all } i = 1, \dots, m & \\ \quad a_{ij} \in [\bar{a}_{ij} - \hat{a}_{ij}, \bar{a}_{ij} + \hat{a}_{ij}], & \text{if } j \in S_i \\ \quad a_{ij} = \bar{a}_{ij}, & \text{otherwise} \end{array} \right\}.$$

Accordingly, the Γ-robust counterpart of a program $\min\{c^{\mathrm{T}} x \mid Ax \geq b, x \in \mathbb{R}^n_{\geq 0}\}$ is

$$\min\{c^{\mathrm{T}} x \mid Ax \geq b \text{ for all } A \in \mathfrak{U}(\Gamma),\ x \in \mathbb{R}^n_{\geq 0}\}. \tag{2.11}$$

By increasing Γ_i, we increase the robustness as well as the cost of the solution. In this way, solving the model for different values of Γ allows to find a good trade-off between robustness and cost; the extreme cases being Soyster's model (set $\Gamma_i = n$ for all $i = 1, \dots, m$) or a non-robust model (set $\Gamma_i = 0$ for all $i = 1, \dots, m$). More verbosely, the program (2.11) reads

$$\min \quad \sum_{j=1}^{n} c_j x_j \tag{2.12}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} \bar{a}_{ij} x_j \; - \max_{\substack{S \subseteq \{1,\dots,n\} \\ |S| \leq \Gamma_i}} \sum_{j \in S} \hat{a}_{ij} x_j \geq b_i \qquad \text{for all } i = 1, \dots, m \tag{2.12a}$$

$$x \in \mathbb{R}^n_{\geq 0} \tag{2.12b}$$

This formulation is not (yet) linear, but it is equivalent to the standard description of the robust counterpart in (2.11): If we fix a selection S of deviating coefficients in any constraint i of the robust counterpart (2.11), then this constraint is most restrictive if a_{ij} deviates to the lower bound $\bar{a}_{ij} - \hat{a}_{ij}$ for all $j \in S$, as $x \geq 0$. This is modeled by the reformulated constraint (2.12a).

The first step for linearizing (2.12) is to observe that the maximization problem in constraint (2.12a) is in fact a linear program. For the i-th constraint and a fixed $x^* \in \mathbb{R}^n_{\geq 0}$,

[4] Actually, the model allows to choose a fractional $\Gamma \in \mathbb{R}^m_{\geq 0}$ with the interpretation that $\lfloor \Gamma_i \rfloor$ coefficients of constraint i deviate maximally and a single coefficient a_{ij} of constraint i deviates by the remaining amount $(\Gamma_i - \lfloor \Gamma_i \rfloor)\bar{a}_{ij}$. The extension is straight-forward, yet omitting it makes the exposition significantly easier. For the full model, see [BS04].

the optimum value may be computed as

$$\max \quad \sum_{j=1}^{n} \hat{a}_{ij} x^* z_j \tag{2.13}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} z_j \leq \Gamma_i \tag{2.13a}$$

$$z_j \leq 1 \qquad \text{for all } j = 1, \ldots, n \tag{2.13b}$$

$$z_j \geq 0 \qquad \text{for all } j = 1, \ldots, n \tag{2.13c}$$

In an optimum solution, the z variables sum up to Γ_i and there is always an optimum solution where all z_j are integer. In such a solution, z_j is set to one if and only if a_{ij} is chosen to deviate from the nominal value.

We could now replace the maximization problem in the constraint (2.12a) by system (2.13). However, after the substitution x is not fixed and we would obtain a quadratic program. Instead, Bertsimas and Sim propose to use the dual of (2.13) which will indeed fix this problem. Introducing a variable γ for constraint (2.13a) and a variable ζ_j, $j = 1, \ldots, n$, for the constraints (2.13b), it reads

$$\min \quad \Gamma_i \gamma + \sum_{j=1}^{n} \zeta_j \tag{2.14}$$

$$\text{s.t.} \quad \gamma + \zeta_j \geq \hat{a}_{ij} x_j^* \qquad \text{for all } j = 1, \ldots, n \tag{2.14a}$$

$$\gamma, \zeta \geq 0 \tag{2.14b}$$

We can now replace the maximization problem in the constraints (2.12a) by its dual (2.14).

$$\min \quad \sum_{j=1}^{n} c_j x_j \tag{2.15}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} \bar{a}_{ij} x_j - \Gamma_i \gamma - \sum_{j=1}^{n} \zeta_j \geq b_i \qquad \text{for all } i = 1, \ldots, m \tag{2.15a}$$

$$\gamma + \zeta_j \geq \hat{a}_{ij} x_j^* \qquad \begin{array}{l}\text{for all } i = 1, \ldots, m \\ \text{and all } j = 1, \ldots, n\end{array} \tag{2.15b}$$

$$x_j, \zeta_j \geq 0 \qquad \text{for all } j = 1, \ldots, n \tag{2.15c}$$

$$\gamma \geq 0 \tag{2.15d}$$

Since in constraint (2.15a) the amount subtracted from $\sum_{j=1}^{n} \bar{a}_{ij} x_j$ is minimized anyway, we can omit the explicit minimization. The result is a linear program. We will use this *dualization technique* again in Chapter 5 to solve a similar problem. The same technique works if the original problem (2.11) is a mixed integer problem. This is due to the fact that the inner maximization problem (2.13) remains a continuous (i.e., not integer) linear program also in this case [BS03].

2.5 Worst-Case Robust Capacitated Network Design

Robust network design problems arise by combining the network design formulations from Section 2.3 with the robustification techniques from Section 2.4.

2.5.1 Terminology

Routing Schemes: Static vs. Dynamic Routing

Both the `sRND` and the `mRND` problem are problems that allow a **dynamic routing** scheme: They permit us to route the traffic along different paths in every scenario and the routing is determined implicitly by the underlying network flow. In some applications, however, this approach is not feasible; rather, the network protocol or the network's hardware may require to fix the routing before the realization of the scenarios is known. In that case, we need a **routing template** $\pi : \mathcal{P} \to [0, 1]$, where $\mathcal{P}$ is the set of all paths in our network. The interpretation of the template is that if d_{ij} is the demand of the node pair $i, j \in V$ in some scenario d, then the flow on the i-j-path $p \in \mathcal{P}$ is exactly $\pi(p) \cdot d_{ij}$. We call such a routing scheme **static**. Generally, a static routing is more restricted and thus requires more capacity than a dynamic one. This is why the routing scheme should be taken into account in the optimization, yielding different models for different routing schemes. If the routing template may only contain a single path per commodity, we say that the routing is **unsplittable**. It will turn out that flow formulations are more useful for problems with static routing whereas capacity formulations are well-suited for the dynamic routing case.

Standard Uncertainty Sets for mRND-P

Consider the application of Bertsimas' and Sim's [BS04] approach (see Section 2.4.3) to capacitated network design. We denote by $\bar{d}_{st}$ a nominal value for the demand of each commodity $s, t \in V$ and we suppose that the true demand of the commodity can deviate from its nominal value by at most $\hat{d}_{st}$. Additionally, there can be at most Γ deviations at the same time. We define the resulting uncertainty set as the **Γ-robustness polytope** for the `mRND-P` problem with static or dynamic routing.

$$\mathfrak{G}(\bar{d}, \hat{d}, \Gamma) := \left\{ (d_{st})_{s,t \in V} \in \mathbb{R}^{V \times V}_{\geq 0} \;\middle|\; \begin{array}{ll} d_{st} \in [\bar{d}_{st} - \hat{d}_{st}, \bar{d}_{st} + \hat{d}_{st}], & \text{if } (s,t) \in S \\ d_{st} = \bar{d}_{st}, & \text{otherwise} \\ \text{where } S \subseteq V \times V \text{ with } |S| \leq \Gamma & \end{array} \right\}.$$

Defining $\mathfrak{S}(\Gamma) := \{\sigma \in [0, 1]^{V \times V} \mid \sum_{s,t \in V} \sigma_{st} \leq \Gamma\}$ as the set of possible deviations, we can rewrite the Γ-robustness polytope equivalently as the following set.

$$\mathfrak{G}(\bar{d}, \hat{d}, \Gamma) = (\bar{d}_{st})_{s,t \in V} + \{(\sigma_{st} \hat{d}_{st})_{s,t \in V} \in \mathbb{R}^{V \times V}_{\geq 0} \mid \sigma \in \mathfrak{S}(\Gamma)\}$$

This transformation is due to Koster, Kutschka and Raack [KKR13].

In practice, networks can be so large that measuring a nominal value and a deviation for each *pair* of nodes is not feasible. At the same time, estimating these values reliably is hard because the traffic in the network can be hard to predict [FST97]. This can render

the Γ-robustness approach impractical in larger networks. Likewise, taking even a small number of traffic samples for all node pairs requires many resources and produces a large amount of data. Specifying the scenario set as a finite list can therefore be unrealistic as well [DGG+99].

As an alternative, Fingerhut, Suri and Turner [FST97] and Duffield, Goyal, Greenberg *et al.* [DGG+99] independently define the *Hose* model. In that model, we only assume that we know the maximum *incoming* traffic d_i^{in} and the maximum *outgoing* traffic d_i^{out} at each node $i \in V$. Any traffic matrix that obeys these bounds is a valid scenario. Thus, the number of parameters is linear in the number of nodes (as opposed to the quadratic number in the previous models). Additionally, these parameters are easier to predict [FST97] and can even be known exactly if they stem from technical specifications or legal contracts. We call the resulting uncertainty set

$$\mathfrak{H}_m(d^{in}, d^{out}) := \left\{ (d_{st})_{s,t \in V} \in \mathbb{R}_{\geq 0}^{V \times V} \;\middle|\; \sum_{i \in V} d_{si} \leq d_s^{out} \text{ and } \sum_{i \in V} d_{is} \leq d_s^{in} \text{ for all } s \in V \right\}$$

the (multi-commodity) **Hose polytope**. We speak of the **symmetric Hose polytope** if $d_i^{in} = d_i^{out}$ for all nodes $i \in V$.

The VPN Problem

The combination of `mRND-P` with static routing and the Hose polytope is known as the *virtual private network design* (`VPN`) problem. In the case of a symmetric Hose polytope and unsplittable static routing, there is always an optimum solution where the routing template forms a tree. This was shown by Goyal, Olver and Shepherd [GOS08]. Gupta, Kleinberg, Kumar *et al.* [GKK+01] show that such a solution can be found in polynomial time and thus this special case is polynomial time solvable. On the other hand, Gupta, Kleinberg, Kumar *et al.* also show that finding an optimum tree solution in the case of an *asymmetric* Hose polytope is $\mathcal{NP}$-hard in general, as is deciding if a given capacity allocation allows for an unsplittable static routing.

2.5.2 Formulations for Robust Capacitated Network Design Problems

Dynamic Routing: Flow Formulations

Robust flow formulations for network design problems with dynamic routing require a full set of flow variables for each vertex of the scenario set. This works if the scenario set is given as a finite list; yet even in this case, flow formulations have the drawback of having a large number of variables. If the scenario set is given in its linear description, then its vertices must be determined before a flow formulation can be used. Still, flow formulations have the advantage of being solvable without a separation algorithm.

An Arc-Flow Formulation for sRND [BLS11]. Buchheim, Liers and Sanità [BLS11] propose a robustification of the `sND` arc-flow formulation (2.2) for the `sRND-F` problem. It has an integer capacity variable u_{ij} for each edge $\{i, j\} \in E$ and two continuous arc-flow

variables f_{ij}^q, f_{ji}^q for each edge $\{i,j\} \in E$ and each scenario $q = 1, \ldots, K$.

$$\min \quad \sum_{\{i,j\} \in E} c_{ij} u_{ij} \tag{2.16}$$

$$\begin{aligned}
\text{s.t.} \quad \sum_{\{i,j\} \in E} (f_{ij}^q - f_{ji}^q) &= b_i^q && \text{for all } i \in V \text{ and } q = 1, \ldots, K \\
\max_{q=1,\ldots,K} (f_{ij}^q + f_{ji}^q) &\leq u_{ij} && \text{for all } \{i,j\} \in E \\
\\
f_{ij}^q, f_{ji}^q &\geq 0 && \text{for all } \{i,j\} \in E \text{ and } q = 1, \ldots, K \\
u_{ij} &\in \mathbb{Z}_{\geq 0}^E && \text{for all } \{i,j\} \in E
\end{aligned}$$

The full set of variables for each scenario is necessary because of the flow balance constraints. Buchheim, Liers and Sanità linearize the program.

$$\min \quad \sum_{\{i,j\} \in E} c_{ij} u_{ij} \tag{2.17}$$

$$\begin{aligned}
\text{s.t.} \quad \sum_{\{i,j\} \in E} f_{ij}^q - f_{ji}^q &= b_i^q && \text{for all } i \in V \text{ and } q = 1, \ldots, K \\
f_{ij}^q + f_{ji}^q &\leq u_{ij} && \text{for all } \{i,j\} \in E \text{ and } q = 1, \ldots, K \\
\\
f_{ij}^q, f_{ji}^q &\geq 0 && \text{for all } \{i,j\} \in E \text{ and } q = 1, \ldots, K \\
u_{ij} &\in \mathbb{Z}_{\geq 0}^E && \text{for all } \{i,j\} \in E
\end{aligned}$$

The formulation matches the definition of `sRND-F` exactly; any feasible solution defines a b^q-flow f^q for all scenarios $q = 1, \ldots, K$ along with minimum integer capacities that support the flows. The constraint matrix of formulation (2.17) is not totally unimodular and thus the integrality requirement for the capacity variables is necessary. Given integer values for u, however, we can always find a feasible f that is integer as well, even though integrality of the b^q-flows is not required in the definition of the `sRND-F` problem. Buchheim, Liers and Sanità propose a Branch-and-Cut algorithm for solving the MIP. In order to strengthen the linear programming relaxation of (2.17), they use *target cuts* [BLO08].

An Arc-Flow Formulation for mRND [Min81]. The robustification of `mND` works analogously to the `sRND` case. For the arc-flow formulation, we introduce one set of arc-flow variables for each traffic matrix in $\{D^1, \ldots, D^K\}$ and each commodity $(s,t) \in V \times V$. This

gives us a robustified version of (2.3).

$$\min \quad \sum_{\{i,j\}\in E} c_{ij}u_{ij} \tag{2.18}$$

$$\text{s.t.} \quad \sum_{\{i,j\}\in E} f^q_{ij}(s,t) - f^q_{ji}(s,t) = \begin{cases} d^q_{st}, & \text{if } i = s \\ -d^q_{st}, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases} \quad \begin{array}{l} \text{for all } i,s,t \in V \\ \text{and } q = 1,\dots,K \end{array}$$

$$\sum_{s,t\in V} f^q_{ij}(s,t) + f^q_{ji}(s,t) \le u_{ij} \quad \begin{array}{l} \text{for all } \{i,j\} \in E \\ \text{and } q = 1,\dots,K \end{array}$$

$$f^q_{ij}(s,t),\ f^q_{ji}(s,t) \ge 0 \quad \begin{array}{l} \text{for all } s,t \in V \\ \text{and } q = 1,\dots,K \end{array}$$

$$u_{ij} \in \mathbb{Z}^E_{\ge 0} \quad \text{for all } \{i,j\} \in E$$

The robust formulation has $\Theta(|V|^2 \cdot |E| \cdot K)$ variables and in the same way as (2.3), the formulation is not too practical even when K is small [Min81].

Dynamic Routing: Capacity Formulations

In general, the problem with robustified capacity formulations is that the separation algorithm must take the uncertainty into account as well. This is not a problem if the vertices of the uncertainty set are given in the input because then, the separation can be performed once for each vertex. If the uncertainty set is given as a linear description, however, the resulting separation problems can be hard in theory and in practice. Nonetheless, there is a separation algorithm for `mRND-P` metric inequalities by Mattia [Mat13] in the next section. It resorts to a non-convex quadratic problem. The current literature does not have capacity formulations with robustified traffic requirements for the `sND` problem, but we will see in Chapter 4 how the ideas of this chapter can be used in the single-commodity case and in Chapter 5 we propose a MIP separation for cut-set inequalities for the `sRND-P` problem.

A Capacity Formulation for mRND [Mat12]. If we require in the non-robust capacity formulation (2.4) of `mND` that the capacity must be sufficient in all scenarios, we obtain the following integer linear program.

$$\min \quad \sum_{\{i,j\}\in E} c_{ij}u_{ij} \tag{2.19}$$

$$\text{s.t.} \quad \sum_{\{i,j\}\in E} \mu_{ij}u_{ij} \ge \sum_{s,t\in V} d_{st} \cdot \text{dist}_\mu(s,t) \quad \begin{array}{l} \text{for all metrics } \mu \in \mathbb{R}^{V\times V}_{\ge 0} \\ \text{and all } D = (d_{ij})_{i,j\in V} \in \mathfrak{D} \end{array}$$

$$u_{ij} \in \mathbb{Z}_{\ge 0} \quad \text{for all } \{i,j\} \in E$$

Here, the uncertainty only affects the right-hand sides of the inequalities and we can apply Soyster's [Soy73] approach from the previous section (an uncertain right-hand side is the same as a column with uncertain coefficients). Equivalently, we can observe that a fixed cut is sufficient for all scenarios if and only if it is sufficient for a scenario with maximum demand. Both observations yield the following linear program that has an inequality for every extreme ray of the metric cone.

$$\min \quad \sum_{\{i,j\}\in E} c_{ij}u_{ij} \tag{2.20}$$

$$\text{s.t.} \quad \sum_{\{i,j\}\in E} \mu_{ij}u_{ij} \geq \max_{(d_{ij})_{i,j\in V}\in\mathfrak{D}} \sum_{s,t\in V} d_{st}\cdot \operatorname{dist}_\mu(s,t) \qquad \text{for all metrics } \mu \in \mathbb{R}^{V\times V}_{\geq 0}$$

$$u_{ij} \in \mathbb{Z}_{\geq 0} \qquad \text{for all } \{i,j\} \in E$$

This formulation is an integer linear program and in principle, it can be solved as it is. For a fixed metric μ, the value $\operatorname{dist}_\mu(s,t)$ is a constant and can be computed from the input. Therefore, to find the correct right-hand side for a fixed metric μ, it suffices to optimize the linear function $\sum_{s,t\in V} d_{st}\cdot\operatorname{dist}_\mu(s,t)$ over $\mathfrak{D}$. The full formulation, however, can be too large and in practice, we need a separation algorithm to solve the formulation in reasonable time.

Mattia [Mat10a; Mat10b; Mat12] shows how to modify the separation LP for `mND` metric inequalities to separate metric inequalities in the `mRND-P` case. Assume that we have a linear description of a multi-commodity scenario set

$$\mathfrak{D} = \{D \in \mathbb{R}^{V\times V} \mid A\cdot D \geq r, D \geq 0\}$$

with a matrix $A = (a_{ist})_{i=1,\dots,m;s,t\in V} \in \mathbb{R}^{m,V\times V}$ and a vector $r \in \mathbb{R}^m$ for some $m \geq 0$. Then, we can separate metric inequalities with the following program. It has a variable σ_{st} for each pair $s,t \in V$ and a variable μ_{ij} for each edge $\{i,j\} \in E$. The interpretation of the program is that μ_{ij} defines a metric and σ_{st} denotes the shortest path distance from s to t with respect to the weights defined by μ.

$$\min \quad \sum_{\{i,j\}\in E} \mu_{ij}u^*_{ij} - B \tag{2.21}$$

$$\text{s.t.} \quad \left.\begin{aligned}\sigma_{si} - \sigma_{sj}\\ \sigma_{sj} - \sigma_{si}\end{aligned}\right\} \leq \mu_{ij} \qquad \begin{aligned}&\text{for all } s \in V\\ &\text{and all } \{i,j\} \in E\end{aligned} \tag{2.21a}$$

$$\sum_{\{i,j\}\in E} \mu_{ij} = 1 \tag{2.21b}$$

$$\mu_{ij} \geq 0 \qquad \text{for all } \{i,j\} \in E \tag{2.21c}$$

$$\sigma_{st} \in \mathbb{R} \qquad \text{for all } s,t \in V \tag{2.21d}$$

$$B = \max \quad \sum_{s,t\in V} \sigma_{st}d_{st} \tag{2.21e}$$

$$\sum_{s,t\in V} a_{ist}d_{st} \geq r_i \qquad \text{for all } i = 1,\dots,m \tag{2.21f}$$

$$d_{st} \geq 0 \qquad \text{for all } s,t \in V \tag{2.21g}$$

The formulation has two levels: The outer level has the global objective function that finds a cost-minimum metric for a fixed scenario d_{st}. Here, the constraints (2.21a)–(2.21d) make sure that μ is a (normalized) metric and that σ_{st} is set to the shortest path distance from s to t with respect to μ. We refer to [Mat10b] for a proof. The inner level chooses a worst-case scenario d for a fixed metric μ. The choice is made over $\mathfrak{D}$, as ensured by the constraints (2.21f) and (2.21g). If the optimum objective value of (2.21) is negative, we obtain a metric inequality that is violated by u^*, otherwise, our current solution u^* satisfies all metric inequalities. Unfortunately, the program (2.21) cannot be solved as a standard linear program due to its bi-level structure. The objective function strives to maximize B even without the explicit maximization in (2.21e) and therefore, the two levels can be collapsed into a single one. This is, however, at the expense of linearity as the resulting program has a quadratic, non-convex objective function.

$$
\begin{aligned}
\min \quad & \sum_{\{i,j\}\in E} \mu_{ij} u^*_{ij} - \sum_{s,t\in V} \sigma_{st} d_{st} && && (2.22)\\
\text{s.t.} \quad & \left.\begin{aligned}\sigma_{si} - \sigma_{sj}\\ \sigma_{sj} - \sigma_{si}\end{aligned}\right\} \leq \mu_{ij} && \text{for all } s \in V \\
& && \text{and all } \{i,j\} \in E \\
& \sum_{\{i,j\}\in E} \mu_{ij} = 1 \\
& \sum_{s,t\in V} a_{ist} d_{st} \geq r_i && \text{for all } i = 1,\dots,m \\
& \mu_{ij} \geq 0 && \text{for all } \{i,j\} \in E \\
& d_{st} \geq 0 && \text{for all } s,t \in V \\
& \sigma_{st} \in \mathbb{R} && \text{for all } s,t \in V
\end{aligned}
$$

The alternative is to replace the inner problem (2.21e)–(2.21g) by its dual. Denote by α_i the dual variable corresponding to the constraints (2.21f), for $i = 1,\dots,m$.

$$
\begin{aligned}
\min \quad & \sum_{\{i,j\}\in E} \mu_{ij} u^*_{ij} - B && && (2.23)\\
\text{s.t.} \quad & \left.\begin{aligned}\sigma_{si} - \sigma_{sj}\\ \sigma_{sj} - \sigma_{si}\end{aligned}\right\} \leq \mu_{ij} && \text{for all } s \in V \\
& && \text{and all } \{i,j\} \in E && (2.23a)\\
& \sum_{\{i,j\}\in E} \mu_{ij} = 1 && && (2.23b)\\
& \mu_{ij} \geq 0 && \text{for all } \{i,j\} \in E && (2.23c)\\
& \sigma_{st} \in \mathbb{R} && \text{for all } s,t \in V && (2.23d)\\
& B = \min \quad \sum_{i=1}^{m} r_i \alpha_i && && (2.23e)\\
& \sum_{i=1}^{m} a_{ist} \alpha_i \geq \sigma_{st} && \text{for all } s,t \in V && (2.23f)\\
& \alpha_i \leq 0 && \text{for all } i = 1,\dots,m && (2.23g)
\end{aligned}
$$

In this new formulation, constraint (2.23e) cannot be moved into the objective function as before. Yet, there is a standard technique for bi-level programs that allows us to linearize formulation (2.23): We simply collapse the levels and ensure optimality for the inner level by adding the (quadratic) complementary slackness conditions (see Theorem 1.12). This is the approach that Mattia; Mattia [Mat13; Mat10b] pursues. The resulting program reads as follows.

$$\min \quad \sum_{\{i,j\}\in E} \mu_{ij} u^*_{ij} - \sum_{i=1}^{m} r_i \alpha_i \tag{2.24}$$

$$\text{s.t.} \quad \left.\begin{aligned} \sigma_{si} - \sigma_{sj} \\ \sigma_{sj} - \sigma_{si} \end{aligned}\right\} \leq \mu_{ij} \qquad \begin{aligned} &\text{for all } s \in V \\ &\text{and all } \{i,j\} \in E \end{aligned} \tag{2.24a}$$

$$\sum_{\{i,j\}\in E} \mu_{ij} = 1 \tag{2.24b}$$

$$\sum_{s,t\in V} a_{ist} d_{st} \geq r_i \qquad \text{for all } i = 1,\dots,m \tag{2.24c}$$

$$\sum_{i=1}^{m} a_{ist} \alpha_i \geq \sigma_{st} \qquad \text{for all } s,t \in V \tag{2.24d}$$

$$\Big(\sum_{s,t\in V} a_{ist} d_{st} - r_i \Big) \cdot \alpha_i = 0 \qquad \text{for all } i = 1,\dots,m \tag{2.24e}$$

$$\Big(\sum_{i=1}^{m} a_{ist} \alpha_i - \sigma_s t \Big) \cdot d_{st} = 0 \qquad \text{for all } s,t \in V \tag{2.24f}$$

$$\mu_{ij} \geq 0 \qquad \text{for all } \{i,j\} \in E \tag{2.24g}$$

$$d_{st} \geq 0 \qquad \text{for all } s,t \in V \tag{2.24h}$$

$$\alpha_i \leq 0 \qquad \text{for all } i = 1,\dots,m \tag{2.24i}$$

$$\sigma_{st} \in \mathbb{R} \qquad \text{for all } s,t \in V \tag{2.24j}$$

Here, constraints (2.24c) and (2.24d) guarantee primal and dual feasibility of d and α, respectively. Complementary slackness is ensured by the constraints (2.24e) and (2.24f). Mattia now proposes to linearize the quadratic complementary slackness constraints using additional variables and big-M constraints.[5] She then proceeds to separate metric inequalities with the resulting linear program. The report shows computational results for the case that the uncertainty set is the Hose polytope.

[5] A constraint $a^{\mathrm{T}} x \geq b$ can be disabled conditionally by introducing a binary variable $y \in \{0,1\}$ and rewriting the constraint as $a^{\mathrm{T}} x + M \cdot y \geq b$, where M is a sufficiently large constant. The new constraint is only relevant for the linear program if $y = 0$. Such a constraint is called big-M constraint. Integer linear programs with big-M constraints can be difficult to solve because their linear programming relaxation generally does not provide a good dual bound.

Static Routing

In the static routing case, the flow formulation does not need a set of arc-flow variables for every scenario because we use the same routing in all scenarios. Instead, we need a single set of flow variables to compute a routing template. Here, for all pairs $s, t \in V$ and all edges $\{i, j\} \in E$, the variables f_{ij}^{st} and f_{ji}^{st} denote the fraction of the demand of the commodity (s, t) that is routed via the arcs (i, j) and (j, i), respectively, in each scenario. As before, we use an integer variable u_{ij} to model the capacity of the edge $\{i, j\}$, for all $\{i, j\} \in E$. We have the following linear program [AABP07].

$$\min \quad \sum_{\{i,j\}\in E} c_{ij} u_{ij} \tag{2.25}$$

$$\text{s.t.} \quad \sum_{\{i,j\}\in E} f_{ij}^{st} - f_{ji}^{st} = \begin{cases} 1, & \text{if } i = s \\ -1, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases} \qquad \begin{array}{l}\text{for all } s, t \in V \\ \text{and all } i \in V\end{array} \tag{2.25a}$$

$$\sum_{s,t\in V} d_{st} \cdot (f_{ij}^{st} + f_{ji}^{st}) \leq u_{ij} \qquad \begin{array}{l}\text{for all } \{i, j\} \in E \\ \text{and all } (d_{st})_{s,t\in V} \in \mathfrak{D}\end{array} \tag{2.25b}$$

$$f_{ij}^{st}, f_{ji}^{st} \in [0, 1] \qquad \begin{array}{l}\text{for all } \{i, j\} \in E \\ \text{and all } s, t \in V\end{array} \tag{2.25c}$$

$$u_{ij} \in \mathbb{Z}_{\geq 0} \qquad \text{for all } \{i, j\} \in E \tag{2.25d}$$

Again, it is sufficient to include the constraints (2.25b) for the vertices of $\mathfrak{D}$. A similar formulation can be obtained using a continuous path variable x_P for each s-t-path $P \in \mathcal{P}(s, t)$ and all $s, t \in V$.

$$\min \quad \sum_{\{i,j\}\in E} c_{ij} u_{ij} \tag{2.26}$$

$$\text{s.t.} \quad \sum_{P\in\mathcal{P}(s,t)} x_P = 1 \qquad \text{for all } s, t \in V$$

$$\sum_{s,t\in V} \sum_{\substack{P\in\mathcal{P}(s,t):\\ \{i,j\}\in P}} d_{st} x_P \leq u_{ij} \qquad \begin{array}{l}\text{for all } \{i, j\} \in E \\ \text{and all } (d_{st})_{s,t\in V} \in \mathfrak{D}\end{array}$$

$$x_P \in [0, 1] \qquad \text{for all } P \in \mathcal{P}(s, t) \text{ and all } s, t \in V$$

$$u_{ij} \in \mathbb{Z}_{\geq 0} \qquad \text{for all } \{i, j\} \in E$$

To model unsplittable routing it suffices to turn the arc-flow variables f in program (2.25) (or the path-flow variables x in program (2.26), respectively) into binary variables.

Static Routing and General Polytopes [BAK05]. Ben-Ameur and Kerivin [BAK05] consider the `mRND` problem with static routing, with an uncertainty set $\mathfrak{D}$ that is given in a

linear description and with upper bounds $\bar{u}$ for the capacities of the edges. They decompose the path-formulation (2.26) in the following way: The *master* problem consists of a variant of the path formulation (2.26), however, it maintains a set $\overline{\mathcal{P}}(s,t) \subseteq \mathcal{P}(s,t)$ of relevant paths for each pair of nodes $s,t \in V$ as well as a set of relevant scenarios $\overline{\mathfrak{D}} \subseteq \mathfrak{D}$. Both sets are initially empty. In more detail, the master problem is the following.

$$\min \quad \sum_{\{i,j\}\in E} c_{ij}u_{ij} \tag{2.27}$$

$$\text{s.t.} \quad \sum_{p\in\mathcal{P}(s,t)} x_p \geq 1 \qquad \text{for all } s,t \in V \tag{2.27a}$$

$$\sum_{\substack{p\in\overline{\mathcal{P}}(s,t):\\ \{i,j\}\in p}} x_p \leq f_{ij}^{st} \qquad \text{for all } s,t \in V \text{ and all } \{i,j\} \in E \tag{2.27b}$$

$$\sum_{s,t\in V} f_{ij}^{st} d_{st} \leq u_{ij} \qquad \text{for all } \{i,j\} \in E \text{ and all } (d_{st})_{s,t\in V} \in \overline{\mathfrak{D}} \tag{2.27c}$$

$$u_{ij} \leq \bar{u}_{ij} \qquad \text{for all } \{i,j\} \in E \tag{2.27d}$$

$$x_p \in [0,1] \qquad \text{for all } s,t \in V \text{ and all } p \in \overline{\mathcal{P}}(s,t) \tag{2.27e}$$

$$f_{ij}^{st} \in [0,1] \qquad \text{for all } s,t \in V \text{ and all } \{i,j\} \in E \tag{2.27f}$$

The master problem (2.27) is bounded. Suppose for the moment that it is feasible as well and let u^* be an optimum solution for the problem. In order to guarantee that u^* is globally optimum, we need to make sure that adding additional paths to $\overline{\mathcal{P}} := \bigcup_{s,t\in V} \overline{\mathcal{P}}(s,t)$ cannot improve the value of u^*. Moreover, u^* must be globally feasible, i.e., the capacities must be sufficient to route all scenarios in $\mathfrak{D}$ (and not only those in $\overline{\mathfrak{D}}$). For the former problem, Ben-Ameur and Kerivin solve a *path satellite problem*. It consists of computing a shortest path between all pairs $s,t \in V$ with respect to the dual variables π and ρ of the constraints (2.27a) and (2.27b). They argue that if $\sum_{\{i,j\}\in p} \rho_{ij}^{st} < \pi_{st}$ for some s-t-path $p \notin \overline{\mathcal{P}}$, then p will improve the current solution u^*. In this case, we add the path p to $\overline{\mathcal{P}}$. To ensure global feasibility on the other hand, Ben-Ameur and Kerivin separate inequalities of type (2.27c) in a *demand satellite* problem. Given fixed routing variables f^* and fixed capacities u^* from an optimum solution of (2.27), it suffices to solve the linear program

$$\begin{aligned} u_{ij}^{\max} := \max \quad & \sum_{s,t\in V} f_{ij}^{*,st} d_{st} \\ \text{s.t.} \quad & (d_{st})_{s,t\in V} \in \mathfrak{D} \end{aligned} \tag{2.28}$$

for all edges $\{i,j\} \in E$. Notice that here, we optimize over the entire scenario set. If for some edge $\{i,j\} \in E$ we find that $u_{ij}^{\max} > u_{ij}^*$, then the inequality

$$\sum_{s,t\in V} f_{ij}^{st} d_{st} \leq u_{ij} \tag{2.29}$$

is violated by (f^*, u^*). We add the corresponding optimum solution d^* of (2.28) to $\overline{\mathfrak{D}}$, thus adding the violated inequality (2.29) to the master problem.

To solve the master problem to global optimality, it now suffices to iteratively call the path satellite, the demand satellite and the master problem itself until neither new paths nor new scenarios are found. If at some point during the computation the master problem becomes infeasible, we call the path satellite and if no improving paths can be found, then the problem instance must be globally infeasible (i.e., the upper bounds for the capacities are too restrictive to route all scenarios in $\mathfrak{D}$).

Static Routing and the Hose Polytope [AABP07]. Altın, Amaldi, Belotti and Pınar [AABP07] apply the dualization technique by Bertsimas and Sim [BS03] (see Section 2.4.3) to the arc-flow formulation (2.25). Assuming that the underlying uncertainty set is the Hose polytope, they equivalently replace constraint (2.25b) by the optimization

$$
\begin{aligned}
u_{ij} = \max \quad & d_{st}(f_{ij}^{st} + f_{ji}^{st}) && && \text{(2.30a)}\\
\text{s.t.} \quad & \sum_{i \in V} d_{si} \leq d_s^{out} && \text{for all } s \in V && \text{(2.30b)}\\
& \sum_{i \in V} d_{is} \leq d_s^{in} && \text{for all } s \in V && \text{(2.30c)}\\
& d_{st} \geq 0 && \text{for all } s,t \in V && \text{(2.30d)}
\end{aligned}
$$

for all $\{i,j\} \in E$. For fixed f, this gives us a bounded, feasible linear program for each edge $\{i,j\} \in E$. Following the technique by Bertsimas and Sim, we replace these programs by their dual. In the linear program for edge $\{i,j\}$, denote by ω^{ij} and υ^{ij} the dual variables corresponding to constraint (2.30b) and (2.30c), respectively. This yields the following dual for each edge $\{i,j\}$.

$$
\begin{aligned}
\min \quad & \sum_{s \in V} d_s^{out}\omega_s^{ij} + d_s^{in}\upsilon_s^{ij} && && \text{(2.31)}\\
\text{s.t.} \quad & \omega_s^{ij} + \upsilon_t^{ij} \geq f_{ij}^{st} + f_{ji}^{st} && \text{for all } s,t \in V\\
& \omega_s^{ij}, \upsilon_s^{ij} \geq 0 && \text{for all } s \in V
\end{aligned}
$$

This program is linear even for a non-fixed f. Altın, Amaldi, Belotti and Pınar insert it into formulation (2.25), replacing constraint (2.25b).

$$\min \quad \sum_{\{i,j\}\in E} c_{ij}u_{ij} \tag{2.32}$$

$$\text{s.t.} \quad \sum_{\{i,j\}\in E} f_{ij}^{st} - f_{ji}^{st} = \begin{cases} 1, & \text{if } i = s \\ -1, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases} \quad \begin{array}{l}\text{for all } s,t \in V \\ \text{and all } i \in V\end{array} \tag{2.32a}$$

$$\sum_{s\in V} d_s^{out}\omega_s^{ij} + d_s^{in}\upsilon_s^{ij} \leq u_{ij} \quad \text{for all } \{i,j\} \in E \tag{2.32b}$$

$$\omega_s^{ij} + \upsilon_t^{ij} \geq f_{ij}^{st} + f_{ji}^{st} \quad \text{for all } s,t \in V \tag{2.32c}$$

$$\omega_s^{ij}, \upsilon_s^{ij} \geq 0 \quad \text{for all } s \in V \tag{2.32d}$$

$$f_{ij}^{st}, f_{ji}^{st} \in [0,1] \quad \begin{array}{l}\text{for all } \{i,j\} \in E \\ \text{and all } s,t \in V\end{array} \tag{2.32e}$$

$$u_{ij} \in \mathbb{Z}_{\geq 0} \quad \text{for all } \{i,j\} \in E \tag{2.32f}$$

As the objective function makes sure that the left-hand side of constraint (2.32b) is minimized, we can omit the explicit minimization without inserting complementary slackness conditions. In this way, we directly obtain a single level mixed integer linear program and do not need any further linearization. Solving the program gives us minimum cost integer capacities for `mRND` with static routing over the Hose polytope. The program has $\Theta(|V|^2|E|)$ variables and $\Theta(|V|^3 + |V|^2|E|)$ constraints. A similar algorithm for a problem variant with multiple facilities was given by Altın, Yaman and Pınar [AYP11].

Static Routing and Gamma-Robustness [KKR13]. Koster, Kutschka and Raack [KKR13] robustify the arc-flow formulation (2.25) with the Γ-robustness model.

$$\min \quad \sum_{\{i,j\}\in E} c_{ij}u_{ij} \tag{2.33}$$

$$\text{s.t.} \quad \sum_{\{i,j\}\in E} f_{ij}^{st} - f_{ji}^{st} = \begin{cases} 1, & \text{if } i = s \\ -1, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases} \quad \begin{array}{l}\text{for all } s,t \in V \\ \text{and all } i \in V\end{array} \tag{2.33a}$$

$$\sum_{s,t\in V} \bar{d}_{st}(f_{ij}^{st} + f_{ji}^{st}) + \max_{\sigma\in\mathfrak{S}(\Gamma)} \sum_{s,t\in V} \sigma_{st}\hat{d}_{st}(f_{ij}^{st} + f_{ji}^{st}) \leq u_{ij} \quad \text{for all } \{i,j\} \in E \tag{2.33b}$$

$$f_{ij}^{st}, f_{ji}^{st} \in [0,1] \quad \begin{array}{l}\text{for all } \{i,j\} \in E \\ \text{and all } s,t \in V\end{array} \tag{2.33c}$$

$$u_{ij} \in \mathbb{Z}_{\geq 0} \quad \text{for all } \{i,j\} \in E \tag{2.33d}$$

They solve the model with both the dualization technique by Bertsimas and Sim [BS03] and the separation approach by Ben-Ameur and Kerivin [BAK05] and computationally

compare the resulting algorithms.

For the dualization approach, Koster, Kutschka and Raack replace the optimization problem in constraint (2.33b) by its dual

$$\begin{aligned} \min \quad & \gamma^{ij} \cdot \Gamma + \sum_{s,t \in V} \tau_{st}^{ij} && && (2.34)\\ \text{s.t.} \quad & \gamma^{ij} + \tau_{st}^{ij} \geq \hat{d}_{st}(f_{ij}^{st} + f_{ji}^{st}) && \text{for all } s,t \in V \\ & \tau_{st}^{ij} \geq 0 && \text{for all } s,t \in V \\ & \gamma^{ij} \geq 0 \end{aligned}$$

where $\{i,j\}$ is a fixed edge. The result is a compact linear program with $\Theta(|V|^2|E|)$ variables and $\Theta(|V|^3 + |V|^2|E|)$ constraints.

$$\begin{aligned} \min \quad & \sum_{\{i,j\} \in E} c_{ij} u_{ij} && && (2.35)\\ \text{s.t.} \quad & \sum_{\{i,j\} \in E} f_{ij}^{st} - f_{ji}^{st} = \begin{cases} 1, & \text{if } i = s \\ -1, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases} && \text{for all } s,t \in V \text{ and all } i \in V \\ & \sum_{s,t \in V} \bar{d}_{st}(f_{ij}^{st} + f_{ji}^{st}) + \gamma^{ij} \cdot \Gamma + \sum_{s,t \in V} \tau_{st}^{ij} \leq u_{ij} && \text{for all } \{i,j\} \in E \\ & \gamma^{ij} + \tau_{st}^{ij} - \hat{d}_{st}(f_{ij}^{st} + f_{ji}^{st}) \geq 0 && \text{for all } s,t \in V \text{ and all } \{i,j\} \in V \\ & \tau_{st}^{ij} \geq 0 && \text{for all } s,t \in V \text{ and all } \{i,j\} \in E \\ & \gamma^{ij} \geq 0 && \text{for all } \{i,j\} \in E \\ & f_{ij}^{st}, f_{ji}^{st} \in [0,1] && \text{for all } s,t \in V \text{ and all } \{i,j\} \in E \\ & u_{ij} \in \mathbb{Z}_{\geq 0} && \text{for all } \{i,j\} \in E \end{aligned}$$

The problem is then solved with a Branch-and-Cut algorithm.

To apply the alternative approach by Ben-Ameur and Kerivin [BAK05], Koster, Kutschka and Raack need to solve the separation problem for the constraints (2.33b). This is the corresponding problem to the demand satellite in [BAK05]. As we have an *arc*-flow formulation here, there is no need to solve a path satellite problem, however. Given fixed f^* and u^*, the separation problem for the constraints (2.33b) is to find a deviation $\sigma \in \mathfrak{S}(\Gamma)$ and an edge $\{i,j\} \in E$ such that

$$\sum_{s,t \in V} \bar{d}_{st}(f_{ij}^{*,st} + f_{ji}^{*,st}) + \sum_{s,t \in V} \sigma_{st} \hat{d}_{st}(f_{ij}^{*,st} + f_{ji}^{*,st}) > u_{ij}^*$$

or to decide that none such combination of a deviation and an edge exists. The problem can be solved separately for each fixed edge $\{i, j\}$. Then, it amounts to solving

$$\max_{\sigma \in \mathfrak{S}(\Gamma)} \sum_{s,t \in V} \sigma_{st} \hat{d}_{st} (f_{ij}^{*,st} + f_{ji}^{*,st}). \tag{2.36}$$

If the optimum value of (2.36) is larger than $u_{ij}^* - \sum_{s,t \in V} \bar{d}_{st} (f_{ij}^{*,st} + f_{ji}^{*,st})$, then we found a violated inequality; otherwise, no violated inequality involving the edge $\{i, j\}$ exists. Koster, Kutschka and Raack observe that to solve (2.36), we can simply sort the values $\hat{d}_{st}(f_{ij}^{*,st} + f_{ji}^{*,st})$ in decreasing order. Then, the first Γ commodities in the order determine a worst-case deviation. This approach yields a program that initially has $\Theta(|V|^3)$ constraints and $\Theta(|V|^2|E|)$ variables. To solve the linear programming relaxation of the problem, we need to solve $\Theta(|E|)$ separation problems per iteration of the separation algorithm.

2.6 More Valid Inequalities

2.6.1 More General Metric Inequalities for the mND

If the left-hand side of a metric inequality only has integer coefficients, then the right-hand side of the inequality can be rounded up. The result is a *rounded* metric inequality, as defined by [BCGT98],

$$\sum_{\{i,j\} \in E} \mu_{ij} u_{ij} \geq \left\lceil \sum_{s,t \in V} d_{st} \cdot \mathrm{dist}_{\mu}(s, t) \right\rceil.$$

Rounded metric inequalities are only useful if there are fractional scenarios.

Avella, Mattia and Sassano [AMS04] improve the separation of metric inequalities with a slight modification of the separation LP (2.6). The resulting *strong* metric inequalities are guaranteed to have a sparse left-hand side. Additionally, they introduce a MIP separation for $\{0, 1\}$-rounded metric inequalities, i.e., another strengthened version of the previous metric inequalities. As a third separation method, they show a shrinking procedure that heuristically tries to turn any separated inequality on an `mND` instance (V, E, D) into a *tight* metric inequality of the form

$$\sum_{\{i,j\} \in E} \mu_{ij} u_{ij} \geq \mathrm{OPT}(V, E, D, \mu),$$

where μ is an arbitrary metric and $\mathrm{OPT}(V, E, D, \mu)$ is the optimum value of the same `mND` instance (V, E, D) with edge costs defined by μ. The resulting inequality is not necessarily a tight metric inequality, but it is guaranteed to have the same violation as the corresponding tight metric inequality on the shrunken graph. By studying the convex hull of all integer feasible solutions

$$\mathfrak{P}_{\texttt{mND}}(V, E, D) := \mathrm{conv}\{u \in \mathbb{Z}_{\geq 0}^E \mid u \text{ is feasible for the } \texttt{mND} \text{ instance } (V, E, D)\}$$

of a `mND` instance (V, E, D) they show that the tight metric inequalities completely characterize $\mathfrak{P}_{\texttt{mND}}$. In particular, all valid inequalities for $\mathfrak{P}_{\texttt{mND}}$ are dominated by a tight metric inequality.

2.6.2 Cut-Set Inequalities for Various Network Design Problems

Cut-set inequalities are a well-known concept in network design; we discuss examples and the literature in the sequel. The general idea is to fix a cut-set S, i.e., a partitioning of the node set into two shores such that there are terminals on either side of the induced cut. Since all connections between the separated terminals must cross from S to $V \setminus S$ at some point, we can derive a lower bound on the minimum capacity of the cut induced by S in this way. Cut-set inequalities exist for several network design problems, including some special cases of `sRND`. In many cases, these inequalities are facet-inducing.

Steiner Tree

Consider the Steiner tree problem 2.2 from Chapters 2 and 3, i.e., the task to connect all terminals $T \subseteq V$ of a connected, undirected graph $G = (V, E)$ with a (minimum cost) subtree. To obtain a cut-set inequality, let $S \subseteq V$ be a cut-set and assume that there are two terminals $s \in S$ and $t \in V \setminus S$ that are separated by S. In order to connect s and t, any feasible Steiner tree must contain at least one edge out of the cut $e \in \delta(S)$. Chopra and Rao [CR94] show that this idea yields a Steiner tree formulation as an integer linear program.

$$\min \quad \sum_{\{i,j\}\in E} c_{ij} x_{ij} \tag{2.37}$$

$$\text{s.t.} \quad \sum_{\{i,j\}\in\delta(S)} u_{ij} \geq 1 \qquad \text{for all } S \subseteq V \text{ with } \emptyset \neq S \cap T \neq T \tag{2.37a}$$

$$u_{ij} \in \{0,1\} \qquad \text{for all } \{i,j\} \in E \tag{2.37b}$$

The constraints (2.37a) are called *Steiner cut-set inequalities*. Given that S separates at least two terminals, they are facet-inducing if and only if S is a strong[6] cut-set [CR94].

Single-Commodity Network Design with a Single Source and Binary Demands

As discussed in Section 2.2, the `mRND` problem is a special case of the `sRND` problem if there is a unique source s that is shared by all commodities. Cut-set inequalities also exist in this setting. Suppose that we have a finite set of scenarios $\mathfrak{D} = \{D^1, \ldots, D^K\} \subseteq \{0,1\}^{V\times V}$ and that these scenarios are binary and use a unique source $s \in V$. Now, denote by $T^q := \{j \in V \mid d_{sj} > 0\}$ the set of sinks of scenario $(d^q_{ij})_{i,j\in V} \in \mathfrak{D}$ and fix some $S \subseteq V \setminus \{s\}$ and a scenario q. Then, we know that each sink in $S \cap T^q$ must be connected to the source s with a capacity of 1. All these connections must use an edge in $\delta(S)$. Thus, for all $S \subseteq V \setminus \{s\}$ we get the following valid inequality for $\operatorname{conv}\{u \in \mathbb{Z}^E_{\geq 0} \mid u \text{ is feasible for } (G, \mathfrak{D})\}$, see [San09; OSZ13].

$$\sum_{\{i,j\}\in\delta(S)} u_{ij} \geq \max_{q=1,\ldots,K} |S \cap T^q| \tag{2.38}$$

[6] As before, we say that a cut-set $\emptyset \subsetneq S \subsetneq V$ is strong if both the induced subgraph $G[S]$ and its complement $G[V \setminus S]$ are connected graphs.

To stress the connection to the general cut-set inequalities that we will use later in this chapter, we observe that we can equivalently rewrite (2.38) as

$$\sum_{\{i,j\}\in\delta(S)} u_{ij} \geq |\sum_{i\in S} b_i|$$

using the definition of b in (2.1).

Multi-Commodity Network Design

Even though they are not sufficient to obtain a mND LP formulation, inequalities of the form

$$\sum_{\{i,j\}\in\delta(S)} u_{ij} \geq \Big\lceil \sum_{i\in S}\sum_{j\in V\setminus S} d_{ij} + \sum_{i\in V\setminus S}\sum_{j\in S} d_{ij} \Big\rceil$$

for some cut-set $S \subseteq V$ are still valid for the mRND polyhedron

$$\mathfrak{P}_{\texttt{mND}}(G,D) := \operatorname{conv}\{u \in \mathbb{Z}^E_{\geq 0} \mid \text{there is a feasible } D\text{-multi-commodity flow in } (G,u)\}$$

by Lemma 1.25. The idea is that if a source $i \in S$ wants to send d_{ij} units of flow to a sink $j \in V \setminus S$, then the cut must have sufficient capacity to support these d_{ij} units of flow.[7]

Cut-set inequalities of this type were first given by Magnanti and Mirchandani [MM93] and Magnanti, Mirchandani and Vachani [MMV95] for the multi-facility case where the authors show that an inequality for a cut-set S is facet-inducing if and only if its right-hand side is non-zero and if S is a strong cutset. These two works provide a Branch-and-Cut algorithm using cut-set inequalities and a heuristic separation. Barahona [Bar96] uses cut-set inequalities for the buy-at-bulk case of mND. He also provides a Branch-and-Cut algorithm; again with a heuristic separation. A variant where the task is to expand existing capacities is given by Bienstock and Günlük [BG96]; also in this case, strong cut-sets characterize facet-inducing cut-set inequalities. Bienstock and Günlük generalize the cut-set inequalities to *flow* cut-set inequalities for an arc-flow formulation. These inequalities contain non-zero coefficients for the flow variables. Atamtürk [Ata02] gives a complete linear description consisting of cut-set inequalities for the sND problem where capacities can only be installed in batch sizes of $c \in \mathbb{Z}_{\geq 1}$. Atamtürk then generalizes the inequalities to the sND and the mND problem with multiple facilities and shows that they remain facet-defining if, essentially, both sides of the inducing cuts have non-zero demands. Ortega and Wolsey [OW03] study the computational effectiveness of cut-set and flow cut-set inequalities. Costa, Cordeau and Gendron [CCG09] compare cut-set and metric inequalities and their relationship to Benders' decomposition [Ben62]. Mattia [Mat10a] adapts the results by Magnanti, Mirchandani and Vachani [MMV95] to the mRND problem as we shall see later, showing in particular facet-inducing robust cut-set inequalities for the mRND-P problem. Altın, Yaman and Pınar [AYP11] show facet-inducing robust cut-set inequalities for the mRND-P with Hose-robustness, multiple facilities and static routing. Raack, Koster, Orlowski

[7] The same is true in the converse situation where a source i in $V \setminus S$ sends d_{ij} units of flow to a sink $j \in S$, because in our definition of multi-commodity flows in undirected graphs, the pairs (i,j) and (j,i) define two separate commodities. As all commodities must be sent simultaneously, we can add up the demands of all separated terminal pairs.

and Wessäly [RKOW11] compare undirected, bi-directed and directed cut-set inequalities for the mND problem and characterize whether these inequalities induce facets. They generalize the inequalities to facet-inducing arc-residual inequalities and study the computational effectiveness of the different inequality classes. Facet-inducing robust cut-set inequalities for the mRND-P problem with Γ-robustness and static routing were given by Koster, Kutschka and Raack [KKR13].

2.6.3 Additional Partitioning Based Inequalities

Cut-set inequalities arise by partitioning the node set of a graph into two partitions and by finding a lower bound for the edges that connect the two partitions. The same principle applies for any number $k \geq 2$ of partitions. Here, finding the best lower bound for the connecting edges often amounts to solving the original problem on an auxiliary graph in which the nodes of each partition are aggregated into a single node. Chopra and Rao [CR94] define Steiner k-partition inequalities for the Steiner tree problem, observing that at least $k-1$ edges are necessary to connect any k-partition of the node set, provided that all partitions contain terminals. They show that Steiner k-partition inequalities induce facets if the auxiliary graph described above is biconnected.

Magnanti, Mirchandani and Vachani [MMV95] show that 3-partition inequalities for the mND problem with two facilities can be derived by building a $\{0, \frac{1}{2}\}$-cut (see [CF96] and Chapter 1) from the cut-set inequalities induced by the three partitions. We show in Chapter 4 that the principle works for the sRND problem. We also characterize which 3-partition inequalities define facets there. Bienstock, Chopra, Günlük and Tsai [BCGT98] show 3-partition inequalities for the basic mND problem.

Also for the basic mND problem, Agarwal [Aga06] shows how to lift any facet of the polyhedron defined by the auxiliary instance such that it yields a facet of the polyhedron of the original instance. A similar lifting theorem exists by Chopra and Rao [CR94] for the Steiner tree problem. Agarwal then uses the lifting theorem to find facet-inducing 4-partition inequalities. Mattia [Mat10a] shows that Agarwal's theorem also holds for the mRND problem.

We show in Chapter 4 that Agarwal's theorem can be translated to the sRND problem as well. Here, Buchheim, Liers and Sanità [BLS11] had previously proposed to separate valid inequalities for the sRND-F flow formulation using general cutting planes (so-called target-cuts [BLO08]). In order to separate a capacity vector u^*, the target-cut procedure projects u^* to the auxiliary instance on k-partitions in the above sense. If the projection of u^* is not feasible, Buchheim, Liers and Oswald find a violated facet-inducing inequality on the auxiliary instance. This is done by solving a linear program. The violated inequality is then lifted back to the original instance where it is still valid and violated by u^*. Thus, the procedure yields a violated k-partition inequality. Furthermore, by our new adaptation of Agarwals' lifting theorem, the lifted inequality is facet-inducing.

2.7 An Overview of Related Works

	capacities	*flow model*	*link model*	*robustness*	*routing*	*LP/IP type*
Ford and Fulkerson [FF58]	fractional	MC	U	—	—	path-flow
Minoux [Min81]	fractional	MC	U	finite	dynamic	arc-flow, capacity
Magnanti and Wong [MW84]	binary	MC	U/D	—	—	arc-flow
Minoux [Min89]	fractional, linear /concave costs	SC/MC	U/D	finite	dynamic	arc-flow, capacity
Magnanti, Mirchandani and Vachani [MMV95]	integer	MC	U, 2-FAC	—	—	arc-flow
Barahona [Bar96]	binary	MC	U (D)	—	—	arc-flow, capacity
Bienstock and Günlük [BG96]	binary	MC	U, 2-FAC	—	—	arc-flow
Bienstock, Chopra, Günlük and Tsai [BCGT98]	integer	MC	D	—	—	arc-flow, capacity
Günlük [Gün99]	integer	MC	U	—	—	arc-flow, capacity
Labbé, Séguin, Soriano and Wynants [LSSW99]	integer	MC	U	finite	dynamic on prescribed subgraphs	arc-flow, lagrangian
Atamtürk [Ata02]	integer	SC	U, k-FAC	—	—	arc-flow, capacity
Mirchandani [Mir00]	integer	SC/MC	U, k-FAC	—	—	arc-flow, capacity
Ortega and Wolsey [OW03]	binary	SC	U	—	—	arc-flow
Avella, Mattia and Sassano [AMS04]	integer	MC	U	finite	dynamic	arc-flow, capacity
Erlebach and Rüegg [ER04]	fractional	MC	B	Hose	static, single-path, tree	arc-flow

	capacities	*flow model*	*link model*	*robustness*	*routing*	*LP/IP type*
Ben-Ameur and Kerivin [BAK05]	fractional	MC	D	general polyhedral, Hose	static	path-flow
Agarwal [Aga06]	integer	MC	U	—	—	arc-flow, capacity
Altın, Amaldi, Belotti and Pınar [AABP07]	integer	MC	U	Hose, Γ	static & unsplit-table	arc-flow, path-flow
Mudchanatongsuk, Ordóñez and Liu [MOL08]	binary	MC	D	general polyhedral, ellipsoidal	dynamic	arc-flow
Koster, Orlowski, Raack *et al.* [KOR+09]	binary	MC	U, multi-layer	—	—	arc-flow
Costa, Cordeau and Gendron [CCG09]	fractional	MC	D	—	—	arc-flow, capacity
Altın, Yaman and Pınar [AYP11]	integer	MC	U, k-FAC	general polyhedal, Hose	static	arc-flow
Buchheim, Liers and Sanità [BLS11]	integer	SC	U	finite	dynamic	arc-flow
Poss and Raack [PR11]	fractional	MC	U	general polyhedral	affine recouse	arc-flow
Raack, Koster, Orlowski and Wessäly [RKOW11]	binary	MC	U/D/B	—	—	arc-flow, capacity
Ljubić, Putz and Salazar-González [LPSG12]	binary	MC, single-source	U/D, k-FAC	—	—	arc-flow, capacity
Koster, Kutschka and Raack [KKR13]	integer	MC	U	Γ	static	arc-flow
Lee, Lee and Park [LLP13]	binary	MC	U, k-FAC	Γ	static	arc-flow, capacity
Mattia [Mat13]	integer	MC	U	general polyhedral, Hose	dynamic	arc-flow, capacity

Chapter 3

Scenarios with a Single Source and Sink

In our definition of the sRND problem, we allow arbitrary balance vectors $b \in \mathfrak{B}$. This means, in particular, that we do not limit the number of nodes which have a non-zero balance. In this chapter, however, we assume that in each scenario $b \in \mathfrak{B}$, there is a single node s with a positive balance $b_s > 0$ and a single node t with a negative balance $b_t < 0$. The additional assumption does not change the complexity of the problem as was shown by Maculan [Mac87a]. If the underlying network is a complete graph, however, there is a polynomial time algorithm by Kabadi, Yan, Du and Nair [KYDN09]. We will see a more detailed discussion of the problem in this chapter, starting with an algorithm by Gomory and Hu [GH61] for the fractional variant. Finally, we show how to solve the problem on hypercubes with uniform balances in constant time. The results in this Chapter were obtained in a collaboration with Eduardo Álvarez-Miranda, Valentina Cacchiani, Tim Dorneth, Michael Jünger, Frauke Liers, Andrea Lodi and Tiziano Parriani. They have been published in [ACDJ+12] and [MCL+14].

3.1 Problem Complexity

The sRND problem with a single source and an arbitrary number of sinks is $\mathcal{NP}$-hard, even if the number k of scenarios is fixed to $k = 3$ [San09; BLS11]. The proof is by a reduction from 3-dimensional matching and inherently requires a non-constant number of sinks. However, Sanità [San09] reports on a reduction from Steiner tree that only uses a single source and a single sink per scenario, but requires a non-constant number of scenarios.

Theorem 3.1 (Sanità, Maculan [San09; Mac87a])**.** *The finite* sRND *problem where each scenario has a single source and a single sink is $\mathcal{NP}$-hard, even when all node balances are -1, 0 or 1.*

Proof. We prove the $\mathcal{NP}$-hardness by a reduction from the $\mathcal{NP}$-hard Steiner tree problem [Kar72]. Suppose that (V, E, c, T) is an arbitrary instance for the Steiner tree problem. We construct an input for the sRND problem in the following way: If T is empty, construct an sRND instance with an empty scenario set. Otherwise, fix some arbitrary terminal $s \in T$ and create a scenario b^t for any $t \in T \setminus \{s\}$ by setting

$$b_i^t := \begin{cases} 1, & \text{if } i = s \\ -1, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases}$$

and collect all scenarios in $\mathfrak{B} := \{b^t \mid t \in T \setminus \{s\}\}$. Let u be an optimum solution for the resulting sRND instance $(V, E, c, \mathfrak{B})$ on the same node and edge set. We claim that the support graph H of u is a Steiner tree and has the same cost as u. Firstly, to make u feasible, all terminals $t \in T$ must be connected to s. Also, H is a tree: If H contained a cycle C, then C would connect all nodes on it with a capacity of 2. However, the maximum absolute balance of any terminal is 1 and therefore we can remove the most expensive edge from C without destroying the feasibility of u. On the other hand, any Steiner tree solution for (V, E, c, T) immediately induces a feasible solution for the sRND instance $(V, E, c, \mathfrak{B})$ of the same cost. Therefore, solving sRND with a single source and sink per scenario also enables us to solve the $\mathcal{NP}$-hard Steiner tree problem. □

A similar reduction can be used to show that the variant with a polyhedral uncertainty set is $\mathcal{NP}$-hard, see Chapter 5.

3.2 The Network Synthesis Problem

The single-source single-sink variant of the sRND problem is in fact the origin of network design problems. Chien [Chi60] defines the Network Synthesis problem in the following way. Denote by $|f_{s\to t}^{\max}|$ the value of a maximum s-t-flow in an undirected network (V, E, u, s, t).

Problem 3.2 (Chien [Chi60])**.** *Given a* complete *undirected graph (V, E) and a flow requirement $r_{st} = r_{ts}$ for all pairs $s \neq t \in V$ of nodes, find capacities $u \in \mathbb{R}_{\geq 0}^E$ with minimum cost $\sum_{i,j \in E} u_{ij}$ such that $|f_{s\to t}^{max}| \geq r_{st}$ for all pairs $s \neq t \in V$.*

This problem is equivalent to the fractional sRND problem where each scenario has a single source and a single sink and the underlying network topology is a complete graph. Any instance of the network synthesis problem can be made into an sRND instance with exactly $|V|(|V|-1)/2$ scenarios, each having a single source s and a single sink t (and, to be more precise, in scenario s,t, the supply/demand of s and t is exactly r_{st}). On the other hand, it does not make sense to have more than one scenario for each unordered pair $\{s,t\}$ of a source s and a sink t, because if we are able to route the scenario with the highest balance for any unordered pair of nodes $\{s,t\}$, we can also route all other scenarios with that set of terminals.[1] Therefore, any sRND instance on a complete graph that has a single source and sink per scenario is a Network Synthesis problem.

Kabadi, Yan, Du and Nair [KYDN09] give a short overview over the existing algorithms for this problem and its variants. Let us, however, focus on an algorithm by Gomory and Hu [GH61] here, because its decomposition technique is similar to the one that we will use in Section 3.3.

3.2.1 A Decompositioning Technique by Gomory and Hu

The algorithm by Gomory and Hu is a divide-and-conquer algorithm that iteratively decomposes the requirement matrix. It ends up with a family of requirement matrices for which suitable networks can be synthesized easily. It then recomposes those networks into a final solution. The difficulty is to show that the re-composed solution is actually an optimum one.

Decomposing Requirements

The Network Synthesis problem has an important decomposition property. If we distribute the requirements of an instance (V,E,r) among two instances (V,E,r') and (V,E,r''), then we can obtain a feasible solution for (V,E,r) by adding up feasible solutions of (V,E,r') and (V,E,r'').

Lemma 3.3 (Gomory and Hu [GH61])**.** *Let (V,E,r) be an instance for the Network Synthesis problem and let $r = r' + r''$ with $r', r'' \in \mathbb{R}^E_{\geq 0}$. If u' and u'' are feasible solutions for (V,E,r') and (V,E,r''), respectively, then $u' + u''$ is a feasible solution for (V,E,r).*

Proof. For all $i,j \in V$ let $f'(i,j)$ be a maximum flow in (V,E,u') of value at least r'_{ij} and let $f''(i,j)$ be a maximum flow in (V,E,u'') of value at least $(r'')_{ij}$. Simply adding up $f'(i,j)$ and $f''(i,j)$ gives us a flow that sends $(r'+r'')_{ij}$ units of flows between i and j. To send $(f'+f'')(i,j)$, the capacities $u'+u''$ are sufficient and therefore, $(u'+u'')$ is a feasible solution for $(V,E,r'+r'')$. □

Unfortunately, even if u' and u'' are optimum solutions, their sum $u'+u''$ is not necessarily optimum for $(V,E,r'+r'')$ (see Figure 3.1). We say in this case that the decomposition does not maintain optimality. If we decompose correctly, however, optimality is preserved: We make sure that both the decomposed and the resulting solution satisfy the following lower bound *exactly* – which implies that the resulting solution is optimal.

[1] Even more, Gomory and Hu [GH61] show that there is always a dominating scenario set of size $|V|-1$, see Lemma 3.8.

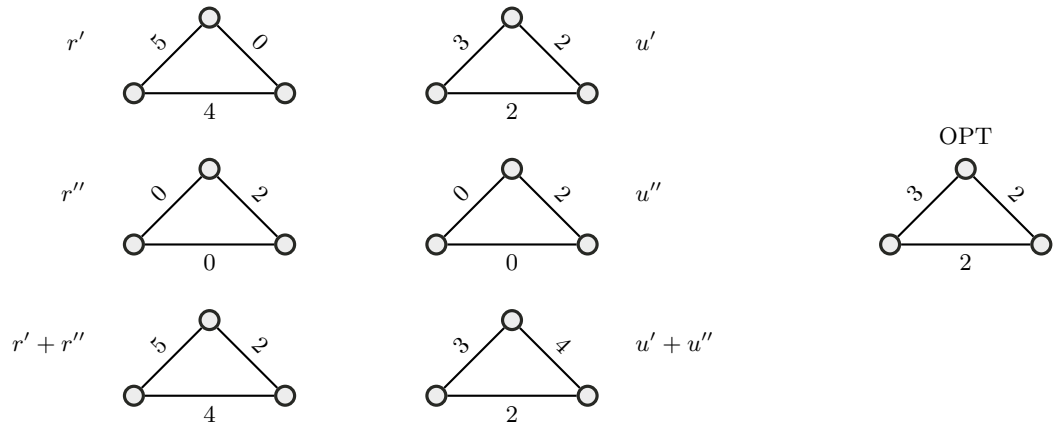


Figure 3.1: If the decomposition is not chosen carefully, adding the solution for the decomposed instances can lead to a suboptimal solution for the original instance. In this case, decomposing r into $r' + r''$ has lead to a solution $u' + u''$ of value 9, while the optimum solution has a value of 7. This is despite the fact that both u' and u'' satisfy (3.1) with equality and despite of r'' being uniform.

Lemma 3.4 (Gomory and Hu [GH61])**.** *Let (V, E, r) be a network synthesis instance. Any feasible solution $u \in \mathbb{R}^E_{\geq 0}$ satisfies*

$$\sum_{\{i,j\} \in E} u_{ij} \geq \frac{1}{2} \sum_{i \in V} \max_{j \in V} r_{ij} \tag{3.1}$$

Proof. Observe that u can only be feasible if at any node i the capacity out of i is at least as large as the maximum requirement that i has, i.e.,

$$\sum_{j \in \delta(i)} u_{ij} \geq \max_{j \in V} r_{ij}.$$

Adding up for all $i \in V$ we obtain

$$\begin{aligned} & \sum_{i \in V} \sum_{j \in \delta(i)} u_{ij} \geq \sum_{i \in V} \max_{j \in V} r_{ij} \\ \iff \qquad & \sum_{\{i,j\} \in E} u_{ij} \geq \frac{1}{2} \sum_{i \in V} \max_{j \in V} r_{ij} \end{aligned}$$

which gives us the desired lower bound on the total capacity. □

We define $r^{min} := \min\{r_{ij} \mid \{i,j\} \in X\}$ as the minimum positive requirement on the support $X := \{\{i,j\} \in E \mid r_{ij} > 0\}$ of r.

$$r''_{ij} := \begin{cases} r^{min} & \text{if } r_{ij} > 0 \\ 0 & \text{otherwise} \end{cases} \qquad \text{for all } \{i,j\} \in E \tag{3.2}$$

In this way, we obtain two instances (V, E, r') and $(V, E, r - r')$. This is a decomposition of (V, E, r) because $r = r' - r + r'$. Suppose that we have feasible capacities u' and u'' for

r' and $r - r'$, respectively, such that both pairs (u', r') and $(u'', r - r')$ satisfy (3.1) with equality. Then we have

$$\begin{aligned}
\frac{1}{2}\sum_{i\in V}\max_{j\in V}\{r_{ij}\} &= \frac{1}{2}\sum_{i\in V}\max_{j\in V}\{r'_{ij} + (r_{ij} - r'_{ij}\} \\
&= \frac{1}{2}\sum_{i\in V}\max_{\{i,j\}\in X}\{r'_{ij} + (r_{ij} - r'_{ij})\} \\
&= \frac{1}{2}\sum_{i\in V}\Big[r^{min} + \max_{\{i,j\}\in X}\{(r_{ij} - r'_{ij})\}\Big] \\
&= \frac{1}{2}\sum_{i\in V}\Big[\max_{\{i,j\}\in X}\{r'_{ij}\} + \max_{\{i,j\}\in X}\{(r_{ij} - r'_{ij})\}\Big] \\
&= \frac{1}{2}\sum_{\{i,j\}\in E} u'_{ij} + u''_{ij}
\end{aligned}$$

and we see that also $(u' + u'', r)$ satisfies (3.1) with equality – and this implies that $u' + u''$ is optimum for (V, E, r). It now remains to find optimum solutions for (V, E, r') and $(V, E, r - r')$ that satisfy (3.1) with equality. Observe at this point that the requirements in (V, E, r') are *uniform*.

Definition 3.5. *For $\beta \in \mathbb{R}_{\geq 0}$, a requirement matrix r is β-uniform on $X \subseteq E$ if*

$$r_{ij} = \begin{cases} \beta, & \text{if } \{i,j\} \in X \\ 0, & \text{otherwise} \end{cases} \qquad \text{for all } \{i,j\} \in E$$

Due to the uniform requirements r', it is conceivable that finding optimum solutions for (V, E, r') is much easier than finding optimum solutions for (V, E, r). The instance $(V, E, r - r')$, however, can have an arbitrary structure and therefore it is not clear why the decomposition should help us. The central observation is now that if we apply the decomposition technique recursively to $(V, E, r - r')$, we will at some point decompose into two instances that are both uniform. Therefore, solving an arbitrary Network Synthesis instance (V, E, r) can be reduced to solving a sequence of *uniform* Network Synthesis instances.

Dominant Requirement Trees

Before we actually synthesize a network, we make a further simplification: Any instance (V, E, r) can be reduced such that the support of r induces a tree. We will see in this subsection why this is true. The key argument is the following famous lemma.

Lemma 3.6 (Gomory and Hu [GH61])**.** *Let $N = (V, E, u)$ be an arbitrary network with capacities $u \in \mathbb{R}^E_{\geq 0}$. Then, for any path $(i_1, \ldots, i_p)$ in N, we have*

$$|f^{max}_{i_1 \to i_p}| \geq \min\{|f^{max}_{i_1 \to i_2}|, \ldots, |f^{max}_{i_{p-1} \to i_p}|\}$$

Proof. We show that for any $i, j, k \in V$, we have

$$|f^{\max}_{i\to k}| \geq \min\{|f^{\max}_{i\to j}|, |f^{\max}_{j\to k}|\}$$

from which our original claim follows by induction. Assume that $|f^{\max}_{i\to k}| < \min\{|f^{\max}_{i\to j}|, |f^{\max}_{j\to k}|\}$ for some choice of $i, j, k \in V$. Then, by the MaxFlow-MinCut Theorem 1.17 there is a cut $X \subseteq V$ such that $i \in X$, $j \in V \setminus X$ and the capacity of X is exactly $|f^{\max}_{i\to k}|$. Now, if $j \in X$, the same Theorem implies that $|f^{\max}_{j\to k}| < |f^{\max}_{i\to k}|$, which is a contradiction. Similarly, if $j \in V \setminus X$ we have a contradiction because $|f^{\max}_{i\to j}| < |f^{\max}_{i\to k}|$. □

This means that if at least ρ flow units can be sent between all two adjacent nodes i_j, i_{j+1} on a i_1-i_p-path, then at least ρ units of flow can be sent from i_1 to i_p. This is of course true: We obtain a valid flow by adding up the flows between the nodes of the path and canceling out any resulting cycles. We now use Lemma 3.6 to reduce the number of non-zero requirements.

Definition 3.7 (Gomory and Hu [GH61])**.** *Let (V, E, r) be an instance of the Network Synthesis Problem. A dominant requirement tree of (V, E, r) is a* maximum spanning tree *of (V, E) where the weight of $\{i, j\} \in E$ is exactly r_{ij}.*

The dominant requirement tree is called *dominant* because its edges induce dominating requirements. This is what the next lemma states.

Lemma 3.8 (Gomory and Hu [GH61])**.** *Let (V, E, r) be an instance of the Network Synthesis Problem and let T be the edge set of a dominant requirement tree of (V, E, r). If we define*

$$r^T := \begin{cases} r_{ij} & \text{if } \{i, j\} \in T \\ 0 & \text{otherwise} \end{cases},$$

then a solution u is feasible for (V, E, r) if and only if it is feasible for (V, E, r^T).

Proof. Any solution for (V, E, r) must in particular satisfy the requirements r_{ij} for all $\{i, j\} \in T$. For the other direction, suppose that u is a feasible solution for (V, E, r^T) and let $i, j \in V$. Then, there is a path $(i, i_1, \ldots, i_p, j)$ connecting i to j in T. Moreover, since u is feasible, $|f^{\max}_{i\to j}| \geq r_{ij}$ for all $\{i, j\} \in T$. By Lemma 3.6, we have

$$|f^{\max}_{i\to j}| \geq \min\{|f^{\max}_{i\to i_1}|, \ldots, |f^{\max}_{i_p\to j}|\} \geq \min\{r_{i,i_1}, \ldots, r_{i_p,p}\} \geq r_{ip},$$

where the last inequality is valid because T is a maximum spanning tree. □

All that remains now is to synthesize optimal networks where the requirements are uniform and induced by a tree.

Synthesizing Networks with Uniform Requirement Trees

Suppose now that (V, E, r) is an instance of the Network Synthesis Problem with β-uniform requirements and that the support $X := \{\{i, j\} \in E \mid r_{ij} > 0\}$ of r is a tree. Gomory and Hu construct an optimum solution $u \in \mathbb{R}^E_{\geq 0}$ for (V, E, r) that exactly meets the lower bound (3.1) in the following way.

If X consists of a single edge $\{i, j\}$, set $u_{ij} = \beta$. Otherwise, choose any cycle $C \subseteq E$ that contains exactly the nodes $i \in V[X]$ (this can be done easily because (V, E) is a complete graph). Then, set $u_{ij} = \beta/2$ for all $\{i, j\} \in C$ and $u_{ij} = 0$, otherwise.

The resulting solution u is feasible for (V, E, r) when $X = \{\{i', j'\}\}$ because the requirement between i' and j' is β and exactly β units of flow can be sent over $\{i', j'\}$. It also satisfies (3.1) with equality in this case, as we have

$$\sum_{\{i,j\}\in E} u_{ij} = u_{i',j'} = \beta = \frac{1}{2}\sum_{i\in V} \max_{\{i,j\}\in X} r_{ij} = \frac{1}{2}\sum_{i\in V} \max_{j\in V} r_{ij}.$$

The solution u is also feasible if $|X| > 1$: Any pair i, j of terminals in (V, E, r) has a requirement of β and lies on the constructed cycle C. It also decomposes C into two disjoint paths, each having a capacity of $\beta/2$. Thus, β units of flow can be sent from i to j (and vice-versa). In this case, we get

$$\sum_{\{i,j\}\in E} u_{ij} = \sum_{\{i,j\}\in X} u_{ij} = \sum_{i\in V[X]} \frac{\beta}{2} = \frac{1}{2}\sum_{i\in V[X]} \max_{j\in V} r_{ij} = \frac{1}{2}\sum_{i\in V} \max_{j\in V} r_{ij}$$

and we see that u satisfies (3.1) with equality.

Summary: The Complete Network Synthesis Algorithm by Gomory and Hu

In summary, we obtain Algorithm 1 that is able to synthesize a network for an arbitrary requirement matrix. It only requires that the underlying network topology is a complete graph.

Algorithm 1: The Network Synthesis Algorithm by Gomory and Hu [GH61]

input : A complete graph $G = (V, E)$ and a requirement matrix $r \in \mathbb{Z}^{V\times V}$
output: Capacities $u \in \mathbb{R}^E_{\geq 0}$ that support r

1 Let $u \equiv 0$
2 Compute a dominant requirement tree T of r
3 Set all requirements outside of T to zero (Lemma 3.8)
4 Compute the bottleneck requirement $r^{min} := \min\{r_{ij} > 0 \mid \{i, j\} \in E\}$
5 **if** *T consists of a single edge $\{ij\}$* **then**
6 Set $u_{ij} = r^{min}$
7 **return** u
8 **else if** *T is uniform* **then**
9 Find a Hamiltonian Cycle C through the nodes of T
10 Set $u_{ij} = r^{min}/2$ for all $\{i, j\} \in C$
11 **return** u
12 **end**
13 Choose r'' according to r^{min} and (3.2)
14 Recursively solve (V, E, r') and $(V, E, r - r')$, obtain solutions u' and u''
15 **return** $u' + u''$ (Lemma 3.3)

3.3 An Algorithm for Hypercube Graphs

The step in Line 9 of Algorithm 1 attempts to find a Hamiltonian Cycle. When the underlying network is not complete, however, this cycle does not necessarily exist and even if it does, finding it is $\mathcal{NP}$-hard. Still, there are other classes of graphs that *always* contain a Hamiltonian cycle and it is a natural idea to adapt Gomory and Hu's algorithm to one of them. In fact, one of these classes is formed by Hypercube Graphs and these shall be the subject of our studies in this section. In the sequel, we use the shorter word *Hypercube* instead of *Hypercube Graph*.

Definition 3.9. *A* d-dimensional hypercube $\mathfrak{C}_d$ *is the result of the following recursive construction:* $\mathfrak{C}_0$ *is the graph that consists of a single node. For* $d > 0$, $\mathfrak{C}_d$ *is obtained by duplicating the nodes and edges of* $\mathfrak{C}_{d-1}$ *and connecting each node* v *to its copy* v' *with an additional edge* $\{v, v'\}$.

Definition 3.10. *We say that two nodes* v, w *are* diagonally opposite *on* $\mathfrak{C}_d$ *if the shortest path from* v *to* w *in* $\mathfrak{C}_d$ *has maximum length, i.e., length* d. *For each* $v \in \mathfrak{C}_d$ *there is exactly one diagonally opposite node* v^o.

The fact that a longest shortest path in $\mathfrak{C}_d$ has length d is well-known, as is the uniqueness of v^o, see for example the survey by Harary, Hayes and Wu [HHW88] on this subject.[2] The same is true for the size of hypercubes.

Observation 3.11. *The hypercube* $\mathfrak{C}_d$ *has* $N_d := 2^d$ *nodes and* $M_d := d \cdot 2^{d-1}$ *edges.*

Moreover, there is a straight-forward inductive proof that for $d > 1$, any hypercube is Hamiltonian: The 2-dimensional hypercube $\mathfrak{C}_2$ is itself a Hamiltonian cycle. Let $d > 2$. By its definition, $\mathfrak{C}_d$ is composed of two copies H_1, H_2 of $\mathfrak{C}_{d-1}$ and each of them admits a Hamiltonian cycle C_1, C_2, respectively, by induction. Then, for some edge $\{i, j\}$ in C_1, we have the Hamiltonian cycle $(C_1 \cup \{i, i'\} \cup \{j, j'\} \cup C_2) \setminus (\{i, j\} \cup \{i', j'\})$ where i' and j' are the copies of i, j in H_2.

Observation 3.12. *For any* $d > 1$, *the hypercube* $\mathfrak{C}_d$ *contains a Hamiltonian cycle.*

Let us now define a family of hypercube instances for the `sRND` problem in the following way. For any $d \geq 1$, the hypercube $\mathfrak{C}_d$ is composed of two $(d-1)$-dimensional hypercubes and as before, we call them H_1, H_2. Each of the copies has exactly 2^{d-1} nodes. For each node $i \in H_1$, we define a scenario where i is the only source, having balance 1 and its diagonally opposite node $i^o \in H_2$ is the only sink, having balance -1. Figure 3.2 shows the construction. We denote the instance obtained in this way by $\mathfrak{C}_d^1$. Scaling all balances in $\mathfrak{C}_d^1$ by $r \in \mathbb{Z}_{\geq 0}$, we obtain the instance $\mathfrak{C}_d^r$. In the remainder of this chapter, we try to find algorithms for solving $\mathfrak{C}_d^r$ with different r.

[2] Harary, Hayes and Wu argue that each node i of $\mathfrak{C}_d$ can be associated with a distinct vector $v^i \in \{0,1\}^d$ such that the distance of two nodes $i, j \in \mathfrak{C}_d$ is exactly the number of bits in which the corresponding vectors v^i, v^j differ. This implies that the maximum distance of two nodes is d and that only the node associated with the complementary vector of v^i can have distance d to node i.

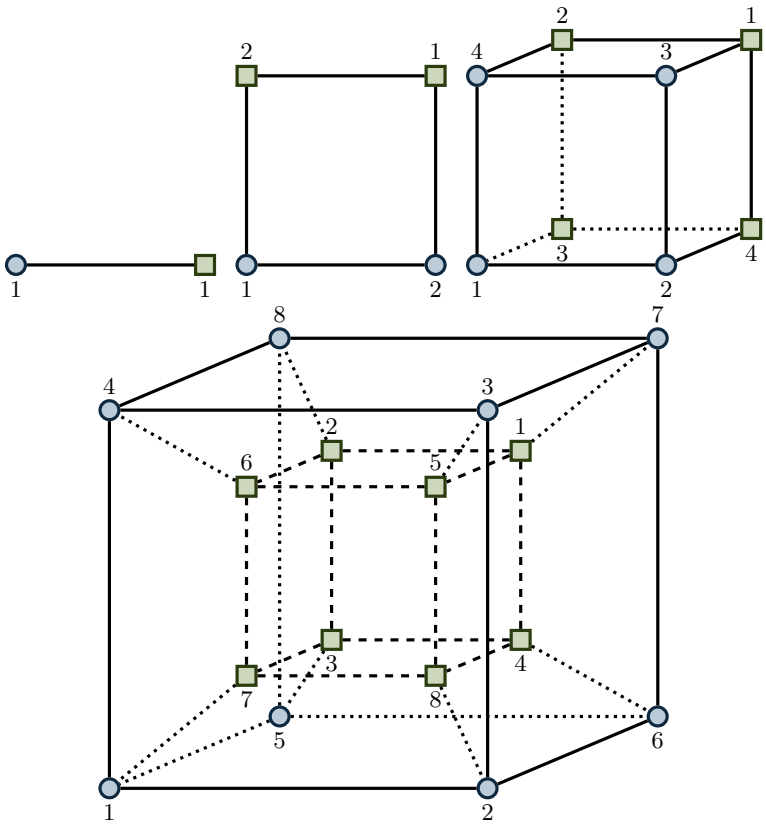


Figure 3.2: Hypercubes $\mathfrak{C}_d$ in $d = 1, 2, 3, 4$ dimensions. Circle nodes depict the nodes that are sources and square nodes depict those nodes that are sinks in our $\mathfrak{C}_d^r$ instances. The number next to the nodes defines the mapping of sources to sinks: Each source is mapped to its diagonally opposite node. All sources have a supply of r.

3.3.1 When Supplies and Demands are Binary

When all supplies and demands are binary, i.e., if $b_i \in \{-1, 0, 1\}$ for all $b \in \mathfrak{B}$, and every scenario $b \in \mathfrak{B}$ has a single source and a single sink node, it is not difficult to find an optimum fractional solution for the sRND problem on our hypercube instances. This is due to the uniform structure of the supplies and demands that allows us to obtain a feasible solution by setting a capacity of $1/d$ on every edge of $\mathfrak{C}_d$: Between any pair of diagonally opposite vertices, there are d disjoint paths and all of them have a capacity of $1/d$. Also, this solution has a value of $d \cdot 2^{d-1} \cdot \frac{1}{d} = 2^{d-1}$, which is optimal by Lemma 3.4, as we have $\frac{1}{2} \sum_{i \in V[\mathfrak{C}_d]} |\max b_i| = \frac{1}{2} |V[\mathfrak{C}_d]| = 2^{d-1}$.

Observation 3.13. *For $d \geq 1$, the cost of an optimum fractional solution of $\mathfrak{C}_d^1$ is 2^{d-1}.*

If we additionally require that the capacities must be integer, it is no longer clear how to find an optimum solution for the problem. Even more, on general graphs, the single source, single sink sRND problem with binary balances is a special case of the $\mathcal{NP}$-hard *Steiner forest* problem.

Definition 3.14. *Given an undirected graph $G = (V, E)$, an edge cost function $c : E \to \mathbb{R}_{\geq 0}$ and k disjoint sets $T_1, \ldots, T_k \subseteq V$, find a minimum cost forest $F \subseteq E$ such that for all $l = 1, \ldots, k$ and all $i, j \in S_l$, there exists a path from i to j in F.*

The Steiner forest problem is $\mathcal{NP}$-hard, since the case where $l = 1$ is exactly the Steiner tree problem. If we make S_i to contain exactly the source-sink pair of the i-th scenario, then we reduced the binary single-source single-sink `sRND` problem (`BSS-sRND`) to Steiner forest. Still, it is not clear if Steiner forest can be reduced so `BSS-sRND` since sets S_i with $|S_i| > 2$ cannot be trivially modeled in an `BSS-sRND` instance. To the best of the author's knowledge, it is unknown if this special case of Steiner forest remains $\mathcal{NP}$-hard (in particular when the underlying graph is a hypercube). There is, however, evidence that the problem is difficult to tackle with Branch-and-Cut methods. More precisely, we show in the sequel that the integrality gap, i.e., the quotient of an optimum integer and an optimum fractional solution converges to 2 as d goes to infinity. To this aim, we establish a lower bound on the value of an integer solution. An example of the situation for $d = 2$ is depicted in Figure 3.3.

Lemma 3.15. *Any integer solution of $\mathfrak{C}_d^1$ has a total cost of at least $2^d - \lfloor 2^{d-1}/d \rfloor$.*

Proof. We first show that any connected component of the support graph of an integer feasible solution u^I contains at least $2d$ nodes: Each connected component C of the support of u^I must contain one source s and its corresponding sink t. Yet, since the shortest path between any source-sink-pair in $\mathfrak{C}_d$ contains d edges, $d - 1$ additional nodes $V' \subseteq V$ must be contained in C. As all nodes are terminals, each node in V' is a terminal in some scenario. However, for no source or sink in V' the corresponding terminal can lie in V' because otherwise the shortest path between such a terminal pair would have less than d edges. Thus, another $d - 1$ nodes need to be contained in C, namely the diagonally opposite nodes of those nodes in A. This gives us that any connected component must contain at least $d + 1 + d - 1 = 2d$ nodes. Therefore, no feasible solution can contain more than $Z := \lfloor N_d/(2d) \rfloor = \lfloor 2^d/(2d) \rfloor$ connected components. However, in order to have at most Z connected components, the solution must contain at least $N_d - Z = 2^d - \lfloor 2^{d-1}/d \rfloor$ edges. □

Thus, we can bound the integrality gap $GAP(I_d)$ as follows:

$$GAP(I_d) \geq \frac{\sum_{e \in E[\mathfrak{C}_d]} u_e^I}{\sum_{e \in E[\mathfrak{C}_d]} u_e^F} \geq \frac{2^d - 2^{d-1}/d}{2^{d-1}} = 2 - \frac{1}{d} \xrightarrow{d \to \infty} 2$$

It remains open, however, if the problem can be solved by an efficient algorithm.

3.3.2 Uniform Supplies and Demands that are not Binary

Even when we explicitly forbid binary demands (i.e., if we require that $b_i \in \mathbb{Z} \setminus \{-1, 1\}$ for all nodes $i \in V$ and all scenarios $b \in \mathfrak{B}$), the `sRND` problem with a single source and a single sink per scenario remains $\mathcal{NP}$-hard on general graphs. This was pointed out to the author by Laura Sanità.

Theorem 3.16. *[San13] The `sRND` problem is $\mathcal{NP}$-hard, even if there is a single source and a single sink in each scenario and if $b_i \in \{-2, 0, 2\}$ for all $i \in V, b \in \mathfrak{B}$.*

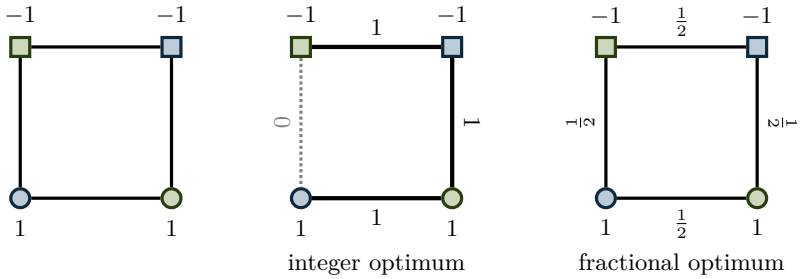


Figure 3.3: The instance $\mathfrak{C}_2$ with unit costs and two scenarios: The two green and blue nodes are the unique source-sink pair of the first and second scenario, respectively. The value of an optimum integer solution is 3 while an optimum fractional solution only has cost 2.

Proof. We show the claim by a reduction from Hamiltonian Cycle. Let $G = (V, E)$ be an arbitrary undirected graph. Construct an `sRND` instance that uses G as the underlying network topology. Then, fix some arbitrary node $s \in V$ and create a scenario b^i for all nodes $i \in V \setminus \{s\}$ by setting $b^i_s = 2$ and $b^i_i = -2$. Finally, set the cost of each edge e to 1. This gives us an `sRND` instance I_R with $|V| - 1$ scenarios. We claim that I_R has a solution of value $|V|$ if and only if G has a Hamiltonian cycle.

If G has a Hamiltonian cycle C, we build a feasible solution for I_R by installing a capacity of 1 on each edge of C. In each scenario b^i, both unique terminals 1 and i lie on C. The node i decomposes C into two paths P_1, P_2 from 1 to i (one clockwise, one counterclockwise). We can route one unit of flow on P_1 and one unit of flow on P_2, satisfying the demands of scenario i. Thus, our solution for I_R is feasible and additionally, it has cost of $|C| = |V|$.

On the other hand, suppose we have a solution for I_R of cost $|V|$. By our choice of scenarios (we have a single source at node 1 and all other nodes are terminals in some scenario), each node must be connected to node 1. Therefore, any feasible solution for I_R must have a support S that induces a connected component of G containing all nodes. S must contain at least $|V| - 1$ edges, otherwise it cannot be connected. If S contains exactly $|V| - 1$ edges, a capacity of 2 must be installed on each edge in S in order to route all demands. However, such a solution has cost of $2 \cdot |V| - 2 > |V|$ and therefore S must contain at least $|V|$ edges. If some node in $G[S]$ has a degree of 1, then we must install a capacity of 2 on its unique incident edge. By the same argument as before, the remaining $|V| - 1$ nodes must be connected by at least $|V| - 1$ edges. Then again, the cost of the solution is at least $|V| - 1 + 2 > |V|$. Therefore, all nodes in $G[S]$ must have a degree of at least 2 and because we can have at most $|V|$ edges in S, each node must have exactly degree 2. Together with our observation that $G[S]$ is connected and contains all nodes, we have a Hamiltonian cycle. □

Hypercubes with Uniform Non-Binary Supplies and Demands

In the following, we will see how to solve the problem efficiently when the underlying graph is a hypercube. This is not a contradiction to the $\mathcal{NP}$-hardness as the above reduction breaks down on Hamiltonian graphs. The idea for our algorithm is again to decompose an instance into instances that can be easily synthesized.

The atomic instances for our construction are the hypercubes $\mathfrak{C}_d^2$ and $\mathfrak{C}_d^3$. Indeed, we can obtain a feasible integer solution for $\mathfrak{C}_d^2$ by using the idea by Gomory and Hu [GH61] from Subsection 3.2.1: We install a capacity of 1 on each edge of a Hamiltonian cycle in $\mathfrak{C}_d^2$. As $\mathfrak{C}_d$ has 2^d nodes, this solution has a cost of 2^d.

Lemma 3.17. *For any $d \geq 2$, there is a feasible integer solution for $\mathfrak{C}_d^2$ with costs 2^d.*

In this case, splitting the supplies and demands evenly along two sides of a cycle yields an integer solution because r is even. If $r = 3$, however, this idea would give us only a fractional solution. Splitting the supplies and demands unevenly is not possible either since we would obtain a suboptimal solution in this way. This means that we need to split the supplies and demands evenly along *three* instead of *two* disjoint paths. The proof of the following lemma shows how this can be done.

Lemma 3.18. *For any $d \geq 3$, there is a feasible integer solution for $\mathfrak{C}_d^3$ with costs $3 \cdot 2^{d-1}$.*

Proof. Let $d \geq 3$. Then $\mathfrak{C}_d$ decomposes into two copies H_1, H_2 of $\mathfrak{C}_{d-1}$ and a set of edges F connecting H_1 and H_2. We install a capacity of 1 on each edge in F. Since $d - 1 \geq 2$, we find Hamiltonian cycles C_1, C_2 in H_1 and H_2, respectively, and install a capacity of 1 on each edge of C_1 and of C_2.

This solution is feasible: For any scenario $i \in \{1, \ldots, q\}$, let s_i, t_i be the corresponding terminal pair. We need to route three units of flow from s_i to t_i. To do that, let $s_i' \in H_2$ and $t_i' \in H_1$ be the unique nodes such that $e_1 = \{s_i, s_i'\} \in F$ and $e_2 = \{t_i', t_i\} \in F$. Also, let $e_3 = \{u, v\} \in F$ with $u \in H_1$ and $v \in H_2$ be an arbitrary connecting edge that is different from e_1 and e_2. Because $d \geq 3$, the set F contains at least four edges. Figure 3.4 shows an example for the situation on $\mathfrak{C}_4^3$. Now, by sending one unit of flow over each of e_1, e_2, e_3, we have reduced the instance to two instances on $\mathfrak{C}_{d-1}$: The first instance is defined on H_1; here, s_i has a balance of 2 and both u and t_i' have a balance of -1. However, these balances can be routed along the Hamiltonian C_1. In the second instance, which is defined on H_2, the sink t_i has a balance of -2 and both s_i' and v have a balance of 1. Again, these balances can be routed along the Hamiltonian cycle C_2.

Both C_1 and C_2 contain exactly 2^{d-1} edges, each with capacity 1. There are 2^{d-1} edges in F, all of them having capacity 1. This gives a total cost of $3 \cdot 2^{d-1}$. □

By Lemma 3.3, we can add up the solutions of $\mathfrak{C}_d^2$ and $\mathfrak{C}_d^3$ to obtain a feasible solution of $\mathfrak{C}_d^{2+3}$. Iterating the argument gives us the following corollary to Lemma 3.3.

Corollary 3.19. *Let $d \geq 2$ and let $r = 2m + 3n$ with $m \in \mathbb{Z}_{\geq 0}$ and $n \in \{0, 1\}$. If there exists an integer feasible solution for $\mathfrak{C}_d^2$ with cost at most c_2 and an integer feasible solution for $\mathfrak{C}_d^3$ with cost at most c_3, then there exists an integer feasible solution for $\mathfrak{C}_d^r$ with cost at most $m \cdot c_2 + n \cdot c_3$.*

Putting together our previous results, we can solve $\mathfrak{C}_d^r$ to optimality.

Theorem 3.20. *For any $d \geq 3$ and any $r \geq 2$, an optimum integer solution for the d-dimensional hypercube instance $\mathfrak{C}_d^r$ has a total cost of $r \cdot 2^{d-1}$.*

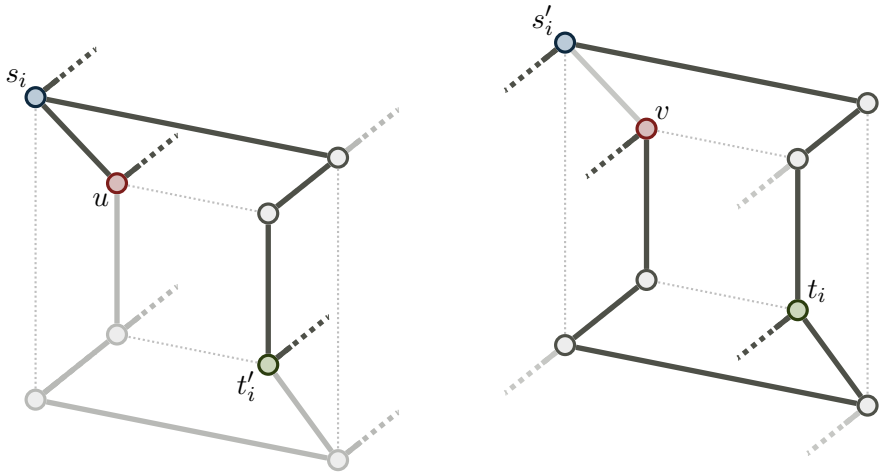


Figure 3.4: Construction for $\mathfrak{C}_4^3$: We subdivide $\mathfrak{C}_4$ into two copies of $\mathfrak{C}_3$ on the left and on the right. The thick edges have a capacity of 1. The node s_i is connected to t_i with a capacity of 3 along the dark edges.

Proof. Let $r = 2m + 3n$ with $m \in \mathbb{Z}_{\geq 0}$ and $n \in \{0, 1\}$. Putting together Lemma 3.19 with Lemma 3.17 and Lemma 3.18, we obtain that there is an integer solution for $\mathfrak{C}_d^r$ with value $c_r := m \cdot 2^d + n \cdot 3 \cdot 2^{d-1}$. If r is even, we have $n = 0$ and $m = r/2$. Therefore, $c_r = r \cdot 2^{d-1}$. On the other hand, if r is odd, we have $n = 1$ and $m = (r-3)/2$. Then, $c_r = (r-3)/2 \cdot 2^d + 3 \cdot 2^{d-1} = r \cdot 2^{d-1} - 3 \cdot 2^{d-1} + 3 \cdot 2^{d-1} = r \cdot 2^{d-1}$. By Lemma 3.4, this is optimal. □

Lemma 3.4 also tells us that no fractional solution can have a value that is lower than $r \cdot 2^{d-1}$ and therefore, the integrality gap of $\mathfrak{C}_d^r$ is 1 for $r \geq 2$ and $d \geq 3$. Analogously to the binary case, such a solution can also be obtained by installing a capacity of r/d on every edge.

Corollary 3.21. *An optimum fractional solution for $\mathfrak{C}_d^r$ has a value of $r \cdot 2^{d-1}$. In particular, for $r \geq 2$ and $d \geq 3$, we have $GAP(\mathfrak{C}_d^r) = 1$.*

We conclude this section by giving an alternative proof of Corollary 3.21. It requires the following structural lemma.

Lemma 3.22. *Let $d \geq 3$. Then in $\mathfrak{C}_d$, $|\delta(S)| \geq d$ for all $\emptyset \subsetneq S \subsetneq V^d$. Moreover, equality is attained if and only if $|S| = 1$ or $|S| = |V^d| - 1$.*

Proof. The first part of the lemma is well-known: Saad and Schultz [SS88, Propositions 3.2 and 3.3] proved that for any two nodes i, j of a d-dimensional hypercube, there are at least d node disjoint paths between i and j. By Menger's Theorem [Men27], this implies that $|\delta(S)| \geq d$ for all $\emptyset \subsetneq S \subsetneq V^d$. Also, if S contains a single node i, then $|\delta(S)| = |\delta(i)| = d$. It remains to show that the inequality is strict if $2 \leq |S| \leq |V^d| - 2$. Without loss of generality, we can assume that $|S| \leq \frac{1}{2}|V^d|$ since $\delta(S) = \delta(V \setminus S)$.

Now, choose an arbitrary decomposition of $\mathfrak{C}_d$ into two $(d-1)$-dimensional hypercubes $H_1 = (V_1, E_1), H_2 = (V_2, E_2)$ such that neither of $S_1 := S \cap V_1$ and $S_2 := S \cap V_2$ is empty.

This is possible because S contains at least two and at most $|V|/2$ nodes. It also implies that neither $S_1 = V_1$ nor $S_2 = V_2$, as otherwise S_2 or S_1 would be empty, respectively.

For $i = 1, 2$, the node set S_i defines a cut $\delta_i(S_i)$ in H_i. As $S_i \neq \emptyset$ and $S_i \neq V_i$, we know that $|\delta_i(S_i)| \geq d-1$, since H_i is a $(d-1)$-dimensional hypercube. Also, $\delta_1(S_1), \delta_2(S_2) \subseteq \delta(S)$ and therefore $|\delta(S)| \geq 2 \cdot (d-1) > d$ for $d \geq 3$. □

We can now give the alternative proof of Corollary 3.21.

Proof of Corollary 3.21. If we define the set

$$\mathcal{S} := \{S \subset V^d \mid S \text{ is connected and separates at least one } v_q \text{ from its partner } v_q^o\},$$

we can find an optimum fractional solution for $\mathfrak{C}_d^r$ with the following linear program from Chapter 4.

$$\begin{aligned} \min \quad & \sum_{e \in E^d} u_e && && (3.3)\\ \text{s.t.} \quad & \sum_{e \in \delta(S)} u_e \geq r && \text{for all } S \in \mathcal{S} \\ & u_e \geq 0 && \text{for all } e \in E \end{aligned}$$

If $d = 2$, it holds that $|S| = d = 2$ for all $S \in \mathcal{S}$. Consequently, if we set $u_e = r/2$ for all $e \in E^d$, all primal constraints are satisfied with equality and the solution is optimal. If $d \geq 3$, we introduce dual variables ξ_S for all $S \in \mathcal{S}$ and obtain the following dual program:

$$\begin{aligned} \max \quad & \sum_{S \in \mathcal{S}} r \cdot \xi_S && && (3.4)\\ \text{s.t.} \quad & \sum_{\substack{S \in \mathcal{S}: \\ \{i,j\} \in S}} \xi_S \leq 1 && \text{for all } \{i,j\} \in E^d \\ & \xi_S \geq 0 && \text{for all } S \in \mathcal{S} \end{aligned}$$

We consider the following pair of primal and dual solutions:

$$u_e := r/d \qquad \text{for all } e \in E^d \qquad\qquad \xi_S := \begin{cases} 0, & \text{if } |\delta(S)| > d \\ 1/2, & \text{if } |\delta(S)| = d \end{cases} \qquad \text{for all } S \in \mathcal{S}.$$

To prove our claim, we need to show that u and ξ are feasible and satisfy complementary slackness. Feasibility of u follows by the first part of Lemma 3.22: For all $S \in \mathcal{S}$, we have $|\delta(S)| \geq d$ and thus $\sum_{e \in \delta(S)} u_e = |\delta(S)|(r/d) \geq r$. Observe that by the second part of Lemma 3.22 equality holds if and only if $|\delta(S)| = d$. Thus, we have $(\sum_{e \in \delta(S)} u_e - r) \cdot \xi_S = 0$ for all $S \in \mathcal{S}$, yielding primal complementary slackness. To see why ξ is feasible for (3.4) we need to show that

$$\sum_{\substack{S \in \mathcal{S}: \\ \{i,j\} \in S}} \xi_S = \sum_{\substack{S \in \mathcal{S}: \\ |\delta(S)| = d \\ \{i,j\} \in S}} \xi_S \leq 1 \qquad \text{for all } \{i,j\} \in E^d.$$

By applying Lemma 3.22, we can rewrite this as

$$\sum_{\substack{S\in\mathcal{S}:\\ |\delta(S)|=d\\ \{i,j\}\in S}} \xi_S = \sum_{\substack{S\in\mathcal{S}:\\ |S|=1\\ \{i,j\}\in S}} \xi_S = \xi_{\{i\}} + \xi_{\{j\}} = 1 \qquad \text{for all } \{i,j\} \in E^d$$

which also yields that $(\sum_{S\in\mathcal{S}:e\in S} \xi_S - 1) \cdot u_e = 0$ for all $e \in E^d$, i.e., we have dual complementary slackness. Finally, both solutions yield the desired objective value of $\sum_{e\in E^d} r/d = d \cdot 2^{d-1} \cdot (r/d) = r \cdot 2^{d-1}$. □

Explicit and Implicit Encodings of Hypercubes: Complexity Issues

In principle, we can encode the instance $\mathfrak{C}_d^r$ by simply stating the numbers r and d. In that case, an efficient algorithm for $\mathfrak{C}_d^r$ is an algorithm with a running time that is polynomial in $\log r + \log d$. This time is sufficient to compute the *value* of an optimum solution. The optimum solution itself, however, has size $\Theta(E_d) = \Theta(d \cdot 2^d)$, a term that is doubly-exponential in $\log d$. Outputting it requires super polynomial time. If, however, the instance $\mathfrak{C}_d^r$ is encoded explicitly the input length is $O(E_d)$. We can then execute the recursive construction from the previous subsection in time that is polynomial in the input length.

3.4 Extensions

The algorithm by Gomory and Hu [GH61] can be extended to yield integer capacities, provided that the underlying network is a complete graph, see for example the work by Kabadi, Yan, Du and Nair [KYDN09] and the references therein. Still, all algorithms in this section assume that each edge of the network has unit costs. For the weighted case, Gomory and Hu [GH62] give an linear programming based algorithm that produces fractional solutions. The author is not aware of any exact combinatorial algorithm yielding integer solutions on complete graphs with general edge weights; though Sanità [San09] observes that 2-approximation algorithms [Jai01; Hoc97] for the Steiner forest Problem also work here.

Chapter 4

The Polyhedral Structure of the sRND Problem

In this chapter, we study the convex hull $\mathfrak{R}(G, \mathfrak{B})$ of all integer feasible capacity vectors for an `sRND` instance $(G, \mathfrak{B})$, where G is a connected, undirected graph and $\mathfrak{B}$ is an arbitrary `sRND` scenario set. Our aim is to find a good linear description of $\mathfrak{R}(G, \mathfrak{B})$. This description should work for the finite case of the `sRND` problem as well as for its polyhedral case. Thus, the results in this chapter are independent of the representation and the structure of $\mathfrak{B}$. At the start of the chapter, we study the basic properties of $\mathfrak{R}(G, \mathfrak{B})$: We show that the polyhedron is full-dimensional and give a linear programming relaxation whose size is independent of the number of vertices of $\mathfrak{B}$. The relaxation consists of cut-set inequalities. We strengthen the relaxation with 3-partition inequalities and show that these inequalities can be derived as $\{0, \frac{1}{2}\}$-cuts. Both our cut-set inequalities and the 3-partition inequalities define facets of $\mathfrak{R}(G, \mathfrak{B})$. Finally, we show that $\mathfrak{R}(G, \mathfrak{B})$ is degenerate. The results in this Chapter were obtained in a collaboration with Eduardo Álvarez-Miranda, Valentina Cacchiani, Tim Dorneth, Michael Jünger, Frauke Liers, Andrea Lodi and Tiziano Parriani. They have been published in [ACDJ+12] and [CJL+14].

4.1 Dimension of the sRND Polyhedron

As before, we assume throughout the chapter that $G = (V, E)$ is a connected, undirected graph and that $\mathfrak{B} \subseteq \mathbb{R}^V$ is an `sRND` scenario set. This gives us an `sRND` instance $(G, \mathfrak{B})$. Most results of this chapter are independent of edge costs, but whenever necessary, we also assume that the costs are given by a vector $c \in \mathbb{R}^E_{\geq 0}$.

The convex hull of all integer capacity vectors that are feasible for the `sRND` instance $(G, \mathfrak{B})$ forms a polyhedron whose structure is independent of the representation of $\mathfrak{B}$. We call this polyhedron the `sRND` polyhedron.

Definition 4.1. *The* `sRND` *polyhedron of a connected, undirected graph $G = (V, E)$ and a scenario set $\mathfrak{B} \subseteq \mathbb{R}^V$ is the set*

$$\mathfrak{R}(G, \mathfrak{B}) := \operatorname{conv}\{u \in \mathbb{Z}^E_{\geq 0} \mid u \text{ is feasible for } (G, \mathfrak{B})\}$$

of all convex combinations of integer feasible capacity vectors for $(G, \mathfrak{B})$.

If we relax the integrality conditions and consider the convex hull of all *fractionally* feasible capacity vectors, we again obtain a polyhedron.

Definition 4.2. *The* fractional `sRND` *polyhedron with respect to an undirected, connected graph $G = (V, E)$ and a scenario set $\mathfrak{B} \subseteq \mathbb{R}^V_{\geq 0}$ is the set*

$$\widetilde{\mathfrak{R}}(G, \mathfrak{B}) := \operatorname{conv}\{u \in \mathbb{R}^E_{\geq 0} \mid u \text{ is feasible for } (G, \mathfrak{B})\}$$

of all fractionally feasible capacity vectors for $(G, \mathfrak{B})$.

In general, the integer `sRND` polyhedron $\mathfrak{R}(G, \mathfrak{B})$ is a proper subset of the fractional polyhedron $\widetilde{\mathfrak{R}}(G, \mathfrak{B})$. Moreover, there are instances where an optimum integer solution of the `sRND` problem is at least two times more expensive than an integer one (see Chapter 3). Thus, the fractional polyhedron $\widetilde{\mathfrak{R}}(G, \mathfrak{B})$ can have fractional vertices in general. In the following, we close a part of the gap between the two polytopes and to that aim, we study the structure of $\mathfrak{R}(G, \mathfrak{B})$. Our first observation is that $\mathfrak{R}(G, \mathfrak{B})$, and thus also $\widetilde{\mathfrak{R}}(G, \mathfrak{B})$, is unbounded: If $u \in \mathfrak{R}(G, \mathfrak{B})$ is a feasible capacity vector, then increasing the capacity of any edge will never make u infeasible. More formally, if υ_i denotes the i-th unit vector, then for $i = 1, \ldots, |E|$ we have $u + \upsilon_i \in \mathfrak{R}(G, \mathfrak{B})$. Thus $\mathfrak{R}(G, \mathfrak{B})$ is unbounded (unless G or $\mathfrak{B}$ are empty – in that case, $\mathfrak{R}(G, \mathfrak{B})$ is empty as well).

Observation 4.3. *Let $G = (V, E)$ an undirected connected graph with $|E| \geq 1$. Let $\emptyset \subsetneq \mathfrak{B} \subsetneq \mathbb{R}^V$ be a scenario set. Then $\mathfrak{R}(G, \mathfrak{B})$ is unbounded and the unit vectors υ_i span unbounded directions of $\mathfrak{R}(G, \mathfrak{B})$ for all $i = 1, \ldots, |E|$.*

In particular, the vectors $u + \upsilon_1, \ldots, u + \upsilon_{|E|}$ are linearly independent and therefore, $\mathfrak{R}(G, \mathfrak{B})$ is full-dimensional.

Lemma 4.4 ([Mat13; Dor12])**.** *Let $G = (V, E)$ be an undirected, connected graph with $|E| \geq 1$ and let $\emptyset \subsetneq \mathfrak{B} \subsetneq \mathbb{R}^V$ be a scenario set. Then $\mathfrak{R}(G, \mathfrak{B})$ has dimension $|E|$.* □

Knowing the dimension of $\mathfrak{R}(G, \mathfrak{B})$ is crucial for understanding its facial structure.

4.2 An IP-Formulation with Facet-Inducing Cut-Set Inequalities

After we considered the unbounded part of the sRND polyhedron in the previous section, we now study its boundary. The first step in our analysis is to develop a linear description for the relaxation $\widetilde{\mathfrak{R}}(G, \mathfrak{B})$, i.e., we need to characterize the feasibility of a capacity vector $u \in \mathbb{R}^E_{\geq 0}$ with linear inequalities. We have seen in Chapter 2 that the Japanese Theorem 1.26 yields such a characterization for the multi-commodity case. It is natural to use the analogon for single-commodity flows in our case and indeed, we use Gale's Theorem 1.23 to derive a new IP formulation for sRND. The formulation builds on the the well-known class of *cut-set inequalities* that are used for several network design problems (see the overview in Section 2.6.1) and its size does not depend on the number of vertices of $\mathfrak{B}$. This will enable us to handle the polyhedral case of sRND.

4.2.1 Characterizing the sRND Problem with Cut-Set Inequalities

Gale's Theorem 1.23 only works for a fixed scenario $b \in \mathfrak{B}$, but it contains a necessary condition for the existence of a b-flow that is based on the following idea. Consider a fixed cut-set $S \subseteq V$. In general, the set S will have an excess of supply or demand in the scenario b and any feasible b-flow must balance out this excess by routing flow units from S into $V \setminus S$ or vice-versa. Whether S has an excess of supply or an excess of demand is determined by the set's total balance $B_S := \sum_{i \in S} b_i$ with respect to b. If B_S is strictly positive, the set S has an excessive supply and any feasible b-flow must route B_S flow units from S to $V \setminus S$. If B_S is strictly negative, we are in the converse situation and any feasible b-flow routes B_S flow units from $V \setminus S$ to S. In both cases, the capacity of $\delta(S)$ must at least be $|B_S|$; otherwise, no feasible b-flow can exist. This is why, for all choices of $S \subseteq V$ and $b \in \mathfrak{B}$, the *non-robust cut-set inequality*

$$\sum_{\{i,j\} \in \delta(S)} u_{ij} \geq \Big|\sum_{i \in S} b_i\Big| \tag{4.1}$$

is a valid inequality for $\mathfrak{R}(G, \mathfrak{B})$. Since the inequality is valid for all choices of $b \in \mathfrak{B}$, it is in particular valid for a $b \in \mathfrak{B}$ that maximizes the right-hand side. We thus obtain that

$$\sum_{\{i,j\} \in \delta(S)} u_e \geq \max_{b \in \mathfrak{B}} \Big|\sum_{i \in S} b_i\Big|$$

is valid for all $S \subseteq V$.

Definition 4.5. *Let $G = (V, E)$ be an undirected graph, let $S \subseteq V$ and assume that $\mathfrak{B}$ is a finite or a polyhedral uncertainty set. We then call the inequality*

$$\sum_{\{i,j\} \in \delta(S)} u_{ij} \geq \max_{b \in \mathfrak{B}} \Big|\sum_{i \in S} b_i\Big| \tag{CS$_S$}$$

the cut-set-inequality *induced by S. We use (CS_S) as a short-hand notation for the inequality and we denote its right-hand side by R_S.*

In the above definition, we can replace S by its complement and still obtain the same inequality: As any $b \in \mathfrak{B}$ satisfies $\sum_{i \in V} b_i = 0$ (which implies that $\sum_{i \in S} b_i = -\sum_{i \in V \setminus S} b_i$), the right-hand sides of CS_S and $\mathrm{CS}_{V \setminus S}$ coincide. We also have $\delta(S) = \delta(V \setminus S)$, giving us that the left-hand sides of CS_S and $\mathrm{CS}_{V \setminus S}$ coincide as well.

Observation 4.6. *For any $S \subseteq V$, we have $CS_S = CS_{V \setminus S}$.*

As any vector $u \in \widetilde{\mathfrak{R}}(G, \mathfrak{B})$ must satisfy the cut-set inequality induced by any $S \subseteq V$, we have found a necessary criterion for the feasibility of u. Gale's Theorem, however, also proves a *sufficient* condition for the existence of a b-flow that depends on a fixed vector $b \in \mathfrak{B}$: If we have $\sum_{\{i,j\} \in \delta(S)} u_{ij} \geq |\sum_{i \in S} b_i|$ for all cut-sets $S \subseteq V$, then a feasible b-flow exists in (G, u). Thus, if a capacity vector $u \in \mathbb{R}^E_{\geq 0}$ satisfies the non-robust cut-set inequality (4.1) for all $S \subseteq V$ and all $b \in \mathfrak{B}$, then u is (fractionally) feasible for $(G, \mathfrak{B})$. We summarize our findings in a theorem.

Theorem 4.7. *Let $G = (V, E)$ be a connected, undirected graph and let $\mathfrak{B}$ be a scenario set. Then $u \in \widetilde{\mathfrak{R}}(G, \mathfrak{B})$ if and only if*

$$\sum_{\{i,j\} \in \delta(S)} u_{ij} \geq \max_{b \in \mathfrak{B}} \Big| \sum_{i \in S} b_i \Big|$$

for all $S \subseteq V$. We have $u \in \mathfrak{R}(G, \mathfrak{B})$ if and only if additionally $u \in \mathbb{Z}^E_{\geq 0}$. □

Thus, we have found a description of $\widetilde{\mathfrak{R}}(G, \mathfrak{B})$ as a linear program

$$\begin{aligned} \min \quad & \sum_{\{i,j\} \in E} c_{ij} u_{ij} && && (4.2) \\ \text{s.t.} \quad & \sum_{\{i,j\} \in \delta(S)} u_{ij} \geq \max_{b \in \mathfrak{B}} \Big| \sum_{i \in S} b_i \Big| && \text{for all } S \subseteq V \\ & u_{ij} \in \mathbb{R}_{\geq 0} && \text{for all } \{i,j\} \in E \end{aligned}$$

consisting only of cut-set inequalities and trivial non-negativity constraints. In particular, we can formulate the `sRND` problem as an integer linear program as follows.

$$\begin{aligned} \min \quad & \sum_{\{i,j\} \in E} c_{ij} u_{ij} && && (4.3) \\ \text{s.t.} \quad & \sum_{\{i,j\} \in \delta(S)} u_{ij} \geq \max_{b \in \mathfrak{B}} \Big| \sum_{i \in S} b_i \Big| && \text{for all } S \subseteq V \\ & u_{ij} \in \mathbb{Z}_{\geq 0} && \text{for all } \{i,j\} \in E \end{aligned}$$

We refer to this capacity formulation as the `sRND` **cut-set formulation**. The program (4.2) is the linear programming relaxation of the cut-set formulation (4.3).

Corollary 4.8. *A capacity vector $u \in \mathbb{Z}^E_{\geq 0}$ is feasible for the* `sRND` *instance $(G, \mathfrak{B})$ if and only if it is feasible for the cut-set formulation* (4.3). □

4.2.2 Cut-Set Formulation vs. Arc-Flow Formulation

Theorem 4.7 yields in particular that a vector u is feasible for the linear programming relaxation of the cut-set formulation (4.3) if and only if it is feasible for the linear programming relaxation of the arc-flow formulation (2.17) by Buchheim, Liers and Sanità [BLS11] that we discussed in Chapter 2. Thus, the two formulations yield exactly the same bound. In that sense, the arc-flow formulation is an extended formulation of the cut-set formulation; it introduces additional variables to avoid having an exponential number of constraints.

Corollary 4.9. *A vector $u \in \mathbb{R}^E_{\geq 0}$ is feasible for the linear programming relaxation of the capacity formulation (4.3) if and only if there exist flows $f^1, \dots, f^k$ such that $(f^1, \dots, f^k, u)$ is feasible for the linear programming relaxation of the flow formulation (2.17).* □

However, this formulation has the same drawbacks as the robust flow-formulations in Chapter 2: It requires variables and constraints for every scenario $b \in \mathfrak{B}$ and even if $\mathfrak{B}$ is given as a finite list, it can thus grow very large. If $\mathfrak{B}$ is given in a linear description, the formulation becomes unmanageable in general. We therefore concentrate on the cut-set formulation in the remaining part of the thesis.

In the non-robust case, the capacity formulation can be obtained by applying Benders' decomposition [Ben62] to the flow formulation, see e.g. [MW81], and although Benders' original decomposition technique yields a slightly weaker version of (4.3), the same principle applies here.

4.2.3 Relationship to the Robustness Models from the Literature

We continue this section with the observation that the cut-set formulation (4.3) is a special case of Soyster's [Soy73] more general robustness framework from Section 2.4. To see why, fix an arbitrary order of the edges in $E = \{e^1, \dots, e^m\}$ and let χ^ℓ be the cut incidence vector of the edge e^ℓ, i.e., for all $\ell = 1, \dots, m$ and all $S \subseteq V$ let $\chi^\ell_S = 1$ if $e^\ell \in \delta(S)$ and $\chi^\ell_S = 0$ otherwise. Then, the cut-set formulation can be equivalently written in Soyster's notation as

$$\begin{aligned} \min \quad & c^{\mathrm{T}} u && && (4.4)\\ \text{s.t.} \quad & \begin{pmatrix} | & & | & | \\ \chi^1 & \cdots & \chi^m & \beta \\ | & & | & | \end{pmatrix} \cdot \begin{pmatrix} u \\ -1 \end{pmatrix} \geq 0 && \text{for all } \beta \in \mathfrak{K} \\ & && u \in \mathbb{R}^E_{\geq 0} \end{aligned}$$

where $\mathfrak{K} = \{(|\sum_{i \in S} b_i|)_{S \subseteq V} \mid b \in \mathfrak{B}\}$ defines the set of possible realizations of the last column of the constraint matrix. Applying Soyster's ideas to this formulation yields the (in this case trivial) observation that we can solve (4.4) by solving an auxiliary linear program.

$$\begin{aligned} \min \quad & c^{\mathrm{T}} u && && (4.5)\\ \text{s.t.} \quad & \begin{pmatrix} | & & | & | \\ \chi^1 & \cdots & \chi^m & \beta^* \\ | & & | & | \end{pmatrix} \cdot \begin{pmatrix} u \\ -1 \end{pmatrix} \geq 0 \\ & && u \in \mathbb{R}^E_{\geq 0} \end{aligned}$$

Here, the vector $\beta^* = (\max_{b\in\mathfrak{B}}|\sum_{i\in S} b_i|)_{S\subseteq V}$ contains the row-wise maxima of the entries of the uncertain last column. In this way, we exactly obtain the cut-set formulation (4.3).

4.2.4 Cut-Set Inequalities Induce Facets

Cut-set inequalities not only characterize $\widetilde{\mathfrak{R}}(G,\mathfrak{B})$, they also induce facets of $\mathfrak{R}(G,\mathfrak{B})$. This result was already known for the non-robust multi-commodity network design problem [MMV91] since the 90's, before Mattia [Mat13] adapted it for the mRND in 2010, showing that tight metric inequalities – and in particular `mRND` cut-set inequalities – are facet-inducing. Finally, Dorneth observed in his diploma thesis [Dor12] that Mattia's proofs only need small adaptations for the `sRND-F` problem. We repeat a more concise version here and make an (easy) extension to the polyhedral demand case. As before we define $R_S := \max_{b\in\mathfrak{B}}|\sum_{i\in S} b_i|$ and we let $B^* := \max_{b\in\mathfrak{B}} \sum_{i\in V} |b_i|$ be an upper bound for the capacity on any edge.

Theorem 4.10 ([Mat13; Dor12])**.** *For any cut $S \subseteq V$, (CS_S) defines a facet of $\mathfrak{R}(G,\mathfrak{B})$ if and only if $R_S > 0$ and if the subgraphs induced by S and $V \setminus S$ are connected.*

Proof. If $R_S = 0$, then (CS_S) cannot be stronger than the trivial inequalities $u_e \geq 0$ for $e \in \delta(S)$. Also, if S (or likewise, $V \setminus S$) decomposes into several connected components $S_1, \ldots, S_k$, then summing up the inequalities we get from $S_1, \ldots, S_k$ yields the same left-hand side as we get from S; yet, the right-hand side of $(\mathrm{CS}_{S_1}) + \cdots + (\mathrm{CS}_{S_k})$ can only be stronger than the one of (CS_S) by the triangle inequality.

Finally, in order to show that (CS_S) defines a facet of $\mathfrak{R}(G,\mathfrak{B})$ we define a vector u^e for every edge $e \in E$ in the way suggested by Mattia [Mat10a, Theorem 3.14]. In doing so, our choice depends on whether e lies in $\delta(S)$. For all $e \in \delta(S)$, define u^e as

$$u^e_{e'} := \begin{cases} R_S & \text{if } e' \in \delta(S), e' = e \\ 0 & \text{if } e' \in \delta(S),\ e' \neq e \\ B^* & \text{if } e' \notin \delta(S) \end{cases} \qquad \text{for all} e' \in E$$

Now, for all $e \notin \delta(S)$ and some fixed $h \in \delta(S)$ choose u^e as

$$u^e_{e'} := \begin{cases} R_S & \text{if } e' \in \delta(S), e' = h \\ 0 & \text{if } e' \in \delta(S), e' \neq h \\ B^* + 1 & \text{if } e' \notin \delta(S), e' = e \\ B^* & \text{if } e' \notin \delta(S), e' \neq e \end{cases} \qquad \text{for all} e' \in E$$

Because we have $R_S \neq 0$, the vectors $u^e, e \in E$, are linearly independent. This is easily verified by considering the upper triangular matrix with the rows u^e for $e \in \delta(S)$ followed by the rows $u^e - u^h$ for $e \notin \delta(S)$. For all $e \in \delta(S)$, the vector u^e satisfies (CS_S) with equality since $\sum_{e'\in\delta(S)} u^e_{e'} = u^e_e = R_S$ by the definition of u^e. If $e \notin \delta(S)$, it follows that $\sum_{e'\in\delta(S)} u^e_{e'} = u^e_h = R_S$ and again, (CS_S) is satisfied with equality.

It remains to show that $u^e \in \mathfrak{R}(G,\mathfrak{B})$ for all $e \in E$. We fix an arbitrary cut $X \subseteq V$ such that the subgraphs $G[X]$ and $G[V \setminus X]$ induced by X and $V \setminus X$, respectively, are connected. It then remains to show that u^e satisfies (CS_X) for all $e \in E$. We can assume

that $X \neq S$ and that $X \neq V \setminus S$ since we have already shown validity for those two cases. Thus, if $\delta(X) \subseteq \delta(S)$ was true, then either $G[X]$ or $G[V \setminus X]$ would not be connected and therefore, there exists at least one edge $e^* \in \delta(X) \setminus \delta(S)$. Using this observation for any $e \in E$ we have

$$\sum_{e' \in \delta(X)} u^e_{e'} \geq \sum_{e' \in \delta(X) \setminus \delta(S)} u^e_{e'} \geq u^e_{e^*} \geq B^* = \max_{b \in \mathfrak{B}} \sum_{i \in V} |b_i| \geq \max_{b \in \mathfrak{B}} \Big| \sum_{i \in X} b_i \Big|$$

which tells us that u^e satisfies (CS_X). We conclude that (CS_S) defines a face of dimension $|E| - 1$ and, therefore, induces a facet of $\mathfrak{R}(G, \mathfrak{B})$. □

4.3 Non-Negativity Constraints Induce Facets

We now show that the non-negativity constraints induce facets of $\mathfrak{R}(G, \mathfrak{B})$. This requires finding $|E| - 1$ feasible and linearly independent vectors that lie on the hyperplane defined by $u_e = 0$ for $e \in E$. However, a vector u with $u_e = 0$ can only be feasible if the edge e is redundant in all scenarios, i.e., if

- removing e from G does not disconnect G or
- the edge e disconnects G into subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, but the total balance of V_1 (and thus, in V_2) is zero in all scenarios.

In all other cases, however, the non-negativity constraints induce facets of $\mathfrak{R}(G, \mathfrak{B})$.

Theorem 4.11. *Let $G = (V, E)$ be a connected, undirected graph with $|E| \geq 1$ and let $\mathfrak{B} \subseteq R^V$ be a scenario set. Then, for any $e \in E$, the inequality $u_e \geq 0$ induces a facet of $\mathfrak{R}(G, \mathfrak{B})$ if and only if one of the following conditions hold.*

1. *The edge e is not a bridge or*
2. *if e is a bridge inducing the partition $V = V_1 \dot\cup V_2$, then $\sum_{i \in V_1} b_i = 0$ for all $b \in \mathfrak{B}$.*

Proof. We fix some arbitrary edge $e^* \in E$. Without loss of generality, we can index the edges in G such that $E = \{e_1, \ldots, e_m\}$ and $e_m = e^*$. We also define $G' := (V, E \setminus \{e_m\})$. If e_m is not a bridge, then G' is connected and $(G', \mathfrak{B})$ is a valid instance of the `sRND` problem. In that case, the face $H := \{u \in \mathbb{R}^E_{\geq 0} \mid u_m = 0\} \cap \mathfrak{R}(G, \mathfrak{B})$ induced by the inequality $u_m \geq 0$ is exactly the polyhedron $\mathfrak{R}(G', \mathfrak{B})$ spanned by the integer solutions of the reduced instance $(G', \mathfrak{B})$. By Lemma 4.4 we have immediately that $\mathfrak{R}(G', \mathfrak{B})$ is full-dimensional, i.e., the induced face H has dimension $|E \setminus \{e_m\}| = |E| - 1$.

If otherwise e_m *is* indeed a bridge inducing the partitioning $V_1 \dot\cup V_2$, then $(G, \mathfrak{B})$ decomposes into two independent instances $(G_1 = (V_1, E_1), \mathfrak{B}_1)$ and $(G_2 = (V_2, E_2), \mathfrak{B}_2)$. Assume w.l.o.g. that $E_1 = \{e_1, \ldots, e_\ell\}$ for $\ell = |E_1|$. By our assumption that $\sum_{i \in V_1} b_i = 0$ for all $b \in \mathfrak{B}$, it follows that both instances are valid `sRND` instances and therefore, their respective polyhedra $\mathfrak{R}(G_1, \mathfrak{B}_1)$ and $\mathfrak{R}(G_2, \mathfrak{B}_2)$ are again full-dimensional by Lemma 4.4. As the instances can be solved independently (any pair of feasible solutions for $(G_1, \mathfrak{B}_1)$ and $(G_2, \mathfrak{B}_2)$ induces a feasible solution for $(G, \mathfrak{B})$ and vice-versa) we now have $H = \mathfrak{R}(G_1, \mathfrak{B}_1) \times \mathfrak{R}(G_2, \mathfrak{B}_2) \times \{0\}$. This implies $\dim H = \dim \mathfrak{R}(G_1, \mathfrak{B}_1) + \dim \mathfrak{R}(G_2, \mathfrak{B}_2) = |E_1| + |E_2| = |E| - 1$.

For the other direction of the theorem, assume by contraposition that both conditions are false, i.e., that $\sum_{i \in V_1} b_i \neq 0$ for some $b \in \mathfrak{B}$. Then, however, the cut-set inequality

$$u_m \geq \max_{b \in \mathfrak{B}} |\sum_{i \in V_1} b_i| > 0$$

induced by V_1 dominates the inequality $u_m \geq 0$ and H cannot be a facet of $\mathfrak{R}(G, \mathfrak{B})$. This concludes the proof. □

4.4 Deriving 3-Partition Facets as Chvátal-Gomory Cuts

For any cut-set $S \subseteq V$, the cut-set inequality (CS_S) gives a lower bound on the capacity needed between a 2-partition $V = S \dot\cup V \setminus S$ of the node set. In general, however, we could ask for lower bounds on the capacity between any k-partition, $k \geq 2$, of the graph; and in particular, we will consider the case $k = 3$ in this section. Magnanti, Mirchandani and Vacchani [MMV93, Equation 11] observe that valid 3-partition inequalities for the mND problem can be obtained as $\{0, \frac{1}{2}\}$-Chvátal-Gomory Cuts as defined by Caprara and Fischetti [CF96]. To this purpose, they add up the cut-set inequalities induced by the partitions S, T and U, divide the resulting inequality by two and round up the right hand side. We will see that the same procedure works for the `sRND` problem and moreover, we generalize the result to arbitrary (i.e., not necessarily disjoint) subdivisions of the node set. It turns out that the 3-partition inequalities induce facets of $\mathfrak{R}(G, \mathfrak{B})$. As before, we write $(S : T) = \delta(S) \cap \delta(T)$ to denote set the of edges between two node sets $S, T \subseteq V$.

4.4.1 Chvátal-Gomory Cuts for the sRND Problem

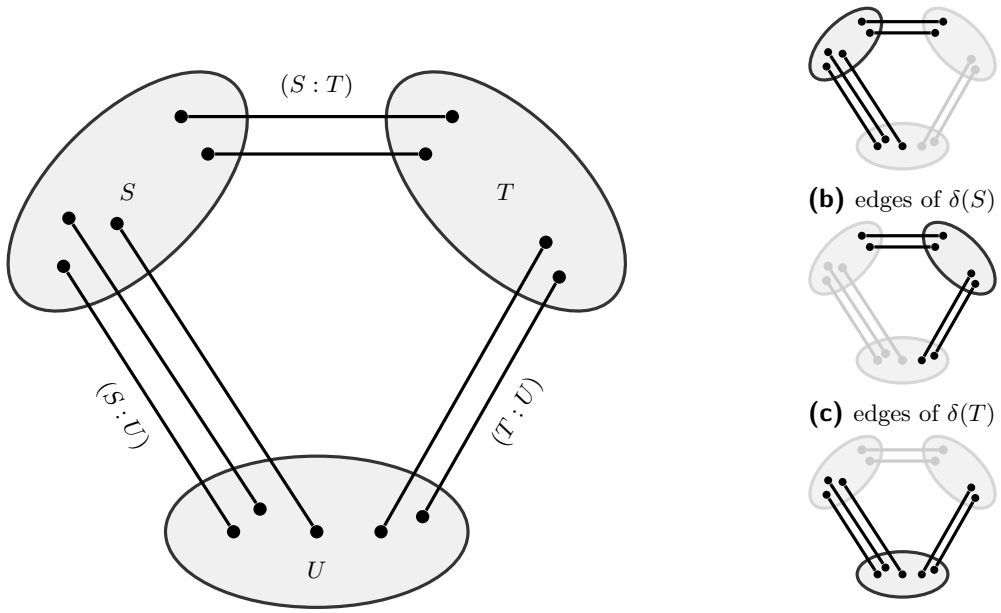
Suppose that we have a 3-partition S, T and U of the node set $V = S \dot\cup T \dot\cup U$. Since $T \dot\cup U$ is the complement of S, each edge in $\delta(S)$ must have one node in T or in U. Thus, we have a disjoint partitioning of the cut $\delta(S)$ as $\delta(S) = (S : T) \dot\cup (S : U)$. Likewise, the edges in $\delta(T)$ have a node in S or in U and those in $\delta(U)$ have a node in S or in T. We can therefore rewrite the cut-set inequalities induced by S, T and U.

$$\begin{array}{lllllll}
\sum\limits_{\{i,j\} \in (S:T)} u_{ij} & + & \sum\limits_{\{i,j\} \in (S:U)} u_{ij} & & & \geq & R_S \qquad (\mathrm{CS}_S) \\
\sum\limits_{\{i,j\} \in (S:T)} u_{ij} & & & + & \sum\limits_{\{i,j\} \in (T:U)} u_{ij} & \geq & R_T \qquad (\mathrm{CS}_T) \\
& + & \sum\limits_{\{i,j\} \in (S:U)} u_{ij} & + & \sum\limits_{\{i,j\} \in (T:U)} u_{ij} & \geq & R_U \qquad (\mathrm{CS}_U)
\end{array}$$

By adding up these cut-set inequalities, we obtain the valid inequality

$$2 \cdot \sum_{\{i,j\} \in (S:T)} u_{ij} + 2 \cdot \sum_{\{i,j\} \in (S:U)} u_{ij} + 2 \cdot \sum_{\{i,j\} \in (T:U)} u_{ij} \geq R_S + R_T + R_U$$

whose left-hand side coefficients are all even.



(a) edges with non-zero coefficient in the resulting $\{0, \frac{1}{2}\}$-cut

(b) edges of $\delta(S)$

(c) edges of $\delta(T)$

(d) edges of $\delta(U)$

Figure 4.1: Adding up the cut-set inequalities induced by S, T and U and dividing by two generates a $\{0, \frac{1}{2}\}$-cut that we interpret as a 3-partition inequality. We observe that each edge appears exactly twice in the three cuts.

Therefore, we obtain a valid $\{0, \frac{1}{2}\}$-cut by adding up $\frac{1}{2}[\mathrm{CS}_S + \mathrm{CS}_T + \mathrm{CS}_U]$ to

$$\tfrac{1}{2}\sum_{\{i,j\}\in\delta(S)} u_{ij} \;+\; \tfrac{1}{2}\sum_{\{i,j\}\in\delta(T)} u_{ij} \;+\; \tfrac{1}{2}\sum_{\{i,j\}\in\delta(U)} u_{ij} \;=\; \sum_{\{i,j\}\in(S:T)} u_{ij} \;+\; \sum_{\{i,j\}\in(S:U)} u_{ij} \;+\; \sum_{\{i,j\}\in(T:U)} u_{ij} \;\geq\; \left\lceil \frac{R_S + R_T + R_U}{2} \right\rceil$$

and because this inequality is induced by the three partitions S, T and U, we call it a **3-partition inequality**. The situation is depicted in Figure 4.1. By Observation 4.6, we can equivalently replace (CS_U) by $(\mathrm{CS}_{V\setminus U}) = \mathrm{CS}_{S\cup T}$ in our construction and still obtain the same $\{0, \frac{1}{2}\}$-cut as

$$\tfrac{1}{2}\Big[\sum_{\{i,j\}\in\delta(S)} u_{ij} \;+\; \sum_{\{i,j\}\in\delta(T)} u_{ij} \;+\; \sum_{\{i,j\}\in\delta(S\cup T)} u_{ij}\Big] \geq \left\lceil \frac{R_S + R_T + R_{S\cup T}}{2} \right\rceil.$$

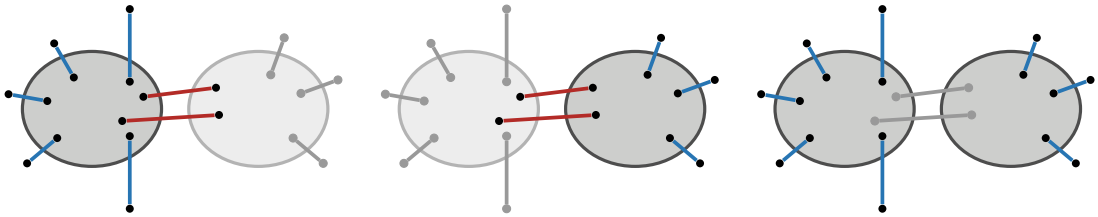
Indeed, Figure 4.2 gives an intuitive argument of why both cuts are equivalent. This rewritten form will be helpful for our separation algorithms.

Still, the above construction only works if the sets S and T are disjoint. If the intersection $S \cap T$ is not empty, then we can observe in Figure 4.3 that

$$\sum_{\{i,j\}\in\delta(S)} u_{ij} \;+\; \sum_{\{i,j\}\in\delta(T)} u_{ij} \;+\; \sum_{\{i,j\}\in\delta(S\cup T)} u_{ij} \;+\; \sum_{\{i,j\}\in\delta(S\cap T)} u_{ij} \;=\; 2\sum_{\{i,j\}\in\delta(S\cup T)} u_{ij} \;+\; 2\sum_{\{i,j\}\in(S:T)} u_{ij} \;+\; 2\sum_{\{i,j\}\in\delta(S\cap T)} u_{ij}.$$

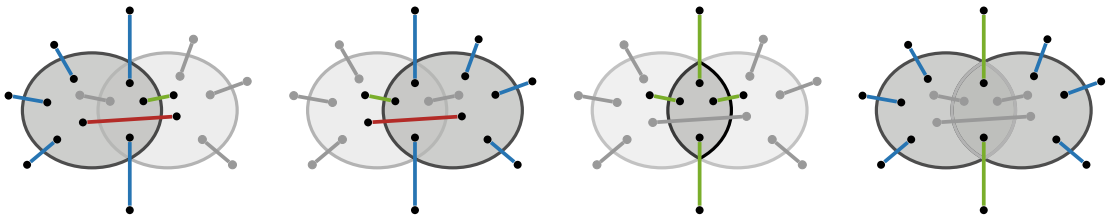
Therefore, given cut-set inequalities (CS_S) and (CS_T), we obtain a valid zero-half cut by adding up $\frac{1}{2}((\mathrm{CS}_S) + (\mathrm{CS}_T) + (\mathrm{CS}_{S\cup T}) + (\mathrm{CS}_{S\cap T}))$ to

$$\sum_{\{i,j\}\in\delta(S\cup T)} u_{ij} \;+\; \sum_{\{i,j\}\in(S:T)} u_{ij} \;+\; \sum_{\{i,j\}\in\delta(S\cap T)} u_{ij} \;\geq \left\lceil \frac{1}{2}(R_S + R_T + R_{S\cup T} + R_{S\cap T}) \right\rceil.$$



(a) $\delta(S)$ **(b)** $\delta(T)$ **(c)** $\delta(S \cup T)$

Figure 4.2: Summing over the edges in $\delta(S)$, $\delta(T)$ and $\delta(S \cup T)$ hits every edge either exactly twice or not at all. Additionally, the sum is the same as summing over $(S : T)$, $(S : V \setminus (S \cup T))$ and $(T : V \setminus (S \cup T))$.



(a) $\delta(S)$ **(b)** $\delta(T)$ **(c)** $\delta(S \cap T)$ **(d)** $\delta(S \cup T)$

Figure 4.3: The figure shows that summing over the edges of the cuts $\delta(S)$, $\delta(T)$, $\delta(S \cap T)$ and $\delta(S \cup T)$ once is the same as summing over the edges of $\delta(S \cup T)$, $\delta(S \cap T)$ and $(S : T)$ twice. The colors indicate a mapping of the edges to $\delta(S \cup T)$ [blue], $\delta(S \cap T)$ [green] and $(S : T)$ [red].

4.4.2 Separating 3-Partition Inequalities by Enumeration

Using the results from the previous subsection, we can separate 3-partition inequalities heuristically inside a cutting plane algorithm for cut-set inequalities. To do so, we consider a capacity vector $u^* \in \mathbb{R}^E_{\geq 0}$ and the set I of all cut-set inequalities that are present in the current linear programming relaxation. We now simply enumerate all pairs of cut-set inequalities $(\mathrm{CS}_S), (\mathrm{CS}_T) \in I$. For each such pair, we generate the cut-set inequality induced by $S \cup T$ and by $S \cap T$. We then combine all four inequalities to a $\{0, \frac{1}{2}\}$-cut as described above. From the analysis in Section 1.3.4, we know that the resulting $\{0, \frac{1}{2}\}$-cut will be violated by our current solution u^* if and only if

- the sum of the right-hand sides of all four participating inequalities is odd and
- the sum of the slacks of the four participating inequalities is strictly less than one.

We only generate the $\{0, \frac{1}{2}\}$-cut if both conditions are satisfied; in particular we can disregard any cut-set inequality in our enumeration that has a slack of at least one at u^*.

In practice, it can be useful to limit the number of $\{0, \frac{1}{2}\}$-cuts that are separated in a single run of the algorithm to some fixed parameter $\ell \in \mathbb{Z}_{\geq 0}$. In this case, the algorithm should not return all $\{0, \frac{1}{2}\}$-cuts, but rather the ℓ "best" $\{0, \frac{1}{2}\}$-cuts that can be generated

from the cut-set inequalities in I. Since it is not clear how to measure the quality of a cut in general, we rely on a heuristic criterion and prefer those cuts that have a high violation.[1] Thus, we seek to generate the ℓ most violated $\{0, \frac{1}{2}\}$-cuts. The computation is again a heuristic: First, we group our cut-set inequalities in $d \in \mathbb{Z}_{>0}$ brackets according to their slack at u^*. Bracket $i \in \{0, \dots, d-1\}$ contains all cut-set inequalities whose slack lies in $[\frac{i}{d}, \frac{i+1}{d})$. We then start by combining only those inequalities that are in bracket 0; subsequently we combine bracket 0 with bracket 1 etc., until the enumeration is complete or ℓ violated cuts have been created. In this way, we hope to generate those $\{0, \frac{1}{2}\}$-cuts with a high violation first.

The resulting algorithm is a heuristic separation method as there is no guarantee that it will find all violated 3-partition inequalities. First, even if both (CS_S) and (CS_T) have a small slack, the slack of ($\mathrm{CS}_{S \cup T}$) and ($\mathrm{CS}_{S \cap T}$) can be arbitrarily large. Second, we cannot predict if the sum of the right-hand sides of (CS_S), (CS_T), ($\mathrm{CS}_{S \cup T}$) and ($\mathrm{CS}_{S \cap T}$) will be odd when we choose S and T. Finally, the set I generally contains only a subset of all cut-set inequalities: It can happen that a good $\{0, \frac{1}{2}\}$-cut could be generated from a cut-set inequality that is not part of the current linear programming relaxation.

In the sequel, we refer to this algorithm as `EnumZH`. It works independently of the description of $\mathfrak{B}$, but its worst-case running time is $O(|I|^2|E|)$ and thus depends quadratically on the number of cut-set inequalities in the current linear programming relaxation. In the next chapter, we will see how we can speed up the algorithm when $\mathfrak{B}$ is given as a finite list of vertices.

4.4.3 Shrinking Graphs and Lifting Facets

Agarwal [Aga06] considers the `mND` problem and in particular, the convex hull $\mathfrak{P}_{\texttt{mND}}(G, D)$ of all integer capacity vectors that are feasible for the `mND` instance (G, D). In a different perspective on k-partition inequalities, Agarwal defines partitions of the graph by iteratively merging two adjacent nodes into a single super node; we call this process *shrinking*. The process can be stopped when the shrunken graph has k nodes and then each node in the shrunken instance $(\overline{G}, \bar{D})$ defines a partition of the node set (consisting of all the nodes that have been merged into that node). Agarwal shows how to lift any inequality that is valid for $\mathfrak{P}_{\texttt{mND}}(\overline{G}, \bar{D})$ to be valid for $\mathfrak{P}_{\texttt{mND}}(G, D)$. Moreover, the lifting procedure maintains facets: if an inequality induces a facet of $\mathfrak{P}_{\texttt{mND}}(\overline{G}, \bar{D})$, then the lifted inequality induces a facet of $\mathfrak{P}_{\texttt{mND}}(G, \mathfrak{B})$.

Buchheim, Liers and Sanità [BLS11] observe that the same shrinking strategy works for the `sRND` problem and that the lifting still maintains validity. In the sequel, we show that the lifting also maintains facets of $\mathfrak{R}(G, \mathfrak{B})$. To this aim, we make an easy adaptation of Agarwal's [Aga06] proof. Following Agarwal's strategy, we then show that our 3-partition inequalities induce facets of $\mathfrak{R}(G, \mathfrak{B})$ by studying the complete graph on three nodes.

We first define what we mean by shrinking in more detail. Consider an `sRND` instance $(V, E, \mathfrak{B})$. In the beginning of our construction, G represents a partitioning of $V = V_1 \dot\cup \dots V_n$ into $n := |V|$ partitions and node $i \in V$ represents the partition $V_i = \{i\}$. For any edge $\{i^*, j^*\} \in E$, we say that we *contract* $\{i^*, j^*\}$ by merging i^* and j^* into a single super-node $\bar{p}$.

[1] A high violation can indicate that the cutting plane will cut "deeply" into the linear programming relaxation, removing a significant part of the fractional solutions.

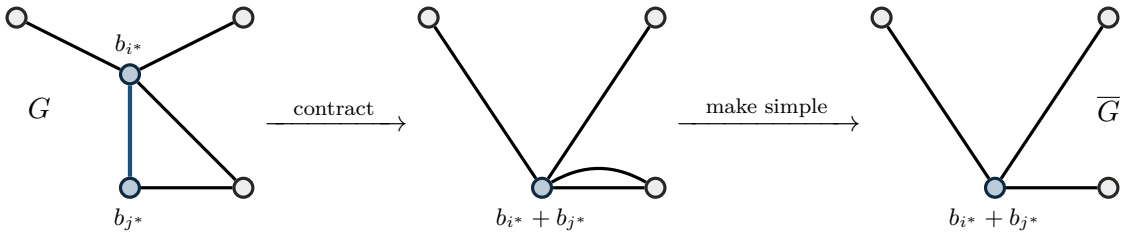


Figure 4.4: Building a shrunken instance by contracting the edge $\{i^*, j^*\}$.

We replace every edge $\{j^*, i\} \in E$ that is incident to j^* by an edge $\{p, i\}$; likewise, we replace any edge $\{i^*, i\} \in E$ by an edge $\{p, i\}$. If some node i is adjacent to both i^* and j^*, this procedure produces two parallel edges and we remove one of them to make sure that our graph remains simple. The new super node $\bar{p}$ now represents the partition $V_p := V_{i^*} \cup V_{j^*}$. To obtain a feasible scenario on the shrunken instance $\overline{G} = (\overline{V}, \overline{E})$ we set $\overline{b}_i = b_i$ if $i \in V \setminus \{i^*, j^*\}$ and $\overline{b}_p = b_{i^*} + b_{j^*}$ for all $b \in \mathfrak{B}$. We denote the resulting new scenario set as $\overline{\mathfrak{B}}$. Figure 4.4 shows the contraction step. Applying the contraction step once, we obtain a $(|V| - 1)$-partition of the original node set. By repeating the contraction step, however, we can contract edges until we have reduced any `sRND` instance $G = (V, E, \mathfrak{B})$ to a graph with any number $k \geq 1$ of nodes that induce a k-partition $V = V_1 \cup \ldots \cup V_k$. The resulting *shrunken* instance $\bar{G} = (\bar{V}, \bar{E}, \bar{\mathfrak{B}})$ is

$$\begin{aligned}
\bar{V} &:= \{\bar{1}, \ldots, \bar{k}\} \\
\bar{E} &:= \{\{\bar{p}, \bar{q}\} \mid (V_p : V_q)_G \neq \emptyset \text{ for } p, q = 1, \ldots, k, p < q\} \\
\bar{\mathfrak{B}} &:= \{(b_{\bar{1}}, \ldots, b_{\bar{k}}) = (\sum_{i \in V_1} b_i, \ldots, \sum_{i \in V_k} b_i) \mid b \in \mathfrak{B}\}.
\end{aligned}$$

In this way, an edge $\{\bar{p}, \bar{q}\}$ in the partitioning graph represents all edges in $(V_p : V_q)$ in the original graph. The balance of any node $\bar{p} \in \overline{V}$ is the total balance of V_p. Therefore, for any $p = 1, \ldots, \ell$, the maximum total balance $\max_{b \in \mathfrak{B}} |\sum_{i \in V_p} b_i|$ of partition V_p is exactly the maximum balance $\max_{\bar{b} \in \overline{\mathfrak{B}}} |\bar{b}_{\bar{p}}|$ of the shrunken node $\bar{p}$.

The shrinking is only useful if we can undo it after we found a valid inequality. We first define how we can undo a single contraction step and then describe what happens if a sequence of contraction steps is undone. Suppose that we started from an instance $(V, E, \mathfrak{B})$ and arrived at a shrunken instance $(\overline{V}, \overline{E}, \overline{\mathfrak{B}})$ by contracting an edge $\{i^*, j^*\}$ to a super node $\bar{p}$. In order to lift a valid inequality

$$\sum_{\{i,j\} \in \overline{E}} \overline{a}_{ij} \overline{u}_{ij} \geq B$$

for $\mathfrak{R}(\overline{G}, \overline{\mathfrak{B}})$ to $\mathfrak{R}(G, \mathfrak{B})$, we define a function $\lambda : \mathbb{R}^{\overline{E}} \to \mathbb{R}^E$ as

$$\lambda(\overline{a})_{ij} := \begin{cases} 0, & \text{if } \{i,j\} = \{i^*, j^*\} \\ \overline{a}_{pj}, & \text{if } \{i,j\} = \{i^*, j\} \in \delta(\{i^*, j^*\}) \\ \overline{a}_{pj}, & \text{if } \{i,j\} = \{j^*, j\} \in \delta(\{i^*, j^*\}) \\ \overline{a}_{ij}, & \text{otherwise} \end{cases}$$

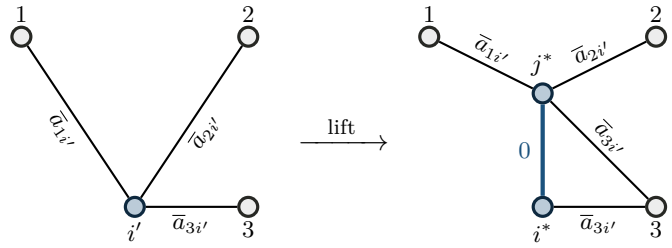


Figure 4.5: Lifting a vector from the shrunken instance back to the original instance.

Then, the lifted inequality is

$$\sum_{\{i,j\}\in E} \lambda(\overline{a})_{ij} u_{ij} \geq B.$$

This means that the contracted edge $\{i^*, j^*\}$ has a coefficient of 0 in the lifted inequality. Any edge that is adjacent to i^* or j^* and to some other node j inherits the coefficient of the corresponding edge $\{\bar{p}, j\}$ in the shrunken graph. Finally, all other edges are present in $\overline{E}$ and their coefficient is not changed by the lifting. The situation is depicted in Figure 4.5.

Suppose that we repeat the edge contractions until we have obtained a graph $\overline{G}$ with k partitions $V = V_1 \cup \ldots \cup V_k$, $k \in \{2, \ldots, |V|\}$. Then, we need to apply the lifting procedure $n - k$ times in order to lift a vector $\overline{a}$ from $\overline{G}$ back to a vector a on G. By the following induction it becomes clear that the lifted vector $a = \lambda^{n-k}(\overline{a})$ satisfies for any edge $\{i, j\} \in E$:

- $a_{ij} = \overline{a}_{pq}$, if $\{i, j\} \in (V_p : V_q)$ crosses between any two partitions $p \neq q$, and
- $a_{ij} = 0$ if $\{i, j\}$ completely lies within one partition p, i.e., $i \in V_p$ and $j \in V_p$.

The initial vector $\overline{a}$ satisfies these conditions: In $\overline{G}$, any edge $\{i, j\}$ starts in partition i, ends in a different partition j and has $\overline{a}_{ij}$ as its corresponding entry in $\overline{a}$. We argue that the same is true in $\lambda^\ell(\overline{a})$ for $\ell \geq 1$. By induction, $\lambda^{l-1}(\overline{a})$ satisfies both conditions and applying λ once maintains them: Consider an edge $\{i, j\}$ in the graph after ℓ lifting operations. If $\{i, j\}$ is the edge that is expanded by the lifting, then $\{i, j\}$ lies completely within some partition V_p (or represents a bundle of edges that lies completely within V_p) and we correctly set $\lambda^\ell(\overline{a})_{ij} = 0$. Otherwise, the edge $\{i, j\}$ inherits its value $\lambda^\ell(a)_{ij}$ from the edge that represented $\{i, j\}$ in $\overline{G}$. By induction, this value has been set correctly in a previous step.

Consequently, lifting the inequality

$$\sum_{\{\bar{p},\bar{q}\}\in\bar{E}} a_{\bar{p}\bar{q}} u_{\bar{p}\bar{q}} \geq B$$

from the shrunken graph $\overline{G}$ to the original graph G yields the lifted inequality

$$\sum_{\{\bar{p},\bar{q}\}\in\bar{E}} \Big(\sum_{\{i,j\}\in(V_p:V_q)} a_{\bar{p}\bar{q}} u_{ij} \Big) \geq B$$

and the lifting preserves validity. This was shown by Buchheim, Liers and Sanità.

Lemma 4.12 ([BLS11])**.** *Let $G = (V, E)$ be a connected, undirected graph and let $\mathfrak{B} \subseteq \mathbb{R}^V$ be a scenario set. Denote by $(\overline{V}, \overline{E}, \overline{\mathfrak{B}})$ the shrunken instance that is induced by a k-partition $V = V_1 \cup \ldots \cup V_k$. If the inequality*

$$\sum_{\{\bar{p},\bar{q}\}\in\overline{E}} a_{\bar{p}\bar{q}} u_{\bar{p}\bar{q}} \geq B$$

is valid for $\mathfrak{R}(\overline{G}, \overline{\mathfrak{B}})$, then the lifted inequality

$$\sum_{\{\bar{p},\bar{q}\}\in\overline{E}} \Big(\sum_{\{i,j\}\in(V_p:V_q)} a_{\bar{p}\bar{q}} u_{ij} \Big) \geq B$$

is valid for $\mathfrak{R}(G, \mathfrak{B})$.

Proof. We first define a *projecting* function π that projects any capacity vector $u \in \mathfrak{R}(G, \mathfrak{B})$ to $\pi(u) \in \mathfrak{R}(\overline{G}, \overline{\mathfrak{B}})$ by setting the capacity of $\{\bar{p}, \bar{q}\}$ to the sum of the capacities on $(V_p : V_q)$

$$\pi(u)_{\bar{p}\bar{q}} := \sum_{\{i,j\}\in(V_p:V_q)} u_{ij} \qquad \text{for all} \{\bar{p}, \bar{q}\} \in \bar{E}.$$

The projection $\pi(u)$ is well-defined, i.e., if $u \in \mathfrak{R}(G, \mathfrak{B})$, then indeed $\pi(u) \in \mathfrak{R}(\overline{G}, \overline{\mathfrak{B}})$ since for any $\overline{S} \subseteq \overline{V}$ and for $S = \bigcup_{\bar{p}\in\overline{S}} V_p$ we have

$$\sum_{\{\bar{p},\bar{q}\}\in\delta_{\overline{G}}(\overline{S})} \pi(u)_{\bar{p}\bar{q}} = \sum_{\{\bar{p},\bar{q}\}\in\delta_{\overline{G}}(\overline{S})} \sum_{\{i,j\}\in(V_p:V_q)_G} u_{ij} = \sum_{\{i,j\}\in\delta_G(S)} u_{ij} \geq \max_{b\in\mathfrak{B}} |\sum_{i\in S} b_i| = \max_{\bar{b}\in\overline{\mathfrak{B}}} |\sum_{\bar{p}\in\bar{S}} \bar{b}_{\bar{p}}|.$$

This means that $\bar{u}$ satisfies all cut-set inequalities of $(\overline{G}, \overline{\mathfrak{B}})$. Suppose now that (4.13) is valid for all $\bar{u} \in \mathfrak{R}(\overline{G}, \overline{\mathfrak{B}})$. We then have for any $u \in \mathfrak{R}(G, \mathfrak{B})$:

$$\sum_{\{\bar{p},\bar{q}\}\in\overline{E}} a_{\bar{p}\bar{q}} \sum_{\{i,j\}\in(V_p:V_q)} u_{ij} = \sum_{\{\bar{p},\bar{q}\}\in\overline{E}} a_{\bar{p}\bar{q}} \pi(u)_{\bar{p}\bar{q}} \geq B$$

and (4.13) is valid for $\mathfrak{R}(G, \mathfrak{B})$. □

Agarwal [Aga06] proves that for the `mND` problem, the lifting also preserves facets. The proof requires a worst-case capacity bound for the edges of the network and we adapt Agarwal's choice of this value to make it suitable for the `sRND` problem. This is the only modification that needs to be made to translate the proof. We repeat the complete proof here and mostly follow Agarwal's structure in doing so (we can omit Agarwal's technical global case distinction by using Lemma 1.1); we do, however, use some original pictures to illustrate technical parts of the proof. Thus, we can verify that the proof works for the `sRND` problem.

Theorem 4.13 ([Aga06])**.** *Let $G = (V, E)$ be a connected undirected graph and let $V = V_1 \cup \ldots \cup V_k$ be a partitioning of the node set V. For a scenario set $\mathfrak{B}$, let $(\overline{V}, \overline{E}, \overline{\mathfrak{B}})$ be the shrunken instance that is induced by $V_1, \ldots, V_k$. Moreover, let the inequality*

$$\sum_{\{\bar{p},\bar{q}\}\in\bar{E}} a_{\bar{p}\bar{q}} u_{\bar{p}\bar{q}} \geq B$$

induce a facet of the shrunken instance $\mathfrak{R}(\bar{G}, \bar{\mathfrak{B}})$*. Then, the lifted inequality*

$$\sum_{\{\bar{p},\bar{q}\}\in\bar{E}} \Big(\sum_{\{i,j\}\in(V_p:V_q)} a_{\bar{p}\bar{q}} u_{ij} \Big) \geq B$$

induces a facet of $\mathfrak{R}(G, \mathfrak{B})$ *if* $G[V_i]$ *is connected for all* $i = 1, \ldots, k$ *and if* $B > 0$*.*

Proof. We index the nodes of G as $V = \{1, \ldots, n\}$ and assume w.l.o.g. that the edge $\{n-1, n\}$ is present in G. We prove our claim for the case that $\overline{G}$ arises from G by contracting the edge $\{n-1, n\}$ (see Figure 4.6). By iteratively applying the argument, we can then conclude the proof. Suppose now that

$$\sum_{\{\bar{p},\bar{q}\}\in\bar{E}} a_{\bar{p}\bar{q}} u_{\bar{p}\bar{q}} \geq B$$

induces a facet of $\mathfrak{R}(G, \mathfrak{B})$, i.e., there are $\overline{m} := |\overline{E}|$ affinely independent vectors $\overline{u}^1, \ldots, \overline{u}^{\overline{m}} \in \mathfrak{R}(\overline{G}, \overline{\mathfrak{B}})$ with $\sum_{\{i,j\}\in\overline{E}} \overline{a}_{ij} \overline{u}^l_{ij} = B$ for all $l = 1, \ldots, \overline{m}$. Because $B > 0$, the vector $0 \in R^{\overline{E}}$ does not lie in the affine subspace spanned by $\overline{u}^1, \ldots, \overline{u}^{\overline{m}}$ and we get from Lemma 1.1 that the vectors $\overline{u}^1, \ldots, \overline{u}^{\overline{m}}$ are *linearly* independent, also. Using the vectors $\overline{u}^1, \ldots, \overline{u}^{\overline{m}}$ as rows, we obtain a $\overline{m} \times \overline{m}$ matrix $\overline{U}$ of rank $\overline{m}$; in particular, we can order the rows of $\overline{U}$ such that all diagonal elements are non-zero (if none of the $u^1, \ldots, u^{\overline{m}}$ has a non-zero entry in column j, then row j is 0).

We now need to find $m := |E|$ linearly independent vectors $u^1, \ldots, u^m \in \mathfrak{R}(G, \mathfrak{B})$ with

$$\sum_{\{i,j\}\in E} \lambda(\overline{a})_{ij} u^l_{ij} = B$$

for all $l = 1, \ldots, m$ (and again, we can use linear independence equivalently to affine independence by Lemma 1.1). For ease of notation, assume that the edges of $\overline{G}$ appear in a particular order in any $\overline{u} \in \mathfrak{R}(\overline{G}, \overline{\mathfrak{B}})$. We define $\Delta := \{i \in V \mid \{i, n-1\} \in E \wedge \{i, n\} \in E\}$ as the set of those nodes that are adjacent to both node n and node $n-1$. Thus, the edges in Δ are exactly the edges that will be combined into a single edge by the shrinking. The edges that are adjacent to n, but not to $n-1$ will be moved to $n-1$; the edge $\{n-1, n\}$ disappears by the contraction and all other edges will be left as they are. We suppose w.l.o.g. that $\overline{u}$ first contains the entries for the edges that are not affected by the shrinking, followed by the entries of the edges from $n-1$ to the set Δ.

$$\overline{u} = (\underbrace{*, \ldots, *}_{E\setminus(\Delta:n-1)}, \underbrace{*, \ldots, *}_{(\Delta:n-1)})$$

We will furthermore assume that the vectors $u \in \mathfrak{R}(G, \mathfrak{B})$ have an identical layout in the first m' coordinates and that the following coordinates correspond to the edges from n to Δ. Finally, we assume that the last coordinate corresponds to the shrunken edge.

$$u = (\underbrace{*, \ldots, *}_{E\setminus(\Delta:\{n-1,n\})}, \underbrace{*, \ldots, *}_{(\Delta:n-1)}, \underbrace{*, \ldots, *}_{(\Delta:n)}, \underbrace{*}_{\{n-1,n\}})$$

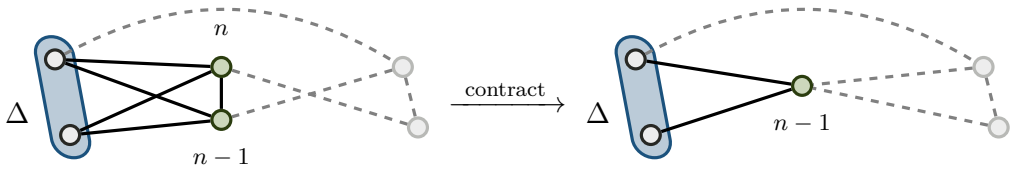


Figure 4.6: Shrinking the edge $\{n-1, n\}$. The set Δ contains all nodes that are adjacent to $n-1$ and n.

We can now build the first $\overline{m}$ linearly independent vectors by extending $\overline{u}^1, \ldots, \overline{u}^{\overline{m}}$. To this aim, let $B^* = \max_{b \in \mathfrak{B}} \sum_{i \in V} |b_i|$ as before.

$$u^l := (\underbrace{\quad \overline{u}^i \quad}_{E \setminus (\Delta:\{n-1,n\}),\ (\Delta:n-1)}, \underbrace{0, \ldots, 0}_{(\Delta:n)}, \underbrace{B^*}_{\{n-1,n\}}) \qquad l = 1, \ldots, \overline{m}$$

These vectors are feasible: They can only be infeasible because of edges adjacent to n or $n-1$. Thus, suppose that in some scenario $\overline{b}$ the flow crosses some edge $\{i, n-1\}$ with $i \in \Delta$ and continues to some node $j \notin \Delta$ via an edge $\{n-1, j\}$. By our extension, we force the flow to use the edge $\{i, n-1\}$ in G also; and because the nodes in Δ are adjacent to both n and $n-1$, this edge is present and it has the same capacity as in $\overline{G}$. Now, it can happen that j is adjacent n, but not adjacent to $n-1$ in G. In this case, we can reroute the flow via $\{n-1, n\}$ to n and because B^* is an upper bound on the necessary capacity of any edge, this can always be done. For a more formal proof, we could equivalently check that u^l satisfies all cut-set inequalities. We observe at this point that for all $l = 1, \ldots, \overline{m}$:

$$\begin{aligned}
\sum_{\{i,j\} \in E} a_{ij} u^l_{ij} &= \sum_{\{i,j\} \in E \setminus (\Delta:\{n-1,n\})} \overline{a}_{ij} \overline{u}^l_{ij} + \sum_{\{i,j\} \in (\Delta:\{n-1\})} \overline{a}_{ij} \overline{u}^l_{ij} + \sum_{\{i,j\} \in (\Delta:\{n\})} \overline{a}_{ij} \cdot 0 + 0 \cdot B^* \\
&= \sum_{\{i,j\} \in \overline{E}} \overline{a}_{ij} \overline{u}^l_{ij} \\
&= B.
\end{aligned}$$

For the next $|\Delta|$ vectors, consider the lower right $|\Delta| \times |\Delta|$ submatrix $\overline{U}'$ spanned by the last $|\Delta|$ entries of the last $|\Delta|$ rows of $\overline{U}$ (i.e., of the vectors $\overline{u}^{\overline{m}-|\Delta|+1}, \ldots, \overline{u}^{\overline{m}}$). Because the last $|\Delta|$ entries of the $\overline{u}^{\overline{m}-|\Delta|+1}, \ldots, \overline{u}^{\overline{m}}$ correspond[2] to the edges from Δ to the node $n-1$, we know that the l-th diagonal entry d_l in $\overline{U}'$ corresponds to the l-th edge in $(\Delta : n-1)$. By our initial assumption about the ordering of the rows in $\overline{U}$, we also know that $d_l > 0$. For $l = 1, \ldots, |\Delta|$, we can now construct a new vector $u^{\overline{m}+l}$ from $\overline{u}^{\overline{m}-|\Delta|+l}$. The first $\overline{m}$ entries of $u^{\overline{m}+l}$ and $\overline{u}^{m-|\Delta|+l}$ correspond to $E \setminus (\Delta : \{n-1, n\})$ and to $(\Delta : \{n-1\})$. We choose them to be identical in $u^{\overline{m}+l}$ and $\overline{u}^{m-|\Delta|+l}$, except for the entry corresponding to the l-th edge in $(\Delta : n-1)$ which we set to zero. The following $|\Delta|$ entries in $u^{\overline{m}-|\Delta|+l}$ correspond to $(\Delta : n)$ and we set them to zero, with the exception of the entry corresponding to the l-th edge in $(\Delta : n)$; we set this entry to d_l. We set the last entry (corresponding to the

[2] Observe that $|(\Delta : n-1)| = |\Delta|$.

edge $\{n-1,n\}$) to B^*. Summarizing, the situations is thus:

$$
\begin{aligned}
u^{\overline{m}+1} &:= (\overbrace{\underbrace{*,\ldots,*}_{E\setminus(\Delta:\{n-1,n\})}, \underbrace{0,*,*,\ldots,*}_{(\Delta:n-1)}}^{\overline{u}^{\overline{m}-|\Delta|+1}}, \underbrace{d_1,0,0\ldots,0}_{(\Delta:n)}, \underbrace{B^*}_{\{n-1,n\}})\\
u^{\overline{m}+2} &:= (\overbrace{\underbrace{*,\ldots,*}_{E\setminus(\Delta:\{n-1,n\})}, \underbrace{*,0,*,\ldots,*}_{(\Delta:n-1)}}^{\overline{u}^{\overline{m}-|\Delta|+2}}, \underbrace{0,d_2,0,\ldots,0}_{(\Delta:n)}, \underbrace{B^*}_{\{n-1,n\}})\\
&\vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad \vdots\\
u^{\overline{m}+|\Delta|} &:= (\overbrace{\underbrace{*,\ldots,*}_{E\setminus(\Delta:\{n-1,n\})}, \underbrace{*,*,*,\ldots,0}_{(\Delta:n-1)}}^{\overline{u}^{\overline{m}-|\Delta|+|\Delta|}}, \underbrace{0,0,0,\ldots,d_{|\Delta|}}_{(\Delta:n)}, \underbrace{B^*}_{\{n-1,n\}})
\end{aligned}
$$

These vectors are again feasible: Any flow that uses the l-th edge $\{i,n-1\}$ of $(\Delta:n-1)$ in $\overline{G}$ can be rerouted via $\{i,n\}$ and $\{n-1,n\}$ as in the previous case. Also, by the same argument as before, the u^l satisfy $\sum_{\{i,j\}\in E} a_{ij}u^l_{ij} = B$ for all $l = \overline{m}+1,\ldots,\overline{m}+|\Delta|$.

In total, we now have $\overline{m}+|\Delta|$ vectors, while we would need $|E| = \overline{m}+|\Delta|+1$ many. We obtain the missing vector $u^{\overline{m}+|\Delta|+1}$ by changing the last entry of u^1 to B^*+1.

$$
u^{\overline{m}+|\Delta|+1} := (\underbrace{\overline{u}^1}_{E\setminus(\Delta:\{n-1,n\}),\ (\Delta:n-1)}, \underbrace{0,\ldots,0}_{(\Delta:n)}, \underbrace{B^*+1}_{\{n-1,n\}})
$$

This vector is feasible and again satisfies $\sum_{\{i,j\}\in E} a_{ij}u^{\overline{m}+|\Delta|+1}_{ij} = B$. By construction the vectors are linearly independent and this concludes our proof. □

4.4.4 3-Partition Inequalities Induce Facets

We can now proceed to show that 3-partition inequalities induce facets of $\mathfrak{R}(G,\mathfrak{B})$ if they are induced by connected partitions. To this aim, we suppose that an arbitrary graph $G=(V,E)$ has been shrunken to the triangle graph G^3 (the complete graph on three nodes) whose three nodes σ,τ,υ represent the partitions $S\cup T\cup U = V$ and whose edges $\{\sigma,\tau\}$, $\{\sigma,\upsilon\}$, $\{\tau,\upsilon\}$ represent the three cuts $(S:T)$, $(S:U)$ and $(T:U)$, respectively. We show that for any choice of $\mathfrak{B}$, the resulting 3-partition inequality

$$
u_{\sigma\tau} + u_{\sigma\upsilon} + u_{\tau\upsilon} \geq \left\lceil \frac{R_S+R_T+R_U}{2} \right\rceil \tag{4.6}
$$

defines a facet of $\mathfrak{R}(G^3,\mathfrak{B}^3)$ if and only if $R_S+R_T+R_U$ is odd (which, in particular, implies that $R_S+R_T+R_U>0$) and all three partitions induce connected subgraphs.

As the dimension of $\mathfrak{R}(G^3,\mathfrak{B}^3)$ is 3, we have to show that for any choice of $\mathfrak{B}$ and of S,T and U, there are always three integer feasible, linearly independent solutions that all

satisfy (4.6) with equality. We claim that the following capacity assignment works

$$
\begin{aligned}
u^1_{\sigma\tau} &:= \left\lfloor \frac{R_S + R_T - R_U}{2} \right\rfloor & u^1_{\sigma\upsilon} &:= \left\lceil \frac{R_S + R_U - R_T}{2} \right\rceil & u^1_{\tau\upsilon} &:= \left\lceil \frac{R_T + R_U - R_S}{2} \right\rceil \\
u^2_{\sigma\tau} &:= \left\lceil \frac{R_S + R_T - R_U}{2} \right\rceil & u^2_{\sigma\upsilon} &:= \left\lfloor \frac{R_S + R_U - R_T}{2} \right\rfloor & u^2_{\tau\upsilon} &:= \left\lceil \frac{R_T + R_U - R_S}{2} \right\rceil \\
u^3_{\sigma\tau} &:= \left\lceil \frac{R_S + R_T - R_U}{2} \right\rceil & u^3_{\sigma\upsilon} &:= \left\lceil \frac{R_S + R_U - R_T}{2} \right\rceil & u^3_{\tau\upsilon} &:= \left\lfloor \frac{R_T + R_U - R_S}{2} \right\rfloor.
\end{aligned}
$$

Our key argument in the following analysis is that the sum $\pm R_S \pm R_T \pm R_U$ has the same parity independently of the sign of $\pm R_S$, $\pm R_T$ and $\pm R_U$; in particular, either we round in *all* of $u^\ell_{\sigma\tau}$, $u^\ell_{\sigma\upsilon}$ and $u^\ell_{\tau\upsilon}$, $\ell = 1, 2, 3$, or for *none* of them.

To check that our assignment is feasible, we verify that it satisfies all cut-set constraints on $(G^3, \mathfrak{B}^3)$. For all three vectors u^1, u^2 and u^3, we check the cut-set inequalities induced by $\{\sigma\}$, $\{\tau\}$ and $\{\upsilon\}$. We do not need to check the cut-sets of size two because their complement is one of $\{\sigma\}$, $\{\tau\}$ and $\{\upsilon\}$.

$$
\begin{aligned}
u^1_{\sigma\tau} + u^1_{\sigma\upsilon} &= \left\lfloor \frac{R_S + R_T - R_U}{2} \right\rfloor + \left\lceil \frac{R_S + R_U - R_T}{2} \right\rceil && (\mathrm{CS}_{\{\sigma\}}) \\
&\overset{obs.}{=} \frac{R_S + R_T - R_U}{2} + \frac{R_S + R_U - R_T}{2} = R_S
\end{aligned}
$$

$$
\begin{aligned}
u^1_{\sigma\tau} + u^1_{\tau\upsilon} &= \left\lfloor \frac{R_S + R_T - R_U}{2} \right\rfloor + \left\lceil \frac{R_T + R_U - R_S}{2} \right\rceil && (\mathrm{CS}_{\{\tau\}}) \\
&\overset{obs.}{=} \frac{R_S + R_T - R_U}{2} + \frac{R_T + R_U - R_S}{2} = R_T
\end{aligned}
$$

$$
\begin{aligned}
u^1_{\sigma\upsilon} + u^1_{\tau\upsilon} &= \left\lceil \frac{R_S + R_U - R_T}{2} \right\rceil + \left\lceil \frac{R_T + R_U - R_S}{2} \right\rceil && (\mathrm{CS}_{\{\upsilon\}}) \\
&\geq \frac{R_S + R_U - R_T}{2} + \frac{R_T + R_U - R_S}{2} = R_U
\end{aligned}
$$

$$
\begin{aligned}
u^2_{\sigma\tau} + u^2_{\sigma\upsilon} &= \left\lceil \frac{R_S + R_T - R_U}{2} \right\rceil + \left\lfloor \frac{R_S + R_U - R_T}{2} \right\rfloor && (\mathrm{CS}_{\{\sigma\}}) \\
&\overset{obs.}{=} \frac{R_S + R_T - R_U}{2} + \frac{R_S + R_U - R_T}{2} = R_S
\end{aligned}
$$

$$
\begin{aligned}
u^2_{\sigma\tau} + u^2_{\tau\upsilon} &= \left\lceil \frac{R_S + R_T - R_U}{2} \right\rceil + \left\lceil \frac{R_T + R_U - R_S}{2} \right\rceil && (\mathrm{CS}_{\{\tau\}}) \\
&\geq \frac{R_S + R_T - R_U}{2} + \frac{R_T + R_U - R_S}{2} = R_T
\end{aligned}
$$

$$
\begin{aligned}
u^2_{\sigma\upsilon} + u^2_{\tau\upsilon} &= \left\lfloor \frac{R_S + R_U - R_T}{2} \right\rfloor + \left\lceil \frac{R_T + R_U - R_S}{2} \right\rceil && (\mathrm{CS}_{\{\upsilon\}}) \\
&\overset{obs.}{=} \frac{R_S + R_U - R_T}{2} + \frac{R_T + R_U - R_S}{2} = R_U
\end{aligned}
$$

$$u^3_{\sigma\tau} + u^3_{\sigma\upsilon} = \left\lceil \frac{R_S + R_T - R_U}{2} \right\rceil + \left\lfloor \frac{R_S + R_U - R_T}{2} \right\rfloor \qquad (\mathrm{CS}_{\{\sigma\}})$$
$$\overset{obs.}{=} \frac{R_S + R_T - R_U}{2} + \frac{R_S + R_U - R_T}{2} = R_S$$

$$u^3_{\sigma\tau} + u^3_{\tau\upsilon} = \left\lceil \frac{R_S + R_T - R_U}{2} \right\rceil + \left\lceil \frac{R_T + R_U - R_S}{2} \right\rceil \qquad (\mathrm{CS}_{\{\tau\}})$$
$$\geq \frac{R_S + R_T - R_U}{2} + \frac{R_T + R_U - R_S}{2} = R_T$$

$$u^3_{\sigma\upsilon} + u^3_{\tau\upsilon} = \left\lfloor \frac{R_S + R_U - R_T}{2} \right\rfloor + \left\lceil \frac{R_T + R_U - R_S}{2} \right\rceil \qquad (\mathrm{CS}_{\{\upsilon\}})$$
$$\overset{obs.}{=} \frac{R_S + R_U - R_T}{2} + \frac{R_T + R_U - R_S}{2} = R_U$$

If $R_S + R_T + R_U$ is odd, the vectors u^1, u^2, u^3 are linearly independent[3] and for all $l = 1, 2, 3$, our assignment u^l satisfies (4.6) with equality.

$$\begin{aligned}
u^1_{\sigma\tau} + u^1_{\sigma\upsilon} + u^1_{\tau\upsilon} &= \frac{R_S + R_T - R_U}{2} - \frac{1}{2} + \frac{R_S + R_U - R_T}{2} + \frac{1}{2} + \frac{R_T + R_U - R_S}{2} + \frac{1}{2} \\
&= \frac{R_S + R_T + R_U}{2} + \frac{1}{2} = \left\lceil \frac{R_S + R_T + R_U}{2} \right\rceil. \\
u^2_{\sigma\tau} + u^2_{\sigma\upsilon} + u^2_{\tau\upsilon} &= \frac{R_S + R_T - R_U}{2} + \frac{1}{2} + \frac{R_S + R_U - R_T}{2} - \frac{1}{2} + \frac{R_T + R_U - R_S}{2} + \frac{1}{2} \\
&= \frac{R_S + R_T + R_U}{2} + \frac{1}{2} = \left\lceil \frac{R_S + R_T + R_U}{2} \right\rceil. \\
u^3_{\sigma\tau} + u^3_{\sigma\upsilon} + u^3_{\tau\upsilon} &= \frac{R_S + R_T - R_U}{2} + \frac{1}{2} + \frac{R_S + R_U - R_T}{2} + \frac{1}{2} + \frac{R_T + R_U - R_S}{2} - \frac{1}{2} \\
&= \frac{R_S + R_T + R_U}{2} + \frac{1}{2} = \left\lceil \frac{R_S + R_T + R_U}{2} \right\rceil.
\end{aligned}$$

Therefore, the inequality (4.6) induces a facet of $\mathfrak{R}(G^3, \mathfrak{B}^3)$. We can now apply Agarwal's lifting Theorem 4.13.

Theorem 4.14. *Let $G = (V, E)$ be a connected, undirected graph with a scenario set $\mathfrak{B} \subseteq \mathbb{R}^V$. Furthermore, let $S \cup T \cup U = V$ be a partitioning of the node set. Then the 3-partition inequality*

$$\sum_{\{i,j\} \in (S:T)} u_{ij} \; + \sum_{\{i,j\} \in (S:U)} u_{ij} \; + \sum_{\{i,j\} \in (T:U)} u_{ij} \; \geq \; \left\lceil \frac{R_S + R_T + R_U}{2} \right\rceil$$

given by the partitions S, T and U induces a facet of $\mathfrak{R}(G, \mathfrak{B})$ if

1. *the induced subgraphs $G[S], G[T]$ and $G[U]$ are connected and*
2. *$R_S + R_T + R_U > 0$ is odd.*

□

[3] Otherwise, we know from our analysis in Section 1.3.4 that the corresponding 3-partition inequality cannot be a facet; and indeed, if $R_A + R_B + R_C$ is even, then the vectors u^1, u^2 and u^3 are identical and the proof breaks down.

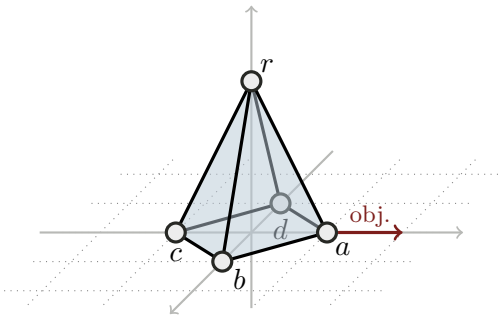


Figure 4.7: A 3-dimensional pyramid as an example for a degenerate polytope from [Grö04]. The vertex r is degenerate as it lies in the intersection of four of the pyramid's facets. Removing any of these facets changes the polytope. In red: The normal vector of the objective function. The optimum is attained at the vertex a in this example.

4.5 Degeneracy

A standard approach to solving linear programs $\min\{c^{\mathrm{T}}x \mid Ax \geq b\}$ (where $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$) is to use the Simplex algorithm by Dantzig [Dan51]. On a high level, the algorithm selects n rows $I \subseteq \{1,\dots,m\}$ of A and solves the system $A_{I*}x = b_I$, thus intersecting the hyperplanes which bound $A^T_{i*}x \geq b_i$ for $i \in I$. If A_{I*} has full rank (we ignore the other case in this high-level description), then the unique solution x^* is a vertex of the polyhedron $\mathfrak{A}$ spanned by $Ax \geq b$. The algorithm then does a pivoting step by swapping a row $i \in I$ for a row $j \notin I$ and obtains a new set $J = I \setminus \{i\} \cup \{j\}$. It then resolves $A_{J*}x \geq b_J$ for a new x^* and iterates until all pivoting steps would increase the objective value of x^*. The pivoting is done such that the objective value of x^* can never increase, however, it can happen that several different choices of I yield the same vertex x^*. In this case, we say that x^*, and thus $\mathfrak{A}$, is degenerate.

The Simplex algorithm can get stuck at degenerate vertices and therefore, they can have a bad impact on the algorithm's performance. One such example is depicted in Figure 4.7. There, the optimum solution of the problem is attained at the vertex a. Suppose that the Simplex algorithm is currently at the vertex c, having selected the hyperplanes spanned by $\{a,b,c,d\}$, by $\{b,c,r\}$ and by $\{c,d,r\}$. From there, it can reach the vertices b, d and r. All three vertices have the same objective value. If the algorithms keeps the hyperplane $\{a,b,c,d\}$ and chooses b or d, it can reach the optimum vertex a within one additional pivoting step. Alternatively, it can replace $\{a,b,c,d\}$ with $\{a,b,r\}$ or $\{a,d,r\}$, thus changing to vertex r. Then, however, two additional pivoting steps are necessary to reach vertex a.

We show next that $\mathfrak{R}(G,\mathfrak{B})$ can be *highly* degenerate as there are simple instances $(G,\mathfrak{B})$ where $\Omega(2^{\dim \mathfrak{R}(G,\mathfrak{B})}/2)$ facets of $\mathfrak{R}(G,\mathfrak{B})$ intersect in a single vertex.

Consider the instance $(G^d,\{b^d\})$ in Figure 4.8. In its unique scenario b^d, it has a unique source s with balance $b^d_s = d$. Any node i that is adjacent to s has a balance of $b^d_i = -1$, $i = 1,\dots,d$ and the auxiliary node a has zero balance $b^d_a = 0$. The instance has exactly $2d$ edges and thus, the dimension of $\mathfrak{R}(G^d,\{b^d\})$ is $2d$. The vector u^* with $u^*_{si} = 1$ and $u^*_{it} = 0$ for all $i = 1,\dots,d$ is a feasible solution for $(G^d,\{b^d\})$ and in fact, it is a vertex of $\mathfrak{R}(G^d,\{b^d\})$: Given any set $X \subseteq T := \{1,\dots,d\}$, the vector u^* satisfies the cut-set

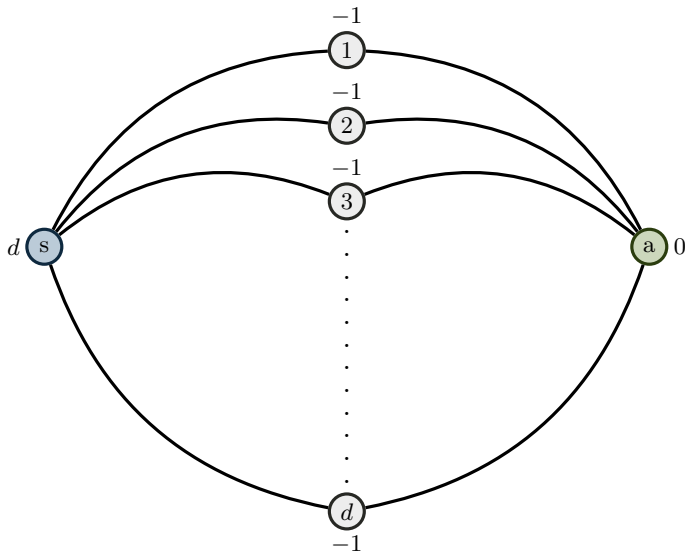


Figure 4.8: An instance with a degenerate polytope.

inequality

$$\sum_{\{i,j\}\in\delta(\{s\}\cup X)} u^*_{ij} = \sum_{i\in X} u^*_{ia} + \sum_{i\in T\setminus X} u^*_{si} = 0 + |T| - |X| = d - |X| = \sum_{i\in\{s\}\cup X} b^d_i$$

induced by $\{s\} \cup X$ with equality. Because both the subgraph induced by $\{s\} \cup X$ and the one induced by its complement $\{a\} \cup (T \setminus X)$ are connected, this cut-set inequality induces a facet if and only if the total balance of $\{s\} \cup X$ is non-zero. This is true if and only if $X \neq T$. Thus, there remain $2^d - 1$ facet-inducing choices for X and we have at least $2^d - 1 = 2^{\dim \mathfrak{R}(G^d,\{b^d\})/2} - 1$ facets that intersect in u^*.

Lemma 4.15. *For any $d \geq 1$ there is an instance $(G^d, \mathfrak{B}^d)$ of the* `sRND` *problem such that the corresponding polytope $\mathfrak{R}(G^d, \mathfrak{B}^d)$ has dimension $2d$ and such that there is a vertex of $\mathfrak{R}(G^d, \mathfrak{B}^d)$ in which $\Omega(2^{\dim \mathfrak{R}(G^d,\mathfrak{B}^d)/2})$ facets of $\mathfrak{R}(G^d, \mathfrak{B}^d)$ intersect.* □

Suppose that our degenerate instance arises from a larger instance $(G', \{b'\})$ by a series of edge contractions. Then, by Agarwal's Lifting Theorem 4.13, all the facets intersecting in u^* are also facets of $\mathfrak{R}(G', \{b'\})$ and they intersect in the lifted vector corresponding to u^*. This means that if an `sRND` instance has a parallel structure like the one in Figure 4.8 as a minor – and if it has the appropriate scenario structure – it is degenerate, too.

Chapter 5

Separation Under Uncertainty

From Chapter 4 we know that if a vector $u^* \in \mathbb{R}^E_{\geq 0}$ is infeasible for a sRND instance $(V, E, \mathfrak{B})$, then there is a violated cut-set inequality and a cut-set S certifying the infeasibility. In this chapter, we will see how to find such a certificate cut-set. This will enable us to decide the feasibility of u^*, to solve the separation problem for the cut-set inequalities and, in particular, to solve the linear programming relaxation of our capacity formulation (4.3) of the sRND problem without having to enumerate all of its exponentially many cut-set constraints. We show that the separation problem for the sRND-F problem is polynomial time solvable while the separation problem (and thus, deciding feasibility) for the sRND-P problem is $\mathcal{NP}$-hard. The latter holds true even for basic scenario polytopes that essentially consist of box constraints. Accordingly, we introduce a MIP-separation algorithm for this case. Finally, we show a heuristic that separates 3-partition inequalities in the sRND-F case. The results in this Chapter were obtained in a collaboration with Eduardo Álvarez-Miranda, Valentina Cacchiani, Tim Dorneth, Michael Jünger, Frauke Liers, Andrea Lodi and Tiziano Parriani. They have been published in [ACDJ+12] and [CJL+14].

5.1 The General Cut-Set Separation Problem

In general, the size of our cut-set formulation (4.3) is exponential in the encoding size of an sRND instance $(G, \mathfrak{B})$. This remains true even if we use Theorem 4.10 and only include the cut-set inequalities for all *strong* cut-sets $S \subseteq V$. Thus, in order to solve the linear programming relaxation of (4.3), we need a separation algorithm for cut-set inequalities.

We assume in the sequel that $G = (V, E)$ is a connected, undirected graph and that $\mathfrak{B} \subseteq \mathbb{R}^V$ is a scenario set. A vector $u^* \in \mathbb{R}^E_{\geq 0}$ satisfies all cut-set inequalities of an sRND instance $(G, \mathfrak{B})$ if and only if

$$\min_{S \subseteq V} \Big[\sum_{\{i,j\} \in \delta(S)} u^*_{ij} \; - \; \max_{b \in \mathfrak{B}} \big| \sum_{i \in S} b_i \big| \Big] \geq 0. \tag{5.1}$$

Thus, problem (5.1) is the separation problem for cut-set inequalities: Any solution (S^*, b^*) with negative objective value induces a violated cut-set inequality. On the other hand, if the optimum solution of (5.1) is non-negative, no violated cut-set inequality exists. Still, the absolute value in the inner sum of the problem makes the further analysis needlessly complicated and we can remove it without changing the problem. This is (basically) possible because the sets S and $V \setminus S$ induce the same cut-set inequality.

Theorem 5.1. *Let $G = (V, E)$ be a connected, undirected graph and let $\mathfrak{B} \subseteq \mathbb{R}^V$ be a scenario set. A vector $u^* \in \mathbb{R}_{\geq 0}$ is fractionally feasible for the* sRND *instance $(G, \mathfrak{B})$ if and only if*

$$\min_{S \subseteq V} \Big[\sum_{\{i,j\} \in \delta(S)} u^*_{ij} \; - \; \max_{b \in \mathfrak{B}} \sum_{i \in S} b_i \Big] \tag{5.2}$$

is non-negative. In any any optimum solution (S, b) for (5.2) we have $\sum_{i \in S} b_i \geq 0$.

Proof. For the main part of the theorem, let $u^* \in \mathbb{R}^E_{\geq 0}$ be a fractionally feasible capacity vector for $(G, \mathfrak{B})$. Then, the vector u^* also satisfies all cut-set inequalities and the first direction of our claim follows:

$$0 \leq \min_{S \subseteq V} \Big[\sum_{\{i,j\} \in \delta(S)} u^*_{ij} \; - \; \max_{b \in \mathfrak{B}} \big| \sum_{i \in S} b_i \big| \Big] \leq \min_{S \subseteq V} \Big[\sum_{\{i,j\} \in \delta(S)} u^*_{ij} \; - \; \max_{b \in \mathfrak{B}} \sum_{i \in S} b_i \Big]$$

For the other direction, suppose that $u^* \in \mathbb{R}_{\geq 0}$ is an arbitrary vector and that we have

$$\min_{S \subseteq V} \Big[\sum_{\{i,j\} \in \delta(S)} u^*_{ij} \; - \; \max_{b \in \mathfrak{B}} \sum_{i \in S} b_i \Big] \geq 0.$$

We have to show that u^* satisfies the cut-set inequality induced by any $S \subseteq V$. Let $b^* \in \mathfrak{B}$ be a worst-case scenario for S. By Observation 4.6, the cut-set inequality induced by S is the same as the one induced by $V \setminus S$ and therefore, we can assume without loss of generality that $\sum_{i \in S} b_i \geq 0$ holds (otherwise, we have $\sum_{i \in V \setminus S} b_i \geq 0$ because b is balanced). It follows that

$$\sum_{\{i,j\} \in \delta(S)} u^*_{ij} - \max_{b \in \mathfrak{B}} \big| \sum_{i \in S} b_i \big| \;\; = \sum_{\{i,j\} \in \delta(S)} u^*_{ij} - \sum_{i \in S} b^*_i \;\; \geq \sum_{\{i,j\} \in \delta(S)} u^*_{ij} - \max_{b \in \mathfrak{B}} \sum_{i \in S} b_i \geq 0.$$

For the last part of the theorem, assume that (S, b) is an optimum solution for (5.2) with $\sum_{i\in S} b_i < 0$. Since b is balanced, we have $\sum_{i\in V\setminus S} b_i = -\sum_{i\in S} b_i > 0 > \sum_{i\in S} b_i$. Together with the fact that $\delta(S) = \delta(V \setminus S)$ it follows that

$$\sum_{\{i,j\}\in\delta(S)} u^*_{ij} - \sum_{i\in S} b_i > \sum_{\{i,j\}\in\delta(V\setminus S)} u^*_{ij} - \sum_{i\in V\setminus S} b_i.$$

Thus, the solution $(V \setminus S, b)$ has a better objective value than (S, b). This is a contradiction to the optimality of (S, b). □

The last part of Theorem 5.1 motivates the following definition.

Definition 5.2. *For a connected, undirected graph $G = (V, E)$ and a Hose polytope $\mathfrak{H}(V, b^{min}, b^{max})$, let $S \subseteq V$ be a cut-set. We say that $b \in \mathfrak{H}(V, b^{min}, b^{max})$ is **relevant** for S if $\sum_{i\in S} b_i \geq 0$.*

Starting from the problem formulation (5.3), we can model the separation problem as a quadratic program. The interpretation of the variables is that $x_i = 1$ if and only if $i \in S$, for all $i \in V$. Likewise, we have $y_{ij} = 1$ if and only $\{i, j\} \in \delta(S)$, for all $\{i, j\} \in E$. The variables given by b select a worst-case scenario for S.

$$\begin{aligned} \min \quad & \sum_{\{i,j\}\in E} u^*_{ij} y_{ij} - \sum_{i\in V} x_i b_i && && (5.3)\\ \text{s.t.} \quad & x_i - x_j \geq y_{ij} && \text{for all } \{i,j\} \in E \\ & x_j - x_i \geq y_{ij} && \text{for all } \{i,j\} \in E \\ & b \in \mathfrak{B} \\ & y \in \{0,1\}^E \\ & x \in \{0,1\}^V \end{aligned}$$

In any optimum solution of program (5.3), the second term of the objective function will be as large as possible. We can therefore omit the explicit maximization.

5.2 General Separation Methods for Robust Linear Programs

We briefly discuss if the general `sRND` cut-set separation problem can be solved with methods from the literature. Here, however, the problem is that already the non-robust counterpart of our cut-set formulation (4.3) has exponential size and must be solved with a separation algorithm. None of the standard algorithms for robust linear programs in Section 2.4 can cope with this situation: As discussed in Chapter 4, Soyster's [Soy73] approach yields our cut-set formulation, but does not give us a way to separate cut-set inequalities.

The general approach by Ben-Tal and Nemirovski [BTN99] does give us a separation algorithm, but its running time depends linearly on the number of rows of the non-robust problem formulation: In order to separate a vector $u^* \in \mathbb{R}^E_{\geq 0}$, it requires to solve the problem

$$\min_{b\in\mathfrak{B}} \sum_{\{i,j\}\in\delta(S)} u^*_{ij} - \Big|\sum_{i\in S} b_i\Big| \qquad (5.4)$$

for every fixed $S \subseteq V$, which is equivalent to computing the correct right-hand side for all cut-set inequalities. In that sense, this is a *brute-force* separation algorithm that works by enumerating all cut-sets and in order to avoid the full enumeration, we would need to find a pair (S, b) such that (5.4) becomes negative. This task, however, leads back to our original separation problem (5.3).

Bertsimas and Sim [BS03] propose to reformulate the robust part of the problem formulation. We could hope to thus obtain a new formulation with constraints that are easier to separate. Unfortunately, that is not the case. Applied here, Bertsimas' and Sim's approach consists in replacing the right-hand side optimization problem

$$\begin{aligned} \max \quad & \sum_{i \in S} b_i && && (5.5) \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} b_i \geq r_i && \text{for all } i = 1, \ldots, k \\ & b_i \in \mathbb{R} && \text{for all } j = 1, \ldots, n \end{aligned}$$

by its dual program

$$\begin{aligned} \min \quad & \sum_{i=1}^{k} r_i y_i && && (5.6) \\ \text{s.t.} \quad & \sum_{i=1}^{k} a_{ij} y_i = 1 && \text{for all } j \in S \\ & \sum_{i=1}^{k} a_{ij} y_i = 0 && \text{for all } j \in V \setminus S \\ & y_i \leq 0 && \text{for all } i = 1, \ldots, m \end{aligned}$$

in each cut-set inequality. The resulting formulation, however, has an exponential number of variables in addition to an exponential number of constraints because we need a full set of dual variables for each cut-set inequality.

$$\begin{aligned} \min \quad & \sum_{\{i,j\} \in E} c_{ij} u_{ij} && && (5.7) \\ \text{s.t.} \quad & \sum_{\{i,j\} \in \delta(S)} u_{ij} \geq \sum_{i=1}^{k} r_i y_i^S && \text{for all } S \subseteq V \\ & \sum_{i=1}^{k} a_{ij} y_i^S = 1 && \text{for all } j \in S \text{ and all } S \subseteq V \\ & \sum_{i=1}^{k} a_{ij} y_i^S = 0 && \text{for all } j \in V \setminus S \text{ and all } S \subseteq V \\ & y_i^S \leq 0 && \text{for all } i = 1, \ldots, m \text{ and all } S \subseteq V \\ & u_{ij} \geq 0 && \text{for all } \{i, j\} \in E \end{aligned}$$

It is neither clear how to solve the separation, nor the pricing problem of this reformulation.

5.3 When the Scenario Set is Given As a Finite List

Another way to state the cut-set separation problem (5.1) is the following. A vector $u \in \mathbb{R}^E_{\geq 0}$ is feasible for an sRND instance if and only if u^* satisfies

$$\sum_{\{i,j\}\in E} u^*_{ij} \geq |\sum_{i\in S} b_i| \tag{5.8}$$

for all cut-sets $S \subseteq V$ *and* all vertices b of $\mathfrak{B}$. Therefore, in order to assert the feasibility of u^*, it suffices to enumerate all vertices b of $\mathfrak{B}$ and to then check the cut-condition (5.8) for the fixed b and all cut-sets S. We assume for this section that this enumeration is possible efficiently, i.e., that the vertices of $\mathfrak{B}$ are given in the input. The result will be a cut-set separation algorithm for the sRND-F problem.

5.3.1 Polynomial Time Separation of Cut-Set Inequalities

Let $G = (V, E)$ be an undirected graph and let $\mathfrak{B} = \{b^1, \ldots, b^K\}$ be a scenario set given as a finite list, i.e., in a vertex description. Let us also consider a fixed vertex $b^q \in \mathfrak{B}$, for some $q \in \{1, \ldots, K\}$ and a capacity vector $u^* \in \mathbb{R}^E_{\geq 0}$. We want to decide if there exists a cut-set $S \subseteq V$ that induces a violated cut-set inequality with respect to u^* and b^q. To this end, we define an auxiliary graph $\widehat{G} = (V \cup \{s\}, \widehat{E})$ with

$$\widehat{E} := E \cup \{\{s, i\} \mid \ i \in V\}.$$

We also extend the capacities given by u^* to our auxiliary graph $\widehat{G}$ by setting

$$\begin{aligned} \hat{u}^*_{si} &:= -b^q_i && \text{for all } i \in V \text{ and} \\ \hat{u}^*_{ij} &:= u^*_{ij} && \text{for all } \{i,j\} \in E. \end{aligned}$$

We can rewrite the weight $w(S \cup \{s\})$ of any minimum s-cut $S \cup \{s\}$ in $\widehat{G}$ as

$$w(S\cup\{s\}) = \sum_{\{i,j\}\in\delta_{\widehat{G}}(S\cup\{s\})} \hat{u}^*_{ij} = \sum_{\substack{\{i,j\}\in\delta_{\widehat{G}}(S\cup\{s\}) \\ s\notin\{i,j\}}} \hat{u}^*_{ij} + \sum_{\substack{\{i,j\}\in\delta_{\widehat{G}}(S\cup\{s\}) \\ s\in\{i,j\}}} \hat{u}^*_{ij} = \sum_{\{i,j\}\in\delta_G(S)} u^*_{ij} - \sum_{i\in V\setminus S} b^q_i.$$

As before (see Theorem 5.1), this implies that $S \cup \{s\}$ satisfies $\sum_{i\in V\setminus S} b^q_i \geq 0$; otherwise, its complement $(V \setminus S) \cup \{s\}$ has a better objective value. It follows that

$$w(X\cup\{s\}) = \sum_{\{i,j\}\in\delta_G(S)} u^*_{ij} - |\sum_{i\in V\setminus S} b^q_i| = \sum_{\{i,j\}\in\delta_G(S)} u^*_{ij} - |\sum_{i\in S} b^q_i|$$

because b^q is balanced. As a consequence, the value of $S \cup \{s\}$ is exactly the slack of the cut-set inequality that S would induce if b^q was the only scenario. The slack of the true cut-set inequality induced by S can only be smaller and therefore we know that if $w(S \cup \{s\}) < 0$, then also

$$0 > \sum_{\{i,j\}\in\delta_G(S)} u^*_{ij} - |\sum_{i\in S} b^q_i| \geq \sum_{\{i,j\}\in\delta_G(S)} u^*_{ij} - \max_{b\in\mathfrak{B}} |\sum_{i\in S} b_i|$$

and S defines a violated cut-set inequality in G. This proves that any minimum s-cut on $\widehat{G}$ with negative weight induces a violated cut-set inequality.

To show that any violated cut-set inequality implies a negative weight cut, assume that some $S \subseteq V$ induces a violated cut-set inequality whose right-hand side is determined by a vertex b^q of $\mathfrak{B}$. As we observed previously, we can assume $\sum_{i \in V \setminus S} b_i^q \geq 0$ without loss of generality; otherwise, we can just consider $V \setminus S$ without changing the inequality. It follows that the weight of the induced cut is negative:

$$\sum_{\{i,j\} \in \delta_G(S)} u_{ij}^* \; - \; \sum_{i \in V \setminus S} b_i^q \; = \; \sum_{\{i,j\} \in \delta_G(S)} u_{ij}^* \; - \; \left|\sum_{i \in S} b_i^q\right| \; = \; \sum_{\{i,j\} \in \delta_G(S)} u_{ij}^* \; - \; \max_{b \in \mathfrak{B}} \left|\sum_{i \in S} b_i\right| \; < 0$$

Thus, by computing a minimum cut on $\widehat{G}$ for each scenario, we can find up to $|\mathfrak{B}|$ violated cut-set inequalities or decide that none exist.

The construction of $\widehat{G}$ uses mixed (i.e., positive and negative) edge weights and in general, the problem of finding a minimum cut in an arbitrary graph with mixed weights is NP-hard [Kar72]. Here, however, all edges with negative weight are incident to a single node s. McCormick, Rao and Rinaldi show that by connecting all nodes to a new artificial node t, this *star negative* case can be reduced to a regular minimum s-t-cut with non-negative weights, see Theorem 1.16. This construction changes the size of G by a constant only and because the minimum s-t-cut problem is polynomial time solvable if the weights are non-negative, we obtain the main theorem of this section.

Theorem 5.3. *Let $G = (V, E)$ be an undirected graph and let $\mathfrak{B} = \{b^1, \ldots, b^K\}$ be a scenario set given in a vertex description. Let $u^* \in \mathbb{R}^E_{\geq 0}$. Then, we can find a cut-set inequality that is violated by u^* or decide that no such inequality exists in time $O(|\mathfrak{B}| \cdot T_{mincut})$, where T_{mincut} denotes the time needed to compute a minimum s-t-cut.* □

As discussed in detail in Chapter 1, any maximum flow algorithm can be used to compute a minimum s-t-cut. The Branch-and-Cut-Algorithm in Chapter 6 uses a custom implementation of preflow-push algorithm by Goldberg and Tarjan [GT88; CG95] with the highest label strategy and the gap heuristic. We can stop the algorithm when a maximum *pre*flow is found, thus omitting its second stage. This results in an overall running time of $\Theta(|\mathfrak{B}| \cdot |V|^2 \cdot \sqrt{|E|})$ for the separation procedure. Using the recent maximum s-t-flow algorithm by Orlin [Orl13] results in a worst-case running time of $\Theta(|\mathfrak{B}| \cdot |V| \cdot |E|)$, which is an improvement if $|E| \in o(|V|^2)$.

5.3.2 Separating 3-Partition Inequalities More Efficiently

The assumption that $\mathfrak{B}$ is finite does not only help us to find an efficient separation procedure for cut-set inequalities; it also enables us to find 3-partition inequalities more efficiently. In the general 3-partition separation algorithm from Section 4.4.2, we observed that we can obtain valid 3-partition inequalities by combining two cut-set inequalities with small slack. Instead of enumerating all pairs of binding cut-set inequalities as in Section 4.4.2, however, we can now develop an algorithm whose running time is linear in the number of binding cut-set inequalities.

The key observation for this more efficient algorithm is the following: Our cut-set separation algorithm yields an inequality with maximum violation. Thus, if we try to

separate a point u^* that already satisfies all cut-set inequalities, it returns an inequality with *minimum slack*. We use this fact to search for candidates for the zero-half cut generation in our algorithm `MinCutZH`: For each binding cut-set inequality (CS_S) in the current LP relaxation, we call the cut-set separation from the previous subsection on the subgraph $G[S]$ that is induced by S. This yields up to $|\mathfrak{B}|$ cut-sets $T_1 \ldots, T_k \subset S$. By adding up ($\mathrm{CS}_{T_i}$), ($\mathrm{CS}_{S \setminus T_i}$) and ($\mathrm{CS}_{T_i \cup S \setminus T_i}$) = ($\mathrm{CS}_S$) we thus obtain one 3-partition inequality for each $i = 1, \ldots, k$. This algorithm has a running time of $O(C \cdot |\mathfrak{B}| \cdot T_{mincut})$ where C is the number of binding cut-set inequalities in the current LP relaxation and T_{mincut} again denotes the time needed to compute a minimum s-t-cut in G. It thus depends linearly on the number of binding cut-set inequalities.

Apart from the running time, the algorithm has another advantage over `EnumZH`: There might be good candidate cut-set inequalities that are not part of the current LP solution – and these can only be found by `MinCutZH`. On the other hand, we cannot guarantee that the right hand side of $(\mathrm{CS}_S) + (\mathrm{CS}_T) + (\mathrm{CS}_S)$ is odd and therefore it can happen that `MinCutZH` does not find a violated 3-partition inequality even though one exists.

5.4 Separating over Scenarios in a Linear Description

If the scenario set is given in a linear description, we cannot necessarily iterate efficiently over all its vertices. Thus, we cannot simply separate cut-set inequalities with the procedure from the previous section. Instead, we have to solve the general separation problem (5.3) that we discussed at the beginning of this chapter. As described there, we cannot rely on general robustness approaches from the literature for solving this problem. However, Mattia [Mat13] shows how to separate metric inequalities for the `mRND` problem by applying the dualization technique by Bertsimas and Sim [BS03] to a formulation of the separation problem. The result is a non-convex quadratic optimization problem that Mattia linearizes with big-M constraints (her approach is discussed in more detail in Section 2.5.2). Metric inequalities are more general than our cut-set inequalities, and indeed we have found an easier way to write the separation problem as a quadratic problem (5.3). We will also find an easier way to solve (5.3) that does not require big-M constraints and yields a much smaller mixed integer linear program. This approach is, however, limited to a certain scenario set that is a single-commodity variant of the standard Hose uncertainty set by Duffield, Goyal, Greenberg *et al.* [DGG+99] and by Fingerhut, Suri and Turner [FST97].

5.4.1 The Hose Uncertainty Set for sRND-P

After the results in Chapter 4 that hold for any scenario set, we focus now on a special scenario polytope that is an adaptation of the Hose model [DGG+99; FST97] from Section 2.5.1 to single-commodity network design.

As in the original Hose model, we extend the problem input: For each node $i \in V$, we define an upper bound $b_i^{max} \in \mathbb{Z}$ and a lower bound $b_i^{min} \in \mathbb{Z}$. We then say that any vector $b \in \mathbb{R}^V$ whose components obey these bounds *while remaining balanced* is a possible scenario

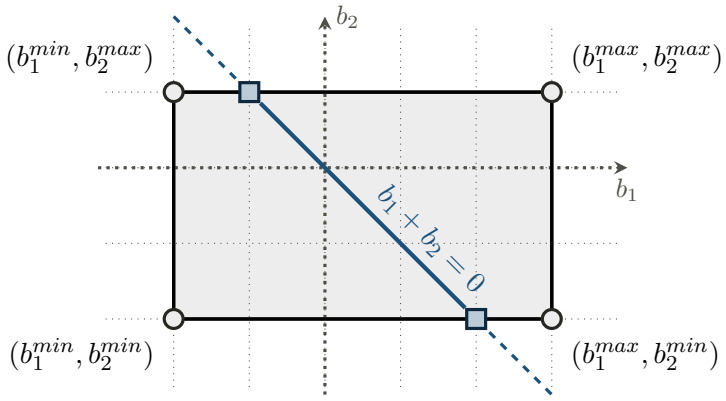


Figure 5.1: The round vertices span a relaxation of the 2-dimensional Hose Polytope (in gray) that does not have the balancing equality. The corresponding 2-dimensional Hose polytope (in blue print) is spanned by the square vertices: It arises from intersecting the relaxed polytope with the balancing equality.

for our optimization. The resulting uncertainty set is the polytope

$$\mathfrak{H}(V, b^{min}, b^{max}) := \left\{ b \in R^V \;\middle|\; b_i \in [b_i^{min}, b_i^{max}] \text{ for all } i \in V \text{ and } \sum_{i \in V} b_i = 0 \right\}.$$

Due to its similarity to the original Hose uncertainty set for `mRND`, we call it the *single commodity Hose polytope*. Figure 5.1 shows an example polytope in two dimensions. The condition that all scenarios should be balanced is necessary by Lemma 1.19. It is also very natural: Any supply or demand that cannot be balanced out will not be shipped through the network and thus does not require any capacity. We observe that if $b^{min} \in \mathbb{Z}^V$ and $b^{max} \in \mathbb{Z}^V$ are integral, then the vertices of $\mathfrak{H}(V, b^{min}, b^{max})$ are integral as well and therefore, the set is indeed a scenario set according to our Definition 2.3.

Lemma 5.4. *Let $b^{min} \in \mathbb{Z}^V$ and $b^{max} \in \mathbb{Z}^V$ be integral vectors. Then all vertices of $\mathfrak{H}(V, b^{min}, b^{max})$ are integral.*

Proof. In standard form, the polytope $\mathfrak{H}(V, b^{min}, b^{max})$ is the set of vectors $b \in \mathbb{R}^V$ that satisfy the following system of inequalities.

$$Ab := \begin{pmatrix} I \\ -I \\ e^{\mathrm{T}} \\ -e^{\mathrm{T}} \end{pmatrix} \cdot b \geq \begin{pmatrix} b^{min} \\ -b^{max} \\ 0 \\ 0 \end{pmatrix}$$

where $I \in \mathbb{R}^{V \times V}$ is the $|V| \times |V|$ identity matrix and $e = (1, \ldots, 1)^{\mathrm{T}} \in \mathbb{R}^V$ is the $|V|$-vector whose entries are all one. By Lemma 1.13, the blocks containing I and $-I$ are irrelevant for the total unimodularity of A and therefore, it suffices to show that

$$E := \begin{pmatrix} e^{\mathrm{T}} \\ -e^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} 1 & \cdots & 1 \\ -1 & \cdots & -1 \end{pmatrix}$$

is totally unimodular. As all entries of E are 1 or -1, any 1×1 square submatrix of E is totally unimodular. Any other square submatrix of E is of size 2×2 and has the form

$$\begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}$$

thus having a determinant of $1 \cdot (-1) - (-1) \cdot 1 = 0$. It follows that A is totally unimodular. □

In the following, we assume that our uncertainty set $\mathfrak{B}$ is the polytope $\mathfrak{H}(V, b^{min}, b^{max})$ and denote the corresponding `sRND` problem by `sRND-H`.

Pesenti, Rinaldi and Ukovich [PRU04] propose a similar uncertainty model for single-commodity flows. They start from the multi-commodity model and limit the traffic demand r_{ij} for each pair of nodes by an individual upper and lower bound, r_{ij}^{max} and r_{ij}^{min}. Given any such matrix $r = (r_{ij})_{i,j \in V}$ with $r_{ij}^{min} \leq r_{ij} \leq r_{ij}^{max}$, they aggregate the commodities to a demand vector $(b_i)_{i \in V} := (\sum_{j \in V} r_{ij} - r_{ji})_{i \in V}$. Any demand vector that can be obtained in this fashion is a scenario that needs to be considered in the optimization. This problem is called the *Network Containment Problem* in the literature. Pesenti, Rinaldi and Ukovich subsequently propose to solve the problem with a branch-and-cut algorithm based on a cut-set formulation and a separation MIP, which is exactly the approach that we will follow in this chapter.

The fundamental difference in the two models is the way that sources are assigned to sinks. In our model, the assignment can be chosen by the optimization. This means that the source-sink assignment is made such that the *least* possible amount of capacity is used. In that sense, our model optimizes for the best case. In the model by Pesenti, Rinaldi and Ukovich however, the capacities must be feasible for *all* source-sink assignments. This optimizes under the assumption that the sources and sinks are connected in a way that consumes the *greatest* possible amount of capacity.

5.4.2 Complexity of sRND with Hose Uncertainties

Finding an optimum integer solution for `sRND-H` is $\mathcal{NP}$-hard, as the problem still contains Steiner Tree as a special case. To show this, we can use a similar reduction as in Section 3.1.

Theorem 5.5. *The* `sRND-H` *problem is $\mathcal{NP}$-hard.*

Proof. Let $I = (V, E, c, T)$ be an input for the Steiner Tree problem, i.e., suppose that $G = (V, E)$ is an undirected graph with edge weights c and that $\emptyset \subsetneq T \subseteq V$ is a set of terminals that need to be connected with a subtree of minimum cost. Steiner Tree is $\mathcal{NP}$-hard [Kar72]. Then, finding an optimum solution for I is equivalent to finding an optimum solution for the following `sRND` instance J: Select some arbitrary node $s \in T$. We set $b_s^{min} = 0$ and $b_s^{max} = 1$. For all other nodes $i \in T \setminus \{s\}$, set $b_i^{min} = -1$ and $b_i^{max} = 0$. Now, the vertices of $\mathfrak{H}(V, b^{min}, b^{max})$ are exactly the scenarios b where $b_s = 1$ and $b_i = -1$ for some node $i \in T$. This means that in any feasible solution for J, there must be a path of capacity 1 from s to all terminals $i \in T \setminus \{s\}$. Also, if the support of any feasible integer solution for J contains a cycle, then one edge of the cycle can be deleted. Thus, any optimum solution for J induces a Steiner Tree and any Steiner Tree solution for I defines a

solution for J; moreover, the costs of the solutions are identical in both cases. Thus, when $\mathfrak{B} = \mathfrak{H}(V, b^{min}, b^{max})$, solving sRND is at least as hard as solving Steiner Tree. □

The sRND-H problem is a special case of the more general sRND-P problem.

Corollary 5.6. *The* sRND-P *problem is $\mathcal{NP}$-hard.* □

We shall see in the remainder of the section that the separation problem for cut-set inequalities is also $\mathcal{NP}$-hard for sRND-H. This proves that sRND-H remains hard even if we relax the integrality requirement.

5.4.3 Separating Cut-Set Inequalities over the Hose Polytope

In the beginning of this chapter we have seen that not all combinations of a cut-set $S \subseteq V$ and a scenario $b \in \mathfrak{B}$ are relevant for the cut-set separation problem: We only need to consider those combinations where the total balance of S with regard to b is non-negative. In the Hose case, the total balance of a cut-set is bounded by the upper and lower bounds on the balance of each node.

Definition 5.7. *For a connected, undirected graph $G = (V, E)$ and a Hose polytope $\mathfrak{H}(V, b^{min}, b^{max})$, let $S \subseteq V$ be a cut-set. If we have that*

$$\sum_{i \in S} b_i^{max} \geq 0 \quad \textit{and} \quad \sum_{i \in V \setminus S} b_i^{min} \leq 0,$$

then we say that S is a Hose source set.

If the sum of the upper node bounds b^{max} in S is negative, then S must have a negative total balance in each scenario. Likewise, if the sum of the lower node bounds b^{min} in $V \setminus S$ is positive, then $V \setminus S$ must have a positive total balance in all scenarios. However, because the scenarios are balanced, this implies a *negative* total balance of S in all scenarios as well. Thus, there can only be a scenario with a positive total balance in S if S is a Hose source set. By Definition 5.2, we call such a set *relevant*.

Lemma 5.8. *If $b \in \mathfrak{H}(V, b^{min}, b^{max})$ is relevant for S, then S is a Hose source set.*

Proof. Let $b \in \mathfrak{H}(V, b^{min}, b^{max})$ be relevant for S. It follows immediately that $\sum_{i \in S} b_i^{max} \geq \sum_{i \in S} b_i \geq 0$. On the other hand, we have $\sum_{i \in V \setminus S} b_i^{min} \leq \sum_{i \in V \setminus S} b_i = -\sum_{i \in S} b_i \leq 0$. □

Our next aim is to prove the existence of a relevant scenario in any Hose source set. The proof will be constructive, i.e., we propose an algorithm that is able to find a relevant worst case scenario given a Hose source-set S. To get a better intuition for the algorithm, suppose for the moment that $0 \in \mathfrak{H}(V, b^{min}, b^{max})$ and consider the following preliminary method to find an optimum solution for the optimization problem on the right-hand side of the cut-set

inequality induced by a fixed Hose source set $S \subseteq V$:

$$\begin{aligned} \max \quad & \sum_{i \in S} b_i && && (5.9) \\ \text{s.t.} \quad & b_i \leq b_i^{min} && \text{for all } i \in V \\ & b_i \geq b_i^{max} && \text{for all } i \in V \\ & \sum_{i \in V} b_i = 0. \end{aligned}$$

We start with the vector $b \equiv 0 \in \mathfrak{H}(V, b^{min}, b^{max})$ and our aim is to install as much supply as possible in S. Equivalently, we could try to install as much demand as possible in S, but since we assumed w.l.o.g. that the maximum total balance of S is non-negative, we rather stick to the maximum supply case. We now select an arbitrary node $i \in S$ with $b_i < b_i^{max}$ and another arbitrary node $j \in V \setminus S$ with $b_j > b_j^{min}$. If no such nodes can be found, the algorithm stops. Finally, we increase b_i by one unit and, at the same time, decrease b_j by one unit to maintain a balanced vector.

Let us analyze this algorithm. It maintains at all times that $\sum_{i \in S} b \leq \sum_{i \in S} b_i^{max}$ and that $\sum_{i \in S} b_i = -\sum_{i \in V \setminus S} b_i \leq -\sum_{i \in V \setminus S} b_i^{min}$. Also, it stops as soon as equality holds in one of the conditions. Thus, if b is the vector that we obtain once the algorithm stops, we have $\sum_{i \in S} b_i = \min\{\sum_{i \in S} b_i^{max}, -\sum_{i \in V \setminus S} b_i^{min}\}$. If equality holds in the first condition, then the upper bounds in S limit the maximum total supply of S. Otherwise, the maximum total supply in S is limited by the lower bounds in $V \setminus S$.

Definition 5.9. *Let $G = (V, E)$ be a connected, undirected graph and let $\mathfrak{H}(V, b^{min}, b^{max})$ be a Hose polytope. For any Hose source set $S \subseteq V$, we say that S is limiting if $\sum_{i \in S} b_i^{max} \leq -\sum_{i \in V \setminus S} b_i^{min}$. Otherwise, we say that $V \setminus S$ is limiting.*

The idea of this preliminary algorithm is to start from a feasible vector and to then increase its objective value. We follow the same idea in the case that $0 \notin [b_i^{min}, b_i^{max}]$ for some $i \in V$, however, we need a slightly more involved algorithm to do so. The problem is that the starting vector $b \equiv 0$ might be infeasible. More verbosely, the node bounds can force us to install supply on a node in $V \setminus S$ or to install demand on a node in S and thereby change the amount of imbalance that we have to distribute. This is why, in contrast to the preliminary algorithm, we start with a vector b that simply satisfies $b_i \in [b_i^{min}, b_i^{max}]$ for all $i \in V$ and then make sure that $\sum_{i \in V} b_i = 0$ in a second phase.

Additionally, the running time of the preliminary algorithm is only pseudopolynomial, as the algorithm needs $\min\{\sum_{i \in S} b_i^{max}, \sum_{i \in V \setminus S} b_i^{min}\}$ many iterations. We overcome this second problem by increasing the b values by as much possible in every iteration. To know this amount, it is necessary to precompute which of the two bounds is reached first, i.e., whether S or $V \setminus S$ is the limiting set. If S has more limiting bounds than $V \setminus S$, we set $b_i = b_i^{max}$ for all $i \in S$; otherwise, we set $b_i = b_i^{min}$ for all $i \in V \setminus S$. In both cases, it only remains to distribute the imbalance of b among the nodes in the non-limiting set. To do this, we iterate over all nodes i in the non-limiting set in arbitrary order and decrease or increase b_i as much as possible in the first and second case, respectively. See Algorithm 2 on page 128 for the pseudo-code of this procedure. When the algorithm stops with a balanced vector b, we obtain again a solution b of value $\min\{\sum_{i \in S} b_i^{max}, -\sum_{i \in V \setminus S} b_i^{min}\}$.

Algorithm 2: Computing a worst-case scenario for a fixed S.

input : Vectors b^{min}, b^{max}, a hose source set $S \subseteq V$
output : A worst-case scenario b for S.

1 let $F := \emptyset$
2 let $b \equiv 0$
Determine which of S and $V \setminus S$ is limiting and store the non-limiting set in F. If S is the limiting set, we choose the b_i value for $i \in S$ as large as possible. Otherwise, the set $V \setminus S$ is limiting and we choose the b_i value for $i \in V \setminus S$ as small as possible.
3 **if** $\sum_{i \in S} b^{max} \leq -\sum_{i \in V \setminus S} b^{min}$ **then**
4 set $F := V \setminus S$
5 **for** $i \in S$ **do** set $b_i := b_i^{max}$
6 **else**
7 set $F := S$
8 **for** $i \in V \setminus S$ **do** set $b_i := b_i^{min}$
9 **end**
Choose an initial value b_i for all nodes $i \in F$ in the non-limiting set. Choose the value from $[b_i^{min}, b_i^{max}]$ that is closest possible to zero.
10 **for** $i \in F$ **with** $b_i^{min} > 0$ **do** set $b_i := b_i^{min}$
11 **for** $i \in F$ **with** $b_i^{max} < 0$ **do** set $b_i := b_i^{max}$
Compute the current balance of b. If b is not balanced, distribute the imbalance in F.
12 let $r := \sum_{i \in V} b_i$
13 **if** $r == 0$ **then return** b
14 **else if** $r < 0$ **then**
If the imbalance is negative, we only consider nodes that can take positive b values. All other nodes cannot reduce the imbalance (due to our choice in lines 10/11).
15 **for** $i \in F$ **with** $b_i^{max} > 0$ **do**
16 let $m := \min\{b_i^{max} - b_i, -r\}$
17 set $b_i := b_i + m$ and set $r := r + m$
18 **if** $r == 0$ **then return** b
19 **end**
20 **else**
If the imbalance is positive, we only consider nodes that can take negative b values. All other nodes cannot reduce the imbalance (due to our choice in lines 10/11).
21 **for** $i \in F$ **with** $b_i^{min} < 0$ **do**
22 let $m := \max\{b_i^{min} - b_i, -r\}$
23 set $b_i := b_i + m$ and set $r := r + m$
24 **if** $r == 0$ **then return** b
25 **end**
26 **end**
27 **return** "$\mathfrak{H}(V, b^{min}, b^{max})$ is empty."

Lemma 5.10. *Given a Hose source set $\emptyset \subsetneq S \subsetneq V$, Algorithm 2 computes a relevant scenario $b \in \mathfrak{H}(V, b^{min}, b^{max})$ with*

$$\sum_{i \in S} b_i = \min\Big\{\sum_{i \in S} b_i^{max}, -\sum_{i \in V \setminus S} b_i^{min}\Big\}$$

or it correctly decides that $\mathfrak{H}(V, b^{min}, b^{max})$ is empty.

Proof. Let $L = S$ or $L = V \setminus S$ be the set that limits how much supply we can install in S. We prove the correctness of the algorithm by showing that Lines 13–27 maintain two invariants: (1) At all times, b respects all bounds, i.e., $b_i \in [b_i^{min}, b_i^{max}]$ for all $i \in V$. (2) At all times, r stores the balance of our current b vector, i.e. $r = \sum_{i \in V} b_i$.

We establish Invariant 1 in lines 3–9 and 10/11 for $i \in L$ and $i \in F$, respectively. Line 12 establishes Invariant 2. Suppose now that $r < 0$ in line 13 (the other case works analogously). We already know that both invariants hold before the first iteration of the loop in lines 14–19 and we assume by induction that the same is true before the j-th iteration, for some $j \geq 2$. Suppose that the j-th iteration considers $i \in F$. Then, b_i is at most increased to $b_i + b_i^{max} - b_i = b_i^{max}$, i.e. Invariant 1 is maintained. Also, r is changed by the same value as b_i and thus still stores the current balance of b. This means that Invariant 2 still holds. When the algorithm stops with $r = 0$, we have found a scenario $b \in \mathfrak{H}(V, b^{min}, b^{max})$. Also, by our choice in lines 3–9, we have $\sum_{i \in S} b_i = \sum_{i \in S} b_i^{max}$ if S is limiting and $\sum_{i \in S} b_i = -\sum_{i \in V \setminus S} = -\sum_{i \in V \setminus S} b_i^{min}$ otherwise. If the algorithm stops with $r < 0$, then $m = b_i^{max} - b_i$ in all iterations and thus, $b_i = b_i^{max}$ for all $i \in F$ where $b_i^{max} > 0$. From line 11, we know that $b_i = b_i^{max}$ for all $i \in F$ with $b_i^{max} < 0$ and our initialization guarantees $0 = b_i = b_i^{max}$ for all the $i \in F$ with $b_i^{max} = 0$. We conclude that $0 > r = \sum_{i \in F} b_i = \sum_{i \in F} b_i^{max}$. If $F = S$, we directly have a contradiction to S being a Hose source set. If $F = V \setminus S$ instead, we also have $b_i = b_i^{max}$ for all $i \in L$. It follows that $\sum_{i \in V} b_i = \sum_{i \in V} b_i^{max} < 0$. Now, let $b' \in \mathfrak{H}(V, b^{min}, b^{max})$. Then, $\sum_{i \in V} b'_i \leq \sum_{i \in V} b_i^{max} < 0$, which is a contradiction to $\sum_{i \in V} b'_i = 0$. Consequently, $\mathfrak{H}(V, b^{min}, b^{max}) = \emptyset$. □

Since Algorithm 2 can compute a relevant scenario for any Hose source set, the necessary condition in Lemma 5.8 is also sufficient.

Corollary 5.11. *There exists a relevant scenario $b \in \mathfrak{H}(V, b^{min}, b^{max})$ for S if and only if S is a Hose source set.* □

It remains to show that Algorithm 2 computes an optimum solution for the right-hand side optimization problem (5.9).

Theorem 5.12. *For a connected, undirected graph $G = (V, E)$ and a Hose polytope $\mathfrak{H}(V, b^{min}, b^{max})$, let $S \subseteq V$ be a Hose source set. Then,*

$$\max\Big\{\sum_{i \in S} b_i \;\Big|\; b \in \mathfrak{H}(V, b^{min}, b^{max})\Big\} = \min\Big\{\sum_{i \in S} b_i^{max},\ -\sum_{i \in V \setminus S} b_i^{min}\Big\}.$$

In particular, the scenario computed by Algorithm 2 is a worst-case scenario.

Proof. For $i = 1, \ldots, |V|$, introduce dual variables υ_i, λ_i for the upper/lower bound constraints of b_i, respectively, and define a dual variable β for the balance constraint. This gives us the dual (5.10) of (5.9)

$$\begin{aligned} \min \quad & \sum_{i \in V} b_i^{max} \upsilon_i - \sum_{i \in V} b_i^{min} \lambda_i && && (5.10) \\ \text{s.t.} \quad & \upsilon_i - \lambda_i + \beta \geq 1 && \text{for all } i \in S \\ & \upsilon_i - \lambda_i + \beta \geq 0 && \text{for all } i \in V \setminus S \\ & \upsilon_i, \lambda_i \geq 0 && \text{for all } i \in V. \end{aligned}$$

If $\sum_{i \in S} b_i^{max} \leq -\sum_{i \in V \setminus S} b_i^{min}$, running Algorithm 2 gives us a scenario b with $\sum_{i \in S} b_i = \sum_{i \in S} b_i^{max}$. We choose $\upsilon_i = 1$ for all $i \in S$, $\upsilon_i = 0$ for all $i \in V \setminus S$, $\lambda_i = 0$ for all $i \in V$ and finally $\beta = 0$. Our choice $(\upsilon, \lambda, \beta)$ is feasible for (H_S^*) and satisfies complementary slackness with b. If $\sum_{i \in S} b_i^{max} > -\sum_{i \in V \setminus S} b_i^{min}$, then Algorithm 2 yields a scenario b with $\sum_{i \in S} = -\sum_{i \in V \setminus S} b_i^{min}$. Choosing $\lambda_i = 1$ for all $i \in V \setminus S$, $\lambda_i = 0$ for all $i \in S$, $\upsilon_i = 0$ for all $i \in V$ and $\beta = 1$ is a feasible solution for (H_S^*) and b satisfies complementary slackness with $(\upsilon, \lambda, \beta)$. □

We can now write (5.2) as a MIP that is a maximum cut problem with additional constraints:

$$\begin{aligned} \min \quad & \sum_{\{i,j\} \in E} u_{ij}^* y_{ij} - B && && (5.11) \\ \text{s.t.} \quad & B \leq \sum_{i \in V} x_i b_i^{max} \\ & B \leq -\sum_{i \in V} (1 - x_i) b_i^{min} \\ \\ & x_i - x_j \leq y_{ij} && \text{for all } \{i,j\} \in E \\ & x_j - x_i \leq y_{ij} && \text{for all } \{i,j\} \in E \\ \\ & x_i \in \{0,1\} && \text{for all } i \in V \\ & y_{ij} \in \{0,1\} && \text{for all } \{i,j\} \in E \end{aligned}$$

The MIP will not give us an actual worst-case scenario; however, we can easily call Algorithm 2 on the set $S := \{i \in V \mid x_i = 1\}$ to obtain one. If we want more than one worst-case scenario, we can even call it several times while permuting the order in which it considers the nodes.

Unless $\mathcal{P} = \mathcal{NP}$, we cannot hope to find a polynomial time separation algorithm for the cut-set inequalities of the `sRND-H` problem because separating cut-set inequalities in the polyhedral case is $\mathcal{NP}$-hard. We demonstrate this by a reduction from *minimum expansion*. Chekuri, Oriolo, Scutellà and Shepherd [CSOS07] use a more complicated reduction from minimum expansion to show the same result for the multi-commodity case.

Theorem 5.13. *Given an instance* $(G, \mathfrak{H}(V, b^{min}, b^{max}))$ *of the* `sRND-H` *problem and a fractional capacity vector* $u \in \mathbb{R}^E$*, the cut-set separation problem "* Find a cut-set inequality that is violated by u or prove that none such inequality exists *" is* $\mathcal{NP}$*-complete. In particular, the feasibility test for* u *is co-*$\mathcal{NP}$*-complete.*

Proof. It can be decided in polynomial time if a given cut-set inequality is violated by u and therefore, the cut-set separation problem lies in $\mathcal{NP}$. We prove that it is $\mathcal{NP}$-hard as well by a reduction from the decision variant of minimum expansion. The variant is defined in the following way: Given an undirected graph $G = (V, E)$, edge capacities u_e for all edges $e \in E$ and a constant $r \in \mathbb{Z}_{>0}$, decide if there exists a set $\emptyset \subsetneq S \subsetneq V$ with $|S| \leq |V|/2$ that has an expansion $\sum_{e \in \delta(S)} u_e/|S|$ of strictly less than r. Minimum expansion is $\mathcal{NP}$-hard [LR99, Section 3.2]. We can check in polynomial time if a given set $S \subseteq V$ has an expansion of less than r and thus, the decision variant of minimum expansion is $\mathcal{NP}$-complete.

If we have an input (V, E, u, r) for minimum expansion, we can define an instance for the cut-set separation problem on the same graph $G = (V, E)$. Set $\hat{b}_i^{max} := r$ and $\hat{b}_i^{min} = -r$ for all $i \in V$. We claim that G has an expansion of strictly less than r if and only if there is a violated cut-set inequality with respect to u and $\mathfrak{H}(V, \hat{b}^{min}, \hat{b}^{max})$.

By our definition of $\hat{b}^{max}$ and $\hat{b}^{min}$, we obtain from Theorem 5.12 that the optimum right hand side B_S of the cut-set inequality induced by S is $r \cdot |S|$ for any $S \subseteq V$ with $|S| \leq |V|/2$.

Thus there exists a cut-set that is violated by u if and only if

$$\sum_{e \in \delta(S)} u_e < B_S = r \cdot |S| \iff \sum_{e \in \delta(S)} u_e/|S| < r$$

for some $S \subseteq V$. □

Corollary 5.14. *Given an instance* $(G, \mathfrak{B})$ *of the* `sRND-P` *problem and a fractional capacity vector* $u \in \mathbb{R}^E$*, the cut-set separation problem* u *is* $\mathcal{NP}$*-complete. In particular, the feasibility test for* u *is co-*$\mathcal{NP}$*-complete.* □

Chapter 6

A Branch-and-Cut Algorithm

In the last chapter of this thesis, we use all the theoretical results from the previous chapters to develop a Branch-and-Cut algorithm that can solve the `sRND-F` and the `sRND-H` problem exactly. For practical efficiency, we will need some additional algorithmic ideas: Several heuristics will allow us to compute better upper bounds for the `sRND` problem. Likewise, we propose modifications of the separation algorithms in order to separate several cut-set inequalities at once. A particularly important modification will be made in the `sRND-H` case. Finally, we analyze the practical performance of the algorithm with several computational experiments. The results in this Chapter were obtained in a collaboration with Eduardo Álvarez-Miranda, Valentina Cacchiani, Tim Dorneth, Michael Jünger, Frauke Liers, Andrea Lodi and Tiziano Parriani. They were published in [ACDJ+12] and [CJL+14].

6.1 The Algorithm

In this chapter, we propose a Branch-and-Cut algorithm to solve the capacity formulation (4.3) of the `sRND` problem. Using different separation procedures, the algorithm is capable of solving the `sRND-F` and the `sRND-H` problem to optimality.

6.1.1 Lower Bounds

We compute lower bounds by solving the linear programming relaxation of the capacity formulation (4.3) and strengthen the formulation with 3-partition inequalities and general $\{0, \frac{1}{2}\}$-cuts.

Cut-Set Separation for sRND-F

We use the cut-set separation algorithm from Section 5.3 for the `sRND-F` case. The algorithm builds on our own implementation of the Preflow-Push algorithm that is originally due to Goldberg and Tarjan [GT88]. Cherkassky and Goldberg [CG95] experimentally analyze the algorithm and find that the use of the *highest label* strategy and of the *gap heuristic* is crucial for the performance of the algorithm. We use both in our implementation. Moreover, it is well-known that in order to find a minimum cut, the Preflow-Push algorithm can be stopped once a maximum *pre*flow has been found. We therefore do not execute its second phase that only serves to convert the maximum preflow into a maximum flow. It can happen that the `sRND-F` separation returns the same cut-set for different scenarios; it is therefore important to check the separated inequalities for duplicates before adding them to the linear programming relaxation. We initialize the linear programming relaxation with all cut-set inequalities induced by the singletons $\{i\}$ for all $i \in V$.

Finding Several Cut-Set Inequalities per Iteration. We propose two additional heuristic separation algorithms to find more violated cut-set inequalities per iteration of the separation procedure. Experiments show by using these heuristics, we can reduce the number of iterations that the cut-set separation makes at the root node. However, with the additional inequalities, the size of the linear programming relaxation increases. This not only increases the time spent in the LP solver, it also increases the memory usage of the algorithm significantly. Additionally, it turns out that the time needed to solve the linear programming relaxation of the cut-set formulation is not a bottleneck of the algorithm – even if the additional heuristics are not used. Overall, it does not pay to use the addition separation heuristics; they do not improve the linear programming bounds, yet they decrease the number of Branch-and-Bound nodes that can be solved in a given amount of time. We thus describe the heuristics for documentation purposes only.

The first heuristic `InterCuts` only works for the polynomial cut-set separation from Section 5.3. We observe that the Preflow-Push algorithm maintains a (possibly suboptimal) s-t-cut at all times. This cut can change after each push operation. We modify our implementation of the algorithm such that it returns all these intermediate cuts and use them to generate potentially violated cut-set inequalities.

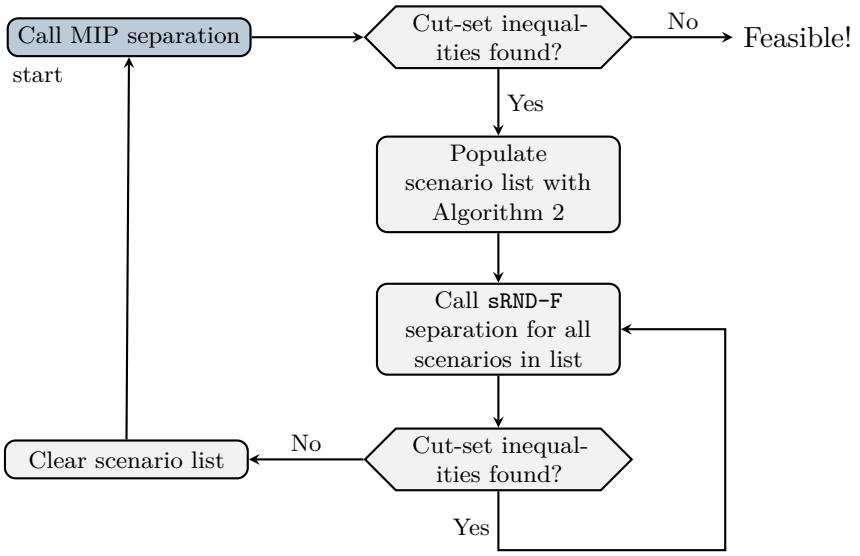


Figure 6.1: Flowchart of our separation algorithm for `sRND-H` cut-set inequalities.

The second heuristic `kOPT` is a general k-Opt neighborhood search for a fixed parameter $k \in \mathbb{Z}_{\geq 0}$. It takes the cut-set S of any violated cut-set inequality and tries to swap up to k nodes from S to $V \setminus S$ or vice-versa. If the resulting cut-set also defines a violated cut-set inequality, we add it to our linear programming relaxation.

Cut-Set-Separation for sRND-H

In principle, the MIP-based cut-set separation algorithm for the `sRND-H` problem would be sufficient to solve the linear programming relaxation of the `sRND-H` cut-set formulation. However, it requires solving a MIP for every separated inequality. To accelerate the separation procedure, we modify the algorithm in the following way: We first call the MIP separation algorithm. Once it returns a cut-set S, we make $k \in \mathbb{Z}_{\geq 0}$ calls[1] to Algorithm 2 to find up to k worst-case scenarios for S. We permute the order in which Algorithm 2 considers the nodes in each call; thus allowing the algorithm to return a different worst-case scenario in each run. The resulting scenarios are stored in a list L. Before calling the MIP-separation again, we first make sure that the capacities are sufficient for all scenarios in L. This can be done by calling the `sRND-F` cut-set separation from Section 5.3 for all $b \in L$. Once the capacities are sufficient for all scenarios in L, we call the MIP separation for `sRND-H` and iterate. The algorithm is depicted in Figure 6.1.

Separation of 3-Partition Inequalities

In the `sRND-F` case, we can separate 3-partition using the `EnumZH` algorithm from Section 4.4.2 and the `MinCutZH` algorithm from Section 5.3.2. However, our experiments show that `EnumZH` is only slightly slower than `MinCutZH` while providing better bounds; we therefore only use `EnumZH` in both the implementation for `sRND-F` and for `sRND-H`.

[1] In our implementation, we use $k = 10$.

We only separate 3-partition inequalities at the root node and only if no violated cut-set inequalities could be found.

Zero-Half-Cuts

If neither violated cut-set inequalities nor violated 3-partition inequalities can be found at the root node, we call a general $\{0, \frac{1}{2}\}$-cut separation algorithm by Andreello, Caprara and Fischetti [ACF07]. The algorithm relies on a tabu search and slightly improves the bounds.

6.1.2 Upper Bounds

We compute upper bounds with several `sRND` problem heuristics. In the description of these heuristics, let $G = (V, E)$ be a connected undirected graph with edge costs $c \in \mathbb{R}^E_{\geq 0}$ and let $\mathfrak{B}$ be a scenario set. With the exception of the `BLSRound` and the `LNSRound` heuristic, the heuristics do not make any asumption about $\mathfrak{B}$. All heuristics are based on rounding a fractional capacity vector $u^* \in \mathbb{R}_{\geq 0} \setminus \mathbb{Z}_{\geq 0}$ that must be feasible for the full linear programming relaxation of the cut-set formulation (4.3). We therefore only call the heuristics if no cut-set inequalities could be separated. In the description of the heuristics, we use

$$\Delta(x) := \min\{x - \lfloor x \rfloor, \lceil x \rceil - x\}$$

to denote the **fractionality** of a non-negative real number $x \in \mathbb{R}_{\geq 0}$. Also, let $G_\varepsilon(u)$ the subgraph of G that is induced by the edge set $E_\varepsilon(u) := \{e \in E \mid \Delta(u_e) \leq \varepsilon\}$ for some $\varepsilon > 0$.

Simple Rounding

Because u^* is fractionally feasible, the rounded vector defined by $u_{ij} = \lceil u^*_{ij} \rceil$ for all $\{i, j\} \in E$ is integer feasible (any flow that is feasible under u^* is feasible under u as well). Thus, a very simple rounding heuristic is to just round up the solution that is obtained from the linear programming relaxation. We denote this heuristic as `SRound`.

Cycle Rounding

Let $C = (e_1, \ldots, e_l)$ be a cycle in $G_\varepsilon(u)$ and assume without loss of generality that e_1 is an edge that minimizes $\min_{e \in C} u_e - \lfloor u_e \rfloor$. Then, set $u'_{e_1} := \lfloor u_{e_1} \rfloor$ and $u'_{e_i} = u_{e_i} + u_{e_1} - \lfloor u_{e_1} \rfloor$ for all $i = 2, \ldots, \ell$. Even though we reduced the capacity of e_1, the new solution u' remains fractionally feasible: Any flow that used the missing capacity on $e_1 = \{i, j\}$ can now send the same flow units on the i-j-path $(e_2, \ldots, e_l)$ whose capacity we increased. We can thus iterate the algorithm on $G_\varepsilon(u')$ until no more cycles can be found. We then apply the simple rounding heuristic to make the last solution integer feasible. In the worst case, the heuristic iterates $|E|$ times because in each iteration, one fractional edge is eliminated. The actual worst-case running time of the heuristic depends on how it looks for the cycles.

DFS and Shortest Path Rounding

There are different ways to find cycles in $G_\varepsilon(u)$, each yielding an instantiation of the Cycle Rounding heuristic from the previous paragraph. In the `DFSRound` heuristic, we use a depth

first search for this purpose. Any time the search finds a *back edge*, we have found a cycle.

Another possibility is to select an arbitrary edge $\{i,j\} \in E_\varepsilon(u)$ and to look for a shortest i-j-path p with respect to the objective function c in $(V, E_\varepsilon(u) \setminus \{i,j\})$. The result is a cycle $p \cup \{i,j\}$ in $G_\varepsilon(u)$. The result of the heuristic depends on how we choose the initial edge $\{i,j\}$; in our implementation, we consider the edges by decreasing order of their fractional part $u_e - \lfloor u_e \rfloor$. Since the fractional parts change after each rounding operation, we organize the edges in a binary heap. The resulting heuristic is the `SPRound` heuristic. We use our own implementation of Dijkstra's algorithm [Dij59] for the shortest path search.

Multi-Commodity Flow Rounding

In the cycle rounding heuristics, we have rerouted the fractional parts of the capacities along a single path. This is an arbitrary restriction and we now propose a heuristic that can use any number of paths. The multi-path splitting can be realized with a multi-commodity flow. To this aim, we maintain a list of commodities C that is initially empty and a capacity vector $\bar{u}$ which we initially set to $\bar{u}_{ij} = 1 - (u_{ij} - \lfloor u_{ij} \rfloor)$ for all $\{i,j\} \in E_\varepsilon(u)$. Thus, the entry $\bar{u}_{ij}$ is the "slack" capacity that would be additionally installed if we would round up u_{ij}. The heuristic now tries to reroute the flows to use these slack capacities in order to be able to round down the capacity of some edges.

The heuristic starts by selecting an arbitrary edge $\{i,j\} \in E_\varepsilon(u)$ and inserting a commodity that has the node i as its source, the node j as its sink and a demand of $u_{ij} - \lfloor u_{ij} \rfloor$. It also sets $\bar{u}_{ij} = 0$. We now check if all commodities in C can be sent in $(V, E_\varepsilon(u) \setminus \{i,j\}, \bar{u})$ and if not, we remove the commodity (i,j) and revert $\bar{u}_{ij}$ to $1 - (u_{ij} - \lfloor u_{ij} \rfloor)$. The check is performed by solving the multi-commodity flow arc formulation (1.16) as a linear program. Regardless of the outcome of the check, we iterate the algorithm until all edges in $G_\varepsilon(u)$ have been considered. In the end, we obtain a vector $u^*_{ij} := \lfloor u_{ij} \rfloor + \bar{u}_{ij}$ that is fractionally feasible. We make it integer feasible with the `sRound` heuristic.

The result of the heuristic again depends on the order in which the edges are considered. We observe that rounding down the capacity of an edge $\{i,j\}$ reduces the costs for that particular edge by $c_{ij} \cdot (u_{ij} - \lfloor u_{ij} \rfloor)$ and we consider the edges in the decreasing order of that amount. The result is the `MCFRound` heuristic.

In the final algorithm, the `MCFRound` heuristic is disabled: While it occasionally produces very good solutions, its running time is too high to make an improvement to the overall algorithm.

A Rounding heuristic by Buchheim, Liers and Sanità [BLS11]

Buchheim, Liers and Sanità propose a heuristic for the `sRND-F` case that only rounds indirectly. In a first step, the heuristic rounds the capacity of each edge $e \in E$ to $\overline{u}_e = \lceil u_e \rceil$, obtaining a vector $\overline{u} \in \mathbb{Z}^E_{\geq 0}$. It then chooses a random cost function $\bar{c} \in \mathbb{R}^E_{\geq 0}$. Given these two vectors, the heuristic computes a minimum cost b-flow f^b in $(G, \overline{u}, \bar{c})$ for each scenario b from the scenario set $\mathfrak{B} := \{b^1, \ldots, b^K\}$. It then returns the integer feasible vector u' with $u'_{ij} := \max_{b \in \mathfrak{B}}(f^b_{ij} + f^b_{ji})$ for all $\{i,j\} \in E$. The process is repeated $k \in \mathbb{Z}_{\geq 0}$ times. We call the result the `BLSRound` heuristic.

Our experiments have shown that the `BLSRound` heuristic outperforms all other rounding heuristics (with the exception of the following `LNSRound` heuristic) in terms of solution quality. We make one iteration of the heuristic at the end of each Branch-and-Bound node.

Large Neighborhood Search Heuristic

The large neighborhood search heuristic only works in the `sRND-F` case. It has two phases: In the constructive phase, it builds a feasible solution by computing a minimum cost b-flow f^b in (G, ∞, c) for each $b \in \mathfrak{B} = \{b^1, \dots, b^K\}$. This results in an integer feasible capacity vector $\bar{u}$ where $\bar{u}_{ij} = \max_{b \in \mathfrak{B}}(f^b_{ij} + f^b_{ji})$. In the following improvement phase, the heuristic tries to reduce the capacities in $\bar{u}$ by solving a MIP.

In order to keep the MIP small, we remove any edges that are not used in the constructive phase. Let thus $\overline{E} := \{e \in E \mid \bar{u}_e > 0\}$. For $\Theta \in \mathbb{Z}_{\geq 0}$, we say that $u \in \mathbb{R}^{\overline{E}}_{\geq 0}$ is in the Θ-neighborhood of $\bar{u}$ if u uses at most Θ additional units of capacity, i.e., if $\sum_{e \in \overline{E}} \bar{u}_e \leq \sum_{e \in \overline{E}} u_e + \Theta$. The improvement phase now looks for the cheapest feasible solution in the Θ-neighborhood of u. To this aim, it solves a variant of the arc-flow formulation (2.17) in which the x variables model the additional capacities.

$$
\begin{aligned}
\min \quad & \sum_{\{i,j\} \in \overline{E}} c_{ij} u_{ij} && && (6.1)\\
\text{s.t.} \quad & \sum_{\{i,j\} \in \overline{E}} (f^q_{ij} - f^q_{ji}) = b^q_i && \text{for all } i \in V \\
& && \text{and all } q = 1, \dots, K \\
& f^q_{ij} + f^q_{ji} \leq u_{ij} && \text{for all } i \in V \\
& && \text{and all } q = 1, \dots, K \\
& u_{ij} \leq \bar{u}_{ij} + x_{ij} && \text{for all } \{i,j\} \in \overline{E} \\
& \sum_{\{i,j\} \in \overline{E}} x_{ij} \leq \Theta && \\
& x_{ij}, u_{ij} \in \mathbb{Z}_{\geq 0} && \text{for all } \{i,j\} \in \overline{E} \\
& f^q_{ij}, f^q_{ji} \in \mathbb{R}_{\geq 0} && \text{for all } i \in V \\
& && \text{and all } q = 1, \dots, K
\end{aligned}
$$

This MIP is then solved with a general MIP solver. This `LNSRound` heuristic is only called in the `sRND-F` case and only at the beginning and at the end of the root node. For the second call, we initialize the improvement phase with the output of the `SRound` heuristic. Both calls have a time limit of 120 seconds.

6.1.3 Preprocessing

Instead of solving an `sRND` instance $(G = (V, E), \mathfrak{B})$ as a whole, we can decompose it into its biconnected components, solve the `sRND` problem independently on all components and then combine the partial solutions to obtain a solution for $(V, E, \mathfrak{B})$. This was shown by Buchheim, Liers and Sanità [BLS11] for the `sRND-F` case.

The decomposition works in the following way: We assume that G is connected. If G is also biconnected, then there is nothing to do. Otherwise, there is at least one biconnected

component $C_1 \subseteq V$ that contains a single cut-vertex i^*; if not, then the biconnected components form a cycle which is a contradiction to G not being biconnected. The cut vertex i^* decomposes G into a subgraph $G[C_1]$ and a remainder subgraph $G[C_2]$, where $C_1, C_2 \subseteq V$, that both contain i^*. Since i^* is the only connection of C to the rest of G, all flow entering or leaving C must go through i^*. Observe, however, that i^* can be contained in several biconnected components in general. We decompose $(V, E, \mathfrak{B})$ into two instances $(C_1, E[C_1], \mathfrak{B}_1)$ and $(C_2, E[C_2], \mathfrak{B}_2)$ in the following way.

Vertex Description of the Scenario Set. In the `sRND-F` case, suppose that the scenario set is given as $\mathfrak{B} = \{b^1, \dots, b^K\}$. For all $q = 1, \dots, K$, we define a new scenario $b^{1,q} \in \mathbb{R}_{\geq 0}^{C_1}$ on C_1 by setting

$$b_i^{1,q} = \begin{cases} b_i^q, & \text{if } i \neq i^* \\ \sum_{i \in C_2} b_i^q, & \text{otherwise} \end{cases} \qquad \text{for all } i \in C_1$$

and a new scenario $b^{2,q} \in \mathbb{R}_{\geq 0}^{C_2}$ on C_2 as

$$b_i^{2,q} = \begin{cases} b_i^q, & \text{if } i \neq i^* \\ \sum_{i \in C_1} b_i^q, & \text{otherwise} \end{cases} \qquad \text{for all } i \in C_2.$$

The new scenarios are balanced by definition and thus, the two new scenario sets $\mathfrak{B}_1 = \{b^{1,1}, \dots, b^{1,K}\}$ and $\mathfrak{B}_2 = \{b^{2,1}, \dots, b^{2,K}\}$ are valid.

Hose Scenario Set. We extend the definition from [BLS11] to the `sRND-H` case by defining two new hose polytopes $\mathfrak{B}_1 := \mathfrak{H}(C_1, b^{1,\min}, b^{1,\max})$ and $\mathfrak{B}_2 := \mathfrak{H}(C_2, b^{2,\min}, b^{2,\max})$ in the following way.

$$b_i^{1,\min} = \begin{cases} b_i^{\min}, & \text{if } i \neq i^* \\ \sum_{i \in C_2} b_i^{\min}, & \text{otherwise} \end{cases} \qquad \text{for all } i \in C_1$$

$$b^{1,\max} = \begin{cases} b_i^{\max}, & \text{if } i \neq i^* \\ \sum_{i \in C_2} b_i^{\max}, & \text{otherwise} \end{cases} \qquad \text{for all } i \in C_1$$

$$b_i^{2,\min} = \begin{cases} b_i^{\min}, & \text{if } i \neq i^* \\ \sum_{i \in C_1} b_i^{\min}, & \text{otherwise} \end{cases} \qquad \text{for all } i \in C_2$$

$$b^{2,\max} = \begin{cases} b_i^{\max}, & \text{if } i \neq i^* \\ \sum_{i \in C_1} b_i^{\max}, & \text{otherwise} \end{cases} \qquad \text{for all } i \in C_2$$

The idea of both definitions is that C_1 and C_2 define two `sRND` instances: In the first one, the node set C_2 is shrunken into a single node (similarly to our construction for 3-partition inequalities in Section 4.4.3) and only the graph $G[C_1]$ remains. In the second one, the

node set C_1 is shrunken into a single node, leaving only the graph $G[C_2]$. In both cases, the balance of the shrunken node is the total balance of the node set that it represents.

Suppose now that we have the instances $(G[C_1], \mathfrak{B}_1)$ and $(G[C_2], \mathfrak{B}_2)$ as defined above. Since C_1 does not contain a cut-vertex, we know that $G[C_1]$ is biconnected. It thus remains to solve $(G[C_1], \mathfrak{B}_1)$ with our Branch-and-Cut algorithm and to further decompose $G[C_2]$. For the latter, we now call the decomposition algorithm recursively on $(G[C_2], \mathfrak{B}_2)$. We continue recursively until G has been completely decomposed into biconnected components. When a recursive step returns, we obtain two solutions u_1 for $G[C_1]$ and u_2 for $G[C_2]$ that can be concatenated to obtain a solution for G.

6.1.4 Additional Settings

Some parameter choices are important for the performance of our Branch-and-Cut algorithm. We refer to Section 1.3.4 for more details on the meaning of these parameters.

First of all, we obtained the best results when using *Strong Branching*. We use an aggressive variant in which we solve the linear programming relaxation of all candidate subproblems to optimality, i.e., we do not bound the number of iterations that the simplex algorithm performs. However, we do not use the cut-set separation in the candidate subproblems and this is why the bounds computed in the candidate subproblems are still estimates only.

Second of all, the linear programming relaxation of the cut-set formulation can grow large even though a separation algorithm is used. We counter this behavior by using *aging* and remove constraints from the relaxation that have been non-binding for more than 10 iterations of the separation procedure. In choosing this parameter, we have to make a threefold trade-off: Keeping the constraints for too long results in a large linear programming relaxation that uses up valuable memory and solution time. On the other hand, discarding constraints too early not only implies otherwise unnecessary calls to the separation algorithm: Since our 3-partition separation algorithms rely on the cut-set inequalities from the linear programming relaxation, we get less 3-partition inequalities if we remove cut-set inequalities prematurely.

Third of all, we found that traversing the Branch-and-Cut tree in *Best-First* order works better than using *Depth-First* order or a *Dive-and-Best* strategy.

6.2 Testbed

Any benchmark instance for the `sRND` problem consists of a network topology, edge costs and a scenario set. This section starts with a description of suitable network topologies and then describes how we generate scenario sets for the `sRND-F` and the `sRND-H` case.

6.2.1 Network Topologies

The network topologies in the benchmark set should be both realistic and computationally challenging. At the same time, we try to use existing instances from the literature in order to compare with previous works.

BLS instances. The first class of instances has been derived from real world network topologies and was compiled by Altın, Amaldi, Belotti and Pınar [AABP07]. The instances were built out of standard backbone networks from the literature, Steiner tree instances and general multi-commodity flow instances. Buchheim, Liers and Sanità [BLS11] use the instances to benchmark their `sRND-F` Branch-and-Cut approach. The instances include edge costs that, depending on the instance, vary from different ranges.

JMP instances. Johnson, Minkoff and Phillips [JMP00] propose the following method to generate realistic Steiner Tree instances: Distribute n nodes uniformly at random in the unit square. Then, connect any two nodes i and j by an edge if and only if the Euclidean distance between i and j is less than $\alpha/\sqrt{n}$, where $\alpha = 2$ is a parameter for the generator. Edge costs are proportional to the Euclidean distance of the incident nodes. The instances of this class originate from [MCL+14].

PA instances. The preferential attachment model by Barabási and Albert [BA99] depends on a parameter $\beta \in \mathbb{Z}_{\geq 0}$. It starts the graph generation with a complete graph on β nodes. Then, the generator iteratively inserts additional nodes. When the generator inserts node i, it also inserts exactly β edges that connect i with β of the previously inserted nodes. The probability that i is connected to a node j is inversely proportional to the degree of j, i.e., new nodes prefer to be connected to existing nodes with high degree. Consequently, a `PA` instance with n nodes has exactly $(n-\beta)\cdot\beta + \beta\cdot(\beta-1)/2 = \beta n - \beta\cdot(\beta-1)/2$ many edges. We choose the edge costs uniformly at random from $\{0,\dots,100\}$. The class contains one network topology of size $|V| = 20, 25, \dots, 50, 60, \dots, 100$ for each choice of $\beta = 2, 3, \dots, 7$. Two example instances are depicted in Figure 6.3.

SND instances. The SNDLib [OPTW07] is an established standard benchmark set consisting of real world network topologies. The instances include uniform edge costs.

6.2.2 Generating Vertex Descriptions of Scenario Sets

BLS instances. Buchheim, Liers and Sanità [BLS11] generated vertex-description based scenario sets for the instances of the `BLS` class. We use the same set. For each of the 76 base network topologies, the class contains an instance with $k = 2, 3, 4, 5$ scenarios and a percentage of $p = 0.25, 0.5, 0.75, 1.0$ of terminals. In theory, this results in a total of 1216 instances, however, some trivial instances have been removed in [BLS11] in order to obtain a fair average. We group the remaining 1156 instances in three classes according to their size: Small instances are those that have up to 149 nodes; instances with 150–299 nodes are medium sized and those instances with more than 300 nodes are large. In order to generate node balances, Buchheim, Liers and Sanità make use of a demand value d_i for each node i that was provided by the underlying network topologies from [AABP07]. In a first phase, the balance of terminal i is randomly set to $+d_i$ or to $-d_i$. In the second phase, a terminal is chosen at random and its demand is decreased by one. The second phase iterates until the scenario is

balanced.[2] The absolute node balances in the BLS class tend to be larger than in the other instance classes that we consider.

Other Instances. Our theoretical results in Section 3.3.1 suggest that instances with binary supplies and demands are hard for Branch-and-Bound based algorithms. This was confirmed by our experimental results in [ACDJ+12]. This is why we generate binary demands on the JMP, PA and SNDLib instance classes. For each scenario, we select a percentage of $p = 0.25, 0.5, 1.0$ terminals and generate distinct terminal pairs s, t accordingly. We set the balance of each pair s, t to 1 and -1, respectively. On the JMP instances we generate one instance for each network topology, each choice of p and each choice of $k = 5, 10$. For the PA instances, we generate 10 instances for each network topology, each choice of p and each number $k = 5, 10, 20, 30, 50, 75, 100$ of scenarios.

6.2.3 Generating Linear Descriptions of Scenario Sets

In the polyhedral case, we consider the SNDLib and the PA instances. To define a Hose polytope on these network topologies, we need a value b_i^{min} and a value b_i^{max} for each terminal i (for each non-terminal i, we set $b_i^{min} = b_i^{max} = 0$). These values are chosen by selecting a center $\bar{b}_i$ and a radius $\hat{b}_i$ for each terminal i. Then, we set $b_i^{min} = \bar{b}_i - \hat{b}_i$ and $b_i^{max} = \bar{b}_i + \hat{b}_i$. The choice of $\hat{b}$ and $\bar{b}$ follows three different distributions.

uniform The centers and the radii are chosen uniformly at random from $\{-5, \ldots, 5\}$ and $\{0, \ldots, 5\}$, respectively.

geometric The radii of the demand intervals are chosen randomly according to a geometric distribution, i.e., there are many nodes with narrow demand intervals and few nodes with broad intervals. The expected radius is 2. The center of the intervals is chosen uniformly at random from $\{-5, \ldots, 5\}$.

zero-one We set $\hat{b}_i = 0$ and $\bar{b}_i = 1$ for all terminals.

It can happen that a randomly generated instance is infeasible; in that case, we generate another one. The class contains ten instances for each network topology and each percentage $p = 0.25, 0.5, 0.75, 1.0$ of terminals.

6.2.4 Software and Hardware for the Experiments

Our Branch-and-Cut algorithm is implemented in C++ using several libraries.

ABACUS Writing a Branch-and-Cut algorithm from scratch is tedious: The algorithm must maintain a Branch-and-Bound tree, store LP bases and cutting planes, make calls to a linear programming solver etc. All these things require large implementation and debugging efforts. For these tasks, we rely on "A-Branch-And-CUt-System (ABACUS)" that is based at the University of Cologne. The C++ library is due to Jünger and

[2] The scenario generation was detailed to the author in a private communication with the authors of [BLS11] in October 2011.

Thienel [JT00]. Many state-of-the-art techniques for Branch-and-Cut algorithms are included in this package and are ready to use. We use `ABACUS 3.2beta U2` for our code. The library depends on the `Open Solver Interface` (`COIN-OSI`) cite that we use in the version provided with `Clp 1.14.8`.

OGDF The `Open Graph Drawing Framework` [OGDF] is developed in a cooperation of the TU Dortmund, the Osnabrück University, the University of Cologne, the University of Sydney and the oreas GmbH. We use it for its extensive graph data structures, although its intention is to provide algorithms for the automated layouting and drawing of graphs. The framework also provides some basic graph algorithms.

CPLEX We use `ILOG Cplex 12.1` with `ILOG Concert 2.9` as the underlying linear programming solver. In our implementation, the MIP functionality is only used for the improvement phase of the `LNS` heuristic and for the Hose separation of cut-set inequalities. We use `CPLEX` in sequential (i.e., non-parallel) mode in all experiments.

All code has been compiled with the `GNU Compiler Collection` (`GCC`) in version 4.7 on `Debian 7`. The experiments were conducted on `Debian 7` as well and ran on a single core of a `Intel(R) Xeon(R) E5410` cpu running at `2.33GHz`. We use a time limit of four hours and limit the algorithms to 3 GB of memory in all experiments.

6.3 Computational Results

6.3.1 Collected Data

We use the following data in our comparison. All numbers are averages over those instances that were solved to optimality within the time and memory constraints. Wherever two algorithms are compared, the average is over those instances only that were solved to optimality by *both* algorithms.

`lp-gap` Denote by I the value of an optimum solution of the cut-set IP formulation (4.3) and by L the optimum value of the corresponding LP relaxation (4.2). Then the `lp-gap` is the ratio $(I - L)/(I \cdot 100)$. This is the gap that the algorithm needs to close to prove optimality.

`#solved` The number of instances that were solved to optimality within the time limit. The number in brackets denotes the number of instances that could not be solved to optimality due to the memory limit.

`cputime` This column depicts the average cpu time in seconds used on those instances that were solved to optimality.

`#nodes` The average number of branch-and-cut nodes that were used on those instances that were solved to optimality.

`root gap` The root gap is the quotient $(I - R)/(I \cdot 100)$ where I is again the value of an optimum solution of the cut-set IP formulation (4.3) and R is the value of an optimum solution of the linear programming relaxation at the end of the root node,

				Cut-Set formulation (CS)							[BLS11]	
$\|V\|$	$\|\mathfrak{D}\|$	#inst	lp-gap	#solved (m)	cputime	#nodes	root-gap	relax-time (m)	sep-time	heur-time	#solved	cputime
$0 \leq \|V\| \leq 149$	2	153	0.02%	153 (0)	2	111	0.00%	0 (0)	0	0	153	0.7
$0 \leq \|V\| \leq 149$	3	153	0.03%	152 (1)	7	265	0.00%	0 (0)	0	0	152	1.1
$0 \leq \|V\| \leq 149$	4	153	0.03%	151 (2)	2	105	0.00%	0 (0)	0	0	150	4.8
$0 \leq \|V\| \leq 149$	5	185	0.02%	182 (3)	0	127	0.00%	0 (0)	0	0	183	5.9
$150 \leq \|V\| \leq 299$	2	68	0.00%	67 (1)	2	78	0.00%	0 (0)	1	0	66	85.6
$150 \leq \|V\| \leq 299$	3	68	0.01%	68 (0)	45	205	0.00%	0 (0)	8	0	61	4.9
$150 \leq \|V\| \leq 299$	4	68	0.00%	66 (2)	2	95	0.00%	0 (0)	1	0	63	27.3
$150 \leq \|V\| \leq 299$	5	68	0.00%	67 (1)	82	186	0.00%	0 (0)	11	0	62	141.2
$300 \leq \|V\| \leq 499$	2	60	0.00%	60 (0)	0	197	0.00%	0 (0)	0	0	60	81.3
$300 \leq \|V\| \leq 499$	3	60	0.00%	60 (0)	0	169	0.00%	0 (0)	0	0	60	103.4
$300 \leq \|V\| \leq 499$	4	60	0.00%	60 (0)	0	221	0.00%	0 (0)	0	0	59	129.8
$300 \leq \|V\| \leq 499$	5	60	0.00%	60 (0)	0	547	0.00%	0 (0)	0	0	55	166.8

Table 6.1: Comparison with the previous approach by Buchheim, Liers and Sanità [BLS11]. The experiments were conducted on the same machine. The experiment shows that our approach is faster in terms of CPU time and solves slightly more instances (except for one case). The instances are not solved at the root node even though the relative root gap is 0.00%, as the absolute root gap is not 0. This is because the solution values are large.

i.e., including separated 3-partition inequalities and general $\{0, \frac{1}{2}\}$-cuts. The number in the table is the average over those instances that were solved to optimality.

`relax time` The time (in seconds) needed to solve the linear programming relaxation of the cut-set formulation (4.3) at the root node, averaged over the instances that were solved to optimality. The number in brackets gives the number of instances where the linear programming relaxation could not be solved due to the time limit.

`sep-time` The time (in seconds) spent in separation routines during the run of the algorithm, averaged over all instances that were solved to optimality.

`heur-time` The time (in seconds) spent in heuristics during the run of the algorithm, averaged over all instances that were solved to optimality.

6.3.2 The Previous Approach by Buchheim, Liers and Sanità

The aim of our first experiment is to show that our new algorithm is competitive. As a reference point, we use the previous Branch-and-Cut algorithm by Buchheim, Liers and Sanità [BLS11]. This algorithm uses an arc-flow formulation with additional target-cuts and is described in Section 2.5.2 of this thesis. It also builds on the `ABACUS` framework.

We run our algorithm on the same instances and the same machines that were used by [BLS11]. We then report on the number of instances that could be solved within four hours of time and 3GB of memory. These are the same settings as used in [BLS11]. Since we have access to the raw computational results of [BLS11], this allows for a direct comparison. Buchheim, Liers and Sanità analyze the performance of their algorithm depending on a

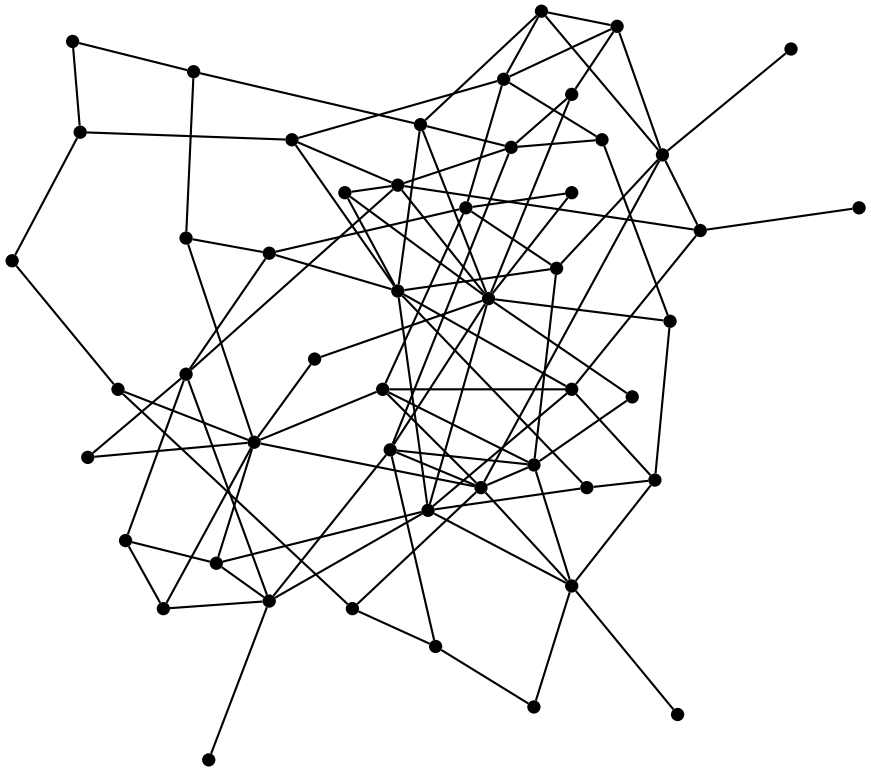


Figure 6.2: The instance `stein_50_9_100_K04_t075` is hard for our algorithm.

parameter choice for the target cut algorithm. In each instance group, we compare with the best parameter choice from [BLS11] for that group.

The computational results of this experiment are depicted in Table 6.1 using the same grouping as in [BLS11]. The results from [BLS11] are depicted on the right, the ones of our algorithm are on the left.

The table shows that our algorithm is competitive: It is faster in most cases and solves slightly more instances than the previous algorithm from [BLS11]. Our algorithm is significantly faster on the largest instances and in contrast to the previous algorithm, it can solve all instances from this class. In total, only 10 instances could not be solved; in all cases, the algorithm aborted due to the memory limit of 3GB. The unsolved instances stem from different subsets of the `BLS` class as defined in [AABP07]. One instance (see Figure 6.2) is part of the `steiner50` subset (46 nodes and 96 edges), five instances stem from the `steiner75` subset (69 nodes and 144 edges) and the remaining four instances are part of the `g200a` subset (199 nodes and 913 edges).

Buchheim, Liers and Sanità also report memory issues due to the large Branch-and-Cut tree, although some instances could not be solved by their algorithm due to the time limit.

The small root gap of the instances is misleading: The actual solution values are in the order of 10^5 to 10^7 and the actual relative root gap therefore has more than 2 significant decimal places. The absolute root gaps are large enough to pose problems to Branch-and-Cut algorithms. We conclude that our algorithm improves on the previous one.

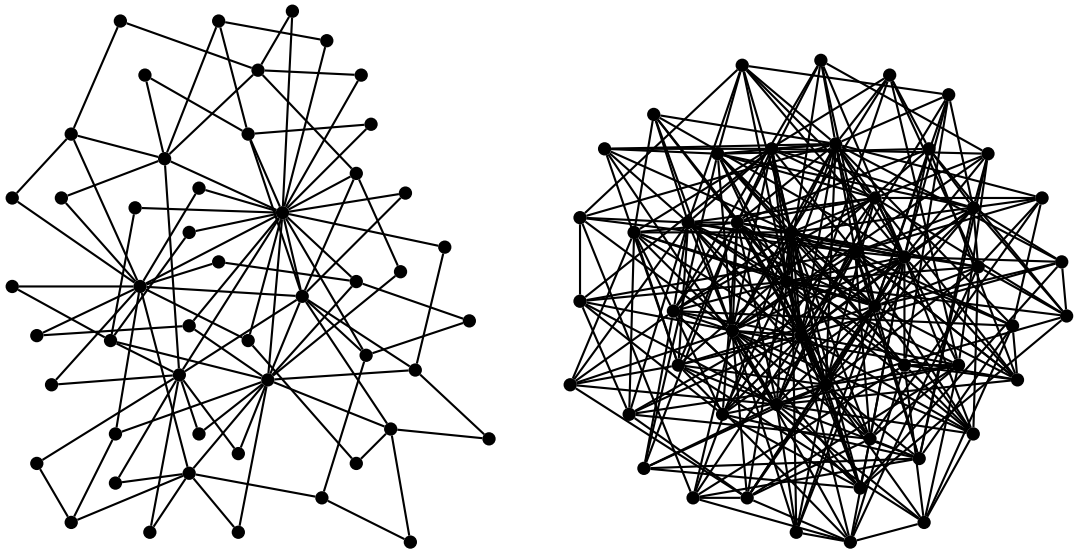


Figure 6.3: Two 50 node instances of the PA class. In the left instance with $\beta = 2$, a large hub node and three lesser hub nodes are visible. The right instance with $\beta = 7$ has a much denser structure.

6.3.3 Impact of our Problem Specific Cutting Planes

In Section 4.4 we have proposed 3-partition inequalities as problem specific cutting planes. We want to use these inequalities to strengthen the linear programming relaxation of the cut-set formulation and give experimental evidence for the effectiveness of these cutting planes in this experiment on the PA instances. To show that the new cutting planes indeed strengthen the relaxation, we measure by how much they advance the linear programming bound: Let L_C and L be the value of an optimum solution of the LP relaxation of the cut-set formulation with and without some additional set of cutting planes C, respectively. Then, the relative progress made by the cutting planes in C is $(L_C - L)/L \cdot 100$. We depict the average relative progress made with different separation strategies in Table 6.2. In the table, the separation strategy EnumZH refers to the enumerative 3-partition separation from Section 4.4.2, the strategy MinCutZH uses the minimum-cut based separation procedure from Section 5.3.2 and the strategy ACF uses the tabu search by Andreello, Caprara and Fischetti [ACF07], as described in Section 6.1.1. In particular, this algorithm can combine an arbitrary number of inequalities to build a $\{0, \frac{1}{2}\}$-cut. It can also compute higher rank $\{0, \frac{1}{2}\}$-cuts because it is not limited to combining cut-set inequalities. Finally, the EnumZH+ACF strategy first executes the EnumZH algorithm and only if this call is not successful, it starts the ACF separation.

From the pure algorithms (EnumZH, MinCutZH and ACF) the enumerative algorithm EnumZH makes the largest progress in closing the root gap. On all instance sizes, it performs significantly better than the other two algorithms. The MinCutZH algorithm is never worse than ACF and starting from instances with 45 nodes, it is significantly better. The mixed algorithm EnumZH+ACF is marginally better than the other three variants. In total, the average relative progress made with any of the four separation strategies decreases with the instance size.

$\|V\|$	EnumZH	MinCutZH	ACF	EnumZH+ACF
20	5.5%	3.2%	3.2%	5.6%
25	5.1%	2.6%	2.3%	5.2%
30	3.6%	1.7%	1.3%	3.6%
35	4.2%	2.0%	1.7%	4.3%
40	3.7%	1.8%	1.6%	3.8%
45	3.2%	1.4%	1.0%	3.2%
50	3.6%	1.8%	1.3%	3.7%
60	3.3%	1.6%	1.2%	3.4%
70	2.9%	1.4%	0.9%	2.9%
80	2.9%	1.4%	0.9%	2.9%
90	3.0%	1.5%	1.0%	3.0%
100	2.9%	1.4%	0.8%	2.9%

Table 6.2: Progress made with the problem specific cutting planes at the root node on the PA sRND-F instances.

In total, the best choice for our algorithm is EnumZH+ACF and we will see in the next experiment that this remains true if we also consider the running time of the algorithms.

It is surprising that the general $\{0, \frac{1}{2}\}$-cut separation ACF does not improve significantly on EnumZH, even though it can separate much more general cutting planes in theory. This behavior could indicate that most of the general cutting planes found by ACF are actually 3-partition inequalities.

6.3.4 Comparison with a Default CPLEX Implementation

The next experiment answers several questions at the same time.

- Ideally, the theoretical results from the previous chapters should have practical implications. Can our algorithm compete with a state-of-the-art MIP solver that would be used in a "hands-on" practical approach?
- In which way is the performance of our algorithm influenced by an instance's characteristics?
- What are the boundaries of our approach? In particular, what is the maximum size of an instance that we can expect to solve?

The arc-flow formulation from Chapter 2 yields a very simple approach for the sRND-F problem: Since no separation algorithm is needed to solve its linear programming relaxation, a practitioner could easily put it into a state-of-the-art commercial MIP solver and hope to solve the problem with the solver's general machinery for MIP problems. Due to the results from Chapter 4, the LP relaxation of the arc-flow formulation yields the same lower bounds as the LP relaxation of our new cut-set formulation and thus, this algorithm might yield comparable results with much less effort. We emulate this approach by solving the arc-flow formulation with CPLEX in sequential mode using the default parameter setting. In particular, we allow CPLEX to use all general MIP cuts. The sequential mode is chosen for fairness: ABACUS does not allow for solving in parallel.

				Cut-Set formulation (CS)							Flow formulation (CPLEX)				
$\|V\|$	$\|E\|$	$\|\mathfrak{B}\|$	lp-gap in %	#solved (m)	cputime	#nodes	root-gap in %	relax-time (t)	sep-time	heur-time	#solved (m)	cputime	#nodes	root-gap in %	relax-time (t)
25	104	5	13.3	3 (0)	1	46	2.9	0 (0)	0	0	3 (0)	0	410	7.7	0 (0)
25	104	10	17.1	3 (0)	24	2016	7.1	0 (0)	3	0	3 (0)	26	2701	12.2	0 (0)
30	121	5	10.6	3 (0)	7	436	2.5	0 (0)	1	0	3 (0)	5	1175	5.6	0 (0)
30	121	10	14.3	3 (0)	125	6875	6.6	0 (0)	15	1	3 (0)	123	12661	9.5	0 (0)
35	155	5	12.3	3 (0)	75	6157	5.3	0 (0)	7	0	3 (0)	9	1808	7.1	0 (0)
35	155	10	12.3	3 (0)	1196	47858	6.2	0 (0)	115	20	3 (0)	597	31355	9.2	0 (0)
40	182	5	17.2	2 (1)	51	1886	6.8	0 (0)	8	0	3 (0)	6	1121	12.0	0 (0)
40	182	10	—	0 (3)	—	—	—	0 (0)	—	—	3 (0)	—	—	—	0 (0)
45	223	5	16.1	1 (2)	15	243	5.6	0 (0)	6	0	3 (0)	10	1106	8.4	0 (0)
45	223	10	—	0 (3)	—	—	—	0 (0)	—	—	1 (0)	—	—	—	0 (0)
50	254	5	—	0 (3)	—	—	—	0 (0)	—	—	2 (0)	—	—	—	0 (0)
50	274	10	—	0 (3)	—	—	—	0 (0)	—	—	0 (0)	—	—	—	0 (0)

Table 6.3: Comparison with the CPLEX flow formulation approach on the JMP instances. All values are averages over those instances that could be solved by both algorithms.

The comparison takes place on two different instance sets: We choose the JMP instances because they are particularly hard; if we can compete on these instances, then we have an indication that our algorithm can compete on many other instances as well. We also compare on the PA class because it has realistic instances with adjustable features. In both cases, we set the time limit to four hours and limit the algorithms to 3GB of memory.

JMP instances.

We show the comparison with the JMP instances in Table 6.3. All entries in the table are averages over those instances that could be solved by both algorithms.

We can solve the JMP instances consistently up to size 35. However, the table shows that with increasing instance size, we have a rapid increase of the running time and the size of the Branch-and-Bound tree. This is particularly true on the instances with 10 scenarios, as the instances with 5 scenarios seem to be slightly easier to solve. We can also see that our separation algorithms are fast; only 10–15 percent of the overall running time are spent in the separation routines. Nonetheless, the 3-partition inequalities and the $\{0, \frac{1}{2}\}$-cuts can reduce the gap between the optimum integer solution and the best linear programming bound significantly already at the root node. This can be seen by comparing the `lp-gap` column with the `root gap` column. The linear programming relaxation at the root node is solved in less than a second in all cases. The heuristics also work very quickly, despite the fact that the rounding heuristics are called at each Branch-and-Bound node.

With the CPLEX arc-flow formulation, it is possible to solve all instances with up to 45 nodes, while for larger instances, CPLEX starts to reach the time limit. Memory, however, is not an issue for the CPLEX implementation. For instances with up to 30 nodes, CPLEX needs a significantly larger Branch-and-Bound tree than we do, even though the instances are solved in comparable time. This behavior changes as the instances grow larger than 30 nodes; then, CPLEX needs less computing time and generates less Branch-and-Bound nodes than our Cut-Set approach. This is despite the fact that the cut-set formulation closes a

					RND			CPLEX		
$\|V\|$	p	$\|\mathfrak{B}\|$	lp-opt	opt	root-lb	root-ub	#nodes	root-lb	root-ub	#nodes
40	0.25	5	25275	32350	30228	38257	3519	27146	38417	1583
40	0.5	5	22619	25849	24780	30502	255	23806	26166	660
40	1	5	38035	44964	41225	51384	57033^{M}	39210	53044	84197
40	0.25	10	33995	42848	38167	50280	61547^{M}	35577	54045	462188
40	0.5	10	39436	49114	44239	59830	55607^{M}	41721	63156	160860
40	1	10	50240	57958	53412	68162	53627^{M}	52232	81339	201836
45	0.25	5	23000	27417	25892	30265	244	25109	30841	1106
45	0.5	5	49970	57780	53694	66873	46829^{M}	51933	73722	73475
45	1	5	58736	65339	61530	76289	47971^{M}	60609	79176	92424
45	0.25	10	36081	–	40639	50368	48291^{M}	37639	55133	308260^{T}
45	0.5	10	51550	–	55678	71383	44615^{M}	54277	71861	151060^{T}
45	1	10	71695	77205	74121	84471	47683^{M}	72885	88967	86614

Table 6.4: More detailed picture of the comparison with the CPLEX flow formulation approach. The instances in the table were (mostly) solved to optimality by CPLEX, but not by our approach. The superscripts T and M denote the instances where an algorithm reached the time or the memory limit, respectively.

significantly larger part of the root gap.

Table 6.4 shows a more detailed picture of those instances that could be solved by CPLEX, but not by our Cut-Set formulation. We observe that the lower bounds provided by our separation algorithms are stronger than the ones computed by CPLEX in all cases. Likewise, with the exception of one instance, the upper bounds computed by our heuristics are better than the ones derived by CPLEX. Still, the highly space efficient commercial Branch-and-Bound implementation of the CPLEX MIP solver is able to fit (at least) ten times more Branch-and-Bound nodes in the 3GB of available memory.

We conclude that on this instance set, the naïve CPLEX approach solves more and larger instances than our problem specific algorithm. Part of this success is due to the commercial implementation that is able to process Branch-and-Bound nodes in less time and space than our research code, enabling it to solve a formulation with weaker bounds through brute force enumeration. However, the fact that the CPLEX implementation sometimes needs *fewer* Branch-and-Bound nodes than our algorithm could indicate a superior branching rule. It could also be a consequence of CPLEX separating general MIP cuts in the Branch-and-Bound nodes. This is opposed to our implementation that separates integer feasibility cuts (i.e., 3-partition inequalities) only at the root node. Still, the CPLEX implementation could be immediately improved by initializing it with our better bounds. In a sense, the better performance of the CPLEX implementation is not unexpected: The JMP instances have relatively few scenarios, making the arc-flow formulation a reasonable choice.

PA instances.

The PA instances allow for a more detailed analysis; in particular, we explore how the balance of the comparison shifts as the scenario set grows larger.

Table 6.5 shows the results of the comparison. Here, we can solve instances with up to 45 nodes and 100 scenarios consistently using our Cut-Set based Branch-and-Cut algorithm where, on average, our algorithm stops successfully after less than 2 minutes.

				Cut-Set formulation (CS)							Flow formulation (CPLEX)				
$\|V\|$	$\|E\|$	$\|\mathfrak{B}\|$	lp-gap in %	#solved (m)	cputime	#nodes	root-gap in %	relax-time (t)	sep-time	heur-time	#solved (m)	cputime	#nodes	root-gap in %	relax-time (t)
20	76	5	8.1	180 (0)	0	21	2.5	0 (0)	0	0	180 (0)	0	43	4.6	0 (0)
20	76	10	8.6	180 (0)	0	51	3.2	0 (0)	0	0	180 (0)	0	101	5.5	0 (0)
20	76	30	7.8	180 (0)	0	45	3.0	0 (0)	0	0	180 (0)	3	102	5.0	0 (0)
20	76	50	7.3	180 (0)	0	42	2.7	0 (0)	0	0	180 (0)	7	88	4.8	0 (0)
20	76	75	6.8	180 (0)	0	31	2.4	0 (0)	0	0	180 (0)	15	80	4.4	0 (0)
20	76	100	6.3	180 (0)	0	25	2.0	0 (0)	0	0	180 (0)	19	63	4.0	1 (0)
25	98	5	9.3	180 (0)	1	163	3.7	0 (0)	0	0	180 (0)	0	258	6.0	0 (0)
25	98	10	9.7	180 (0)	5	489	4.4	0 (0)	1	0	180 (0)	7	651	6.8	0 (0)
25	98	30	8.4	180 (0)	6	366	3.9	0 (0)	2	0	180 (0)	57	684	6.2	0 (0)
25	98	50	8.0	180 (0)	7	323	3.7	0 (0)	3	0	180 (0)	140	681	6.0	1 (0)
25	98	75	7.7	180 (0)	9	312	3.6	0 (0)	4	0	180 (0)	306	693	5.8	2 (0)
25	98	100	7.4	180 (0)	10	319	3.4	0 (0)	6	0	180 (0)	497	689	5.6	4 (0)
30	121	5	6.3	180 (0)	3	213	2.5	0 (0)	0	0	180 (0)	1	212	4.0	0 (0)
30	121	10	6.6	180 (0)	9	529	2.8	0 (0)	2	0	180 (0)	15	469	4.5	0 (0)
30	121	30	5.9	180 (0)	8	322	2.6	0 (0)	3	0	180 (0)	122	469	4.1	0 (0)
30	121	50	5.4	180 (0)	12	365	2.4	0 (0)	5	0	180 (0)	399	541	3.9	2 (0)
30	121	75	5.2	180 (0)	10	233	2.4	0 (0)	5	0	178 (0)	544	397	3.8	5 (0)
30	121	100	4.9	180 (0)	9	178	2.3	0 (0)	5	0	172 (0)	792	368	3.6	8 (0)
35	143	5	7.7	180 (0)	8	399	3.2	0 (0)	1	0	180 (0)	2	326	4.9	0 (0)
35	143	10	8.2	180 (0)	36	1610	3.9	0 (0)	6	0	180 (0)	39	967	5.7	0 (0)
35	143	30	7.6	180 (0)	79	2609	3.8	0 (0)	21	2	180 (0)	582	1950	5.7	1 (0)
35	143	50	7.1	180 (0)	57	1539	3.6	0 (0)	21	2	180 (0)	1327	1447	5.3	4 (0)
35	143	75	6.6	180 (0)	45	998	3.3	0 (0)	21	2	174 (0)	2540	1216	5.0	10 (0)
35	143	100	5.6	180 (0)	30	488	2.8	0 (0)	16	2	158 (0)	2906	729	4.2	14 (0)
40	166	5	6.6	180 (0)	6	259	2.4	0 (0)	1	0	180 (0)	2	283	3.8	0 (0)
40	166	10	6.7	180 (0)	15	568	2.8	0 (0)	3	0	180 (0)	24	510	4.2	0 (0)
40	166	30	6.0	180 (0)	28	772	2.5	0 (0)	9	0	180 (0)	338	788	4.0	2 (0)
40	166	50	5.5	180 (0)	26	582	2.4	0 (0)	11	1	179 (0)	944	742	3.8	6 (0)
40	166	75	4.9	180 (0)	18	357	2.1	0 (0)	10	1	169 (0)	1496	550	3.4	13 (0)
40	166	100	4.4	180 (0)	11	187	1.8	0 (0)	7	0	132 (30)	1405	387	3.1	21 (0)
45	188	5	5.8	180 (0)	6	226	2.1	0 (0)	2	0	180 (0)	1	235	3.2	0 (0)
45	188	10	6.2	180 (0)	15	576	2.6	0 (0)	5	0	180 (0)	27	518	4.0	0 (0)
45	188	30	5.2	180 (0)	21	553	2.3	0 (0)	9	0	180 (0)	318	645	3.6	2 (0)
45	188	50	4.8	180 (0)	23	477	2.2	0 (0)	13	0	180 (0)	999	629	3.4	9 (0)
45	188	75	4.3	180 (0)	20	322	2.0	0 (0)	13	1	174 (0)	1985	572	3.2	21 (0)
45	188	100	4.2	180 (0)	21	374	1.9	0 (0)	13	1	89 (90)	1883	645	3.0	26 (30)
50	211	5	6.9	173 (7)	113	3042	2.9	0 (0)	14	0	180 (0)	17	909	4.0	0 (0)
50	211	10	6.2	152 (28)	196	4766	2.8	0 (0)	29	2	180 (0)	102	1372	4.0	0 (0)
50	211	30	5.2	146 (34)	192	3900	2.4	0 (0)	52	5	146 (0)	913	1743	3.5	5 (0)
50	211	50	4.7	143 (37)	167	3083	2.2	0 (0)	61	6	133 (0)	1664	1721	3.2	18 (0)
50	211	75	4.4	153 (27)	92	1672	2.0	0 (0)	46	5	117 (29)	2101	1064	3.0	38 (0)
50	211	100	4.7	152 (28)	58	1099	2.5	0 (0)	33	3	45 (120)	1356	888	3.1	23 (60)
60	256	5	6.3	169 (11)	165	3272	2.7	0 (0)	20	1	180 (0)	20	842	3.7	0 (0)
60	256	10	5.3	133 (47)	269	5525	2.4	0 (0)	43	4	177 (0)	125	1407	3.3	0 (0)
60	256	30	4.5	135 (45)	279	4841	2.2	0 (0)	79	9	142 (0)	1290	1884	3.0	12 (0)
60	256	50	3.9	134 (46)	154	2409	1.9	0 (0)	61	6	119 (0)	2523	1458	2.6	40 (0)
60	256	75	3.8	138 (42)	103	1623	1.8	0 (0)	52	5	68 (90)	2112	1188	2.6	56 (30)
60	256	100	4.0	141 (39)	24	477	1.5	0 (0)	17	1	30 (150)	382	509	2.3	169 (60)
70	301	5	4.5	135 (45)	181	2909	1.9	0 (0)	23	1	180 (0)	21	694	2.5	0 (0)
70	301	10	3.5	109 (71)	228	3921	1.5	0 (0)	40	3	160 (0)	92	919	2.2	1 (0)
70	301	30	3.0	107 (73)	212	2979	1.3	0 (0)	61	5	120 (0)	821	1042	1.9	18 (0)
70	301	50	2.6	110 (70)	219	2399	1.3	0 (0)	77	7	111 (0)	2278	867	1.8	80 (0)
70	301	75	3.3	110 (70)	279	3899	1.4	0 (0)	138	13	44 (120)	1431	1111	2.2	121 (60)
70	301	100	—	107 (72)	—	—	—	0 (0)	—	—	0 (180)	—	—	—	129 (120)
80	346	5	3.3	107 (73)	169	2260	1.2	0 (0)	21	1	180 (0)	10	405	1.8	0 (0)
80	346	10	2.6	91 (89)	148	2649	1.1	0 (0)	28	1	145 (0)	48	440	1.5	2 (0)
80	346	30	2.5	93 (87)	368	3951	1.3	0 (0)	95	7	100 (0)	1215	765	1.6	39 (0)
80	346	50	3.2	93 (85)	366	3652	1.5	0 (0)	131	12	66 (60)	2402	781	2.0	131 (0)
80	346	75	3.6	89 (87)	102	1527	1.3	0 (0)	64	4	30 (150)	512	634	2.0	304 (60)
80	346	100	—	88 (90)	—	—	—	0 (0)	—	—	0 (180)	—	—	—	29 (150)
90	391	5	2.7	88 (92)	330	4393	1.1	0 (0)	41	2	177 (0)	11	381	1.4	0 (0)
90	391	10	1.2	68 (111)	136	2142	0.6	0 (0)	30	1	121 (0)	31	223	0.7	3 (0)
90	391	30	1.0	67 (111)	261	1603	0.7	0 (0)	55	3	85 (0)	2070	640	0.8	66 (0)
90	391	50	1.6	71 (109)	634	7437	0.9	0 (0)	262	21	49 (90)	1544	1814	1.1	141 (30)
90	391	75	2.1	72 (108)	1285	11281	1.2	0 (0)	586	47	23 (150)	1767	1426	1.4	220 (120)
90	391	100	—	72 (107)	—	—	—	—	—	—	0 (180)	—	—	—	—
100	436	5	2.0	81 (99)	186	2042	0.7	0 (0)	25	0	170 (0)	6	230	1.0	0 (0)
100	436	10	1.3	71 (109)	264	3055	0.7	0 (0)	49	2	103 (0)	104	382	0.8	4 (0)
100	436	30	1.2	67 (108)	383	3043	0.7	1 (0)	100	7	63 (0)	1753	774	0.9	104 (0)
100	436	50	1.7	66 (108)	817	6624	1.0	1 (0)	311	23	38 (90)	2010	1425	1.2	221 (30)
100	436	75	—	62 (107)	—	—	—	0 (0)	—	—	0 (180)	—	—	—	20 (150)
100	436	100	—	59 (110)	—	—	—	—	—	—	0 (180)	—	—	—	—

Table 6.5: Comparison of the arc-flow formulation and the cut-set formulation on the instances of the PA set.

Still, the number of solved instances decreases as the size of the instances increases above 45. Here, if the algorithm failed to solve an instance, it was due to the memory limit in the large majority of cases. Generally, instances with 5 scenarios seem to be easier for our algorithm than instances with more than 5 scenarios. Nonetheless, we are able to solve 59 out of the 180 largest instances (100 nodes, 100 scenarios).

As before, our separation algorithms already close a significant amount of the `lp-gap` at the root node, although the effect is less pronounced than in the `JMP` instances. This could be because the `lp-gap` is generally smaller here. The separation is still fast, but we observe a clear dependence of the separation time on the number of scenarios. On the instances with 100 scenarios, about one half of the cputime is spent in separation routines. This is despite the fact that the size of the cut-set formulation itself does *not* depend on the number of scenarios. Yet, the cut-set formulation at the root node can be solved in less than 2 seconds for all instances sizes. The heuristics remain fast even for large instances, regardless of the size of the scenario set.

On the other hand, we can (at least) solve instances of up to 100 nodes with the `CPLEX` implementation if the scenario set is small. As the size of the scenario set increases, however, the `CPLEX` implementation needs more and more time to solve the linear programming relaxation at the root node. Starting from the 45 node instances, the `CPLEX` implementation can no longer solve the root linear programming relaxation of a larger part of the instances with large scenario sets. Given that the size of the arc-flow formulation depends on the number of scenarios, this is to be expected. This effect has a clear impact on the overall performance of the `CPLEX` implementation; starting at the instances with 30 nodes, the `CPLEX` implementation has difficulties solving instances with 75 or 100 scenarios. This is in contrast to the performance of the cut-set based algorithm. Finally, starting from 70 nodes, the `CPLEX` implementation is no longer able to solve any of the instances with 100 scenarios to optimality. Additionally, on the instances that can be solved to optimality, our Cut-Set based algorithm needs significantly less cpu time than the `CPLEX` implementation if the scenarios set size is at least 30. As before, the `CPLEX` implementation closes less `lp gap` than our cut-set based approach.

Table 6.6 shows the same computational results of our cut-set formulation based algorithm in a different grouping. It allows us to analyze the influence of the network's density on the performance of the algorithm. From left to right, the network topologies become denser as the β parameter of the preferential attachment model increases (we have argued previously that an instance with n nodes has roughly βn edges). As before, all instances with at most 45 nodes are solved without problems, but it is apparent now that both the running time of our algorithm and the number of Branch-and-Bound nodes that it generates tend to increase with the instance's density. This is why the sparse instances generated with $\beta = 2$ can be consistently solved to optimality even if they have 80 nodes. At the same time, the algorithm starts to fail on instances with 60 nodes if we suppose a medium density (i.e., $\beta = 3, 4, 5$). For the higher values of β, instances with 50 nodes and many scenarios cause problems. Looking at the maximum size instances with 100 nodes and 100 scenarios, the algorithm still solves almost two thirds of the instances if $\beta = 2$, yet, only few instances can be solved to optimality for the highest density level $\beta = 7$. Still, the density of an instance cannot be the only difficulty: The `PA` instances with 20 nodes and $\beta = 7$ have approximately the same number of edges as the `PA` instances with 70 nodes and $\beta = 2$. However, the (thus

		$\beta = 2$			$\beta = 3$			$\beta = 4$			$\beta = 5$			$\beta = 6$			$\beta = 7$		
$\|V\|$	$\|\mathfrak{B}\|$	#solved	cputime	#nodes	#solved	cputime	#nodes	#solved	cputime	#nodes	#solved	cputime	#nodes	#solved	cputime	#nodes	#solved	cputime	#nodes
20	5	30	0	45	30	0	4	30	0	12	30	0	33	30	0	16	30	0	22
20	10	30	0	98	30	0	9	30	0	12	30	0	128	30	0	24	30	0	44
20	30	30	0	123	30	0	11	30	0	13	30	0	79	30	0	20	30	0	33
20	50	30	0	129	30	0	10	30	0	9	30	1	65	30	0	21	30	1	27
20	75	30	0	91	30	0	9	30	0	9	30	1	35	30	1	22	30	1	30
20	100	30	0	82	30	0	6	30	0	8	30	1	21	30	1	13	30	1	26
25	5	30	1	284	30	0	50	30	4	462	30	0	38	30	0	23	30	2	128
25	10	30	4	681	30	0	103	30	20	1615	30	1	59	30	0	38	30	7	445
25	30	30	6	740	30	1	91	30	19	1015	30	1	52	30	2	47	30	6	260
25	50	30	7	598	30	1	99	30	21	888	30	2	53	30	2	35	30	8	271
25	75	30	7	438	30	1	59	30	30	1041	30	3	79	30	2	38	30	9	222
25	100	30	9	493	30	2	85	30	33	1019	30	3	57	30	3	44	30	10	225
30	5	30	0	18	30	0	46	30	0	46	30	0	53	30	0	25	30	18	1097
30	10	30	0	18	30	1	135	30	0	68	30	1	106	30	1	36	30	53	2817
30	30	30	0	19	30	1	82	30	1	58	30	3	97	30	2	37	30	44	1649
30	50	30	0	18	30	2	115	30	2	51	30	4	97	30	3	35	30	62	1882
30	75	30	0	28	30	3	123	30	2	58	30	5	96	30	3	29	30	73	1699
30	100	30	0	27	30	3	132	30	3	54	30	7	137	30	4	32	30	76	1588
35	5	30	0	86	30	0	24	30	13	663	30	10	494	30	9	464	30	13	671
35	10	30	1	146	30	0	21	30	97	4458	30	14	743	30	38	1631	30	66	2670
35	30	30	2	179	30	1	42	30	331	10608	30	19	703	30	55	1834	30	69	2293
35	50	30	2	129	30	1	36	30	174	4793	30	13	312	30	61	1710	30	93	2260
35	75	30	3	119	30	2	45	30	148	3660	30	17	349	30	65	1430	30	92	1904
35	100	30	4	150	30	2	33	30	165	3522	30	17	272	30	63	1178	30	96	1773
40	5	30	0	68	30	2	180	30	2	127	30	20	813	30	5	206	30	5	166
40	10	30	1	175	30	4	259	30	6	332	30	59	1803	30	10	448	30	9	398
40	30	30	2	180	30	6	263	30	8	267	30	115	2853	30	16	503	30	18	573
40	50	30	2	101	30	7	210	30	8	187	30	148	2782	30	17	399	30	19	431
40	75	30	3	115	30	7	170	30	10	205	30	152	2874	30	22	434	30	27	550
40	100	30	3	99	30	8	140	30	11	198	30	169	2442	30	22	340	30	28	448
45	5	30	1	158	30	1	83	30	7	283	30	5	150	30	14	502	30	8	185
45	10	30	5	506	30	4	268	30	17	599	30	6	191	30	44	1379	30	17	522
45	30	30	4	213	30	5	220	30	27	670	30	9	176	30	54	1460	30	27	583
45	50	30	6	240	30	7	238	30	23	461	30	12	171	30	63	1255	30	29	499
45	75	30	7	222	30	8	175	30	29	457	30	14	164	30	76	1310	30	24	296
45	100	30	11	277	30	12	279	30	48	790	30	16	156	30	81	1228	30	32	350
50	5	30	7	658	30	163	5906	30	5	211	30	36	1046	29	208	4779	24	300	6385
50	10	30	50	3687	26	418	11498	30	6	216	30	111	3092	24	464	7177	12	233	3626
50	30	30	46	2233	25	588	12595	30	7	171	30	119	2714	17	626	6734	14	452	4444
50	50	30	32	1292	24	714	12435	30	10	204	30	200	3618	19	815	8719	10	24	97
50	75	30	32	994	29	528	9049	30	11	179	30	161	2576	23	1057	10113	11	213	1406
50	100	30	38	969	29	889	13116	30	14	204	30	179	2502	22	1302	10172	11	254	1713
60	5	30	5	274	30	104	3302	30	113	2730	30	116	2417	25	416	6593	24	310	5274
60	10	30	9	467	26	394	10179	27	375	8200	24	315	6060	13	377	4318	13	205	2566
60	30	30	12	494	27	428	8884	26	318	5281	29	463	7999	11	295	1736	12	292	2177
60	50	30	16	486	27	287	5302	27	736	10219	26	393	5327	11	402	2222	13	546	6058
60	75	30	16	382	28	293	4616	29	783	9798	28	467	5627	12	675	4027	11	137	1040
60	100	30	24	478	29	492	7019	29	769	8329	29	547	6151	10	34	64	14	866	5624
70	5	30	45	1813	26	284	5962	11	55	577	30	91	1550	19	364	4003	19	283	2873
70	10	30	138	4404	17	460	7760	10	18	63	29	286	4257	12	278	2398	11	100	961
70	30	29	91	2533	18	762	10006	10	79	196	30	155	2055	10	60	129	10	24	36
70	50	30	76	1684	18	894	9856	10	164	317	30	99	1012	12	423	2442	10	47	75
70	75	30	160	3102	19	1218	10847	10	244	415	30	194	1768	11	422	1517	10	90	128
70	100	30	130	2060	17	958	8340	10	416	653	30	187	1527	10	109	164	10	81	102
80	5	30	25	883	29	335	5429	10	4	8	12	250	2071	11	48	354	15	268	1949
80	10	30	128	4297	20	351	4897	10	51	107	11	160	1142	10	15	28	10	17	37
80	30	30	180	4848	21	1059	9704	10	369	586	11	157	551	10	97	205	11	186	664
80	50	30	91	1660	22	1081	9763	10	947	1091	10	212	429	10	112	168	11	307	1321
80	75	30	102	1528	19	915	6746	10	2274	2446	10	116	198	10	130	181	10	141	197
80	100	30	163	2236	18	1166	7204	9	2020	1873	11	435	1646	10	226	264	10	192	291
90	5	24	437	9192	21	585	5893	13	467	3236	10	5	5	10	9	9	10	11	13
90	10	18	436	7975	10	5	17	10	10	21	10	30	64	10	33	47	10	62	68
90	30	15	401	4986	12	321	1991	10	20	34	10	54	79	10	180	204	10	612	531
90	50	18	1292	15610	12	662	3079	11	392	1319	10	258	353	10	276	259	10	1101	803
90	75	17	1285	11282	14	926	4422	11	534	1423	10	220	267	10	433	333	10	1013	616
90	100	19	1646	14305	14	1091	4987	10	77	99	10	228	247	10	765	552	9	4693	2505
100	5	28	288	4151	12	329	2727	11	248	1473	10	9	6	10	14	11	10	9	11
100	10	21	673	10124	10	15	39	10	13	22	10	66	77	10	264	212	10	104	88
100	30	18	813	8393	10	39	86	10	28	36	10	228	185	9	3081	1526	10	1966	1286
100	50	19	1258	10399	10	56	113	10	63	77	10	1283	931	7	6148	2325	10	2806	1353
100	75	19	1378	10285	10	88	154	10	52	54	10	1342	863	6	5186	2111	7	5246	2186
100	100	18	1644	10274	10	139	206	10	99	106	10	2990	1761	5	4606	1154	6	3523	1306

Table 6.6: Computational results of the cut-set set based Branch-and-Cut algorithm. The results are grouped according to the density of the instances. An instance with n nodes has approximately βn edges.

sparser) instances with 70 nodes are clearly more difficult for our algorithm. Additionally, there are some unexplained outliers in the $\beta = 4$ case (instances with 25 and 35 nodes).

From Tables 6.5 and 6.6 we conclude that the `PA` instances are generally easier for our algorithm than the `JMP` instances. Comparing the average number of edges, they are also slightly sparser, i.e., for a fixed number of nodes, the `PA` instances have less edges. Indeed, Table 6.6 confirms that the running time of our algorithm increases with the density of an instance if the number of nodes is fixed. The running time also increases with the size of the instances, regardless of its density.

The running time of both the arc-flow and the cut-set based algorithm depends on the number of scenarios; however, the dependency of the cut-set based algorithm is much less pronounced. This makes the cut-set based algorithm a better choice for instances with more than 10 scenarios: Here it clearly outperforms the "hands-on" `CPLEX` implementation. We see an improvement not only in terms of bounds and CPU time, but also in the number of instances that can be solved.

Even though the worst-case running time of the cut-set separation algorithm has a dependency on the size of the scenario set, it is easily possible to solve the linear programming relaxation of the cut-set formulation even on large instances with many scenarios. This is not true for the arc-flow formulation that seems to grow too large to be solved efficiently on medium sized instances with a medium number of scenarios. This is a clear advantage of the cut-set formulation.

Conclusions

Our experiments show that the cut-set based algorithm improves upon a previous algorithm by Buchheim, Liers and Sanità [BLS11]: it is mostly faster and solves more instances.

If the scenario set has up to 10 scenarios, a simple arc-flow based Branch-and-Bound implementation in a commercial solver is faster than our algorithm. However, it is not clear how much of the speedup is due to the streamlined commercial code. Still, even on very hard instances, we can handle networks of up to 35 nodes. If the scenario set has 15 or more scenarios, the cut-set based algorithm is preferable. It is able to solve the larger part of the `PA` instances with up to 70 nodes and up to 100 scenarios. On these instance sizes, the commercial solver is no longer able to consistently solve the linear programming relaxation at the root node. Also, the solutions are computed in reasonable time. The difficulty of the instances increases with their density and, in general, with the number of edges in the instance, but also with the size of the scenario set. This is true for both the cut-set based and the arc-flow based algorithm. In all cases, memory is more limiting for our algorithm than time. Our algorithm generally produces much better root-bounds than the arc-flow formulation, however, it does not always make use of this advantage. Here, a better branching rule could help. Nonetheless, our problem specific separation routines are fast and effective, with the `EnumZH+ACF` separation being method of choice for the 3-partition separation.

		t						t			
$\|V\|$	a	0.25	0.5	0.75	1.0	$\|V\|$	a	0.25	0.5	0.75	1
10	2	8	7	7	6	10	2	1	11	39	153
10	3	9	8	9	6	10	3	1	3	18	117
10	4	7	9	6	5	10	4	1	8	34	89
10	5	7	6	9	9	10	5	1	8	21	144
10	6	6	8	8	5	10	6	1	8	25	92
10	7	7	9	7	7	10	7	1	8	20	207
15	2	8	5	7	4	15	2	2	7	263	2625
15	3	6	9	8	8	15	3	2	36	205	1433
15	4	7	9	7	7	15	4	2	12	95	2051
15	5	6	8	10	8	15	5	2	28	235	1217
15	6	8	8	6	8	15	6	2	15	358	1489
15	7	7	5	6	8	15	7	3	19	71	1443
20	2	9	10	7	8	20	2	7	207	7090	29302
20	3	6	7	7	6	20	3	6	126	594	9668
20	4	8	9	3	7	20	4	6	86	2848	12644
20	5	7	9	7	8	20	5	9	82	4110	72987
20	6	4	7	9	9	20	6	3	118	1323	15300
20	7	8	9	6	7	20	7	6	95	1134	15654
25	2	7	8	5	5	25	2	6	297	8556	49176
25	3	8	7	6	6	25	3	17	307	1355	109225
25	4	6	6	8	1	25	4	10	442	19433	115704
25	5	7	9	7	5	25	5	10	210	4542	84910
25	6	8	8	8	6	25	6	8	808	5126	52224
25	7	5	7	6	6	25	7	9	321	9294	106710

Table 6.7: Using PORTA to convert the linear description of the instances from the `PA` Hose class in the geometric demand distribution. The grouping in the table is by the percentage $t \in \{0.25, 0.5, 0.75, 1.0\}$ of terminal nodes. *On the left:* The number of instances that could be converted within 1800 seconds. *On the right:* Resulting average number of vertices of the demand polytope.

6.3.5 Results on the Hose Instance Sets

The arc-flow formulation only works on a Hose instance if we first find all the vertices of its uncertainty set. We evaluate this approach by computing the vertices of the Hose uncertainty sets on the `PA-geometric` instances. To that aim, we use the `PORTA` software package [CL08] with a time limit of 1800s. The results are depicted in Table 6.7.

We see that only in very few cases, we are able to convert all ten instances of a given size and density, even if the instance size is tiny. If the conversion is successful the resulting finite description can be large; on instances with 10 nodes we obtain up to 200 scenarios. On instances with 25 nodes, the number of scenarios already ranges in the order of 10^5 to 10^6 on average. Looking at the results from the previous section, we cannot expect to solve these `sRND-F` instances with the `CPLEX` arc-flow based algorithm. Instances with few terminals would be solvable with both `sRND-F` algorithms, but even those instances cannot be converted reliably.

We conclude that it is no longer possible to compare with the `CPLEX` flow formulation. There is also no algorithm that we could use as a reference point for a comparison. However, we can still analyze the practical performance of our `sRND-H` Branch-and-Cut algorithm and try to find answers to the following questions.

- Can we solve instances of non-trivial size with our `sRND-H` algorithm? In particular, is the MIP separation practical?
- Does the distribution of the Hose intervals influence the running time of our algorithm?
- Where are the limits of our algorithm?

			$\beta = 2$			$\beta = 3$			$\beta = 4$			$\beta = 5$			$\beta = 6$			$\beta = 7$		
	$\|V\|$	$\|E\|$	#solved	cputime	#nodes	#solved	cputime	#nodes	#solved	cputime	#nodes	#solved	cputime	#nodes	#solved	cputime	#nodes	#solved	cputime	#nodes
geometric	10	42	40	0	21	40	0	21	40	0	23	40	0	25	40	0	26	40	0	29
	15	77	40	0	17	40	0	40	40	0	41	40	0	56	40	0	54	40	0	52
	20	112	40	0	56	40	0	69	40	0	83	40	0	77	40	0	75	40	0	82
	25	147	40	0	86	40	0	76	40	4	138	40	0	106	40	0	115	40	0	121
	30	182	40	0	97	40	0	116	40	0	114	40	0	148	40	0	161	40	21	216
	35	217	40	0	118	40	0	140	40	3	193	40	0	175	40	3	261	40	5	274
	40	252	40	0	128	40	0	225	40	0	213	40	2	293	40	3	270	40	2	265
	45	287	40	0	185	40	0	161	40	1	299	40	9	277	40	10	369	40	0	253
	50	322	40	0	208	40	5	417	40	1	281	40	33	583	40	344	573	40	224	649
	60	392	40	0	278	40	6	429	40	25	757	40	57	774	39	579	898	39	680	973
	70	462	40	3	361	40	13	920	36	731	2121	40	4	821	37	357	1830	40	46	1400
	80	532	40	1	499	40	11	1035	31	436	2908	40	207	2083	38	547	2418	39	322	2366
	90	602	40	7	824	40	98	1917	40	23	2068	35	785	3212	37	1041	3938	34	677	4799
	100	672	40	56	1410	39	400	3524	40	525	3442	31	968	6508	19	645	7996	19	965	7299
uniform	10	42	40	0	23	40	0	24	40	0	23	40	0	29	40	0	29	40	0	28
	15	77	40	0	21	40	0	41	40	0	44	40	0	54	40	0	51	40	0	56
	20	112	40	0	59	40	0	71	40	0	85	40	0	90	40	0	92	40	0	92
	25	147	40	0	92	40	0	92	40	25	166	40	0	117	40	0	116	40	5	130
	30	182	40	0	98	40	0	125	40	0	125	40	0	154	40	1	174	40	84	259
	35	217	40	0	131	40	0	142	40	1	215	40	1	197	40	5	278	40	8	292
	40	252	40	0	144	40	0	220	40	1	242	40	6	322	40	4	328	40	3	264
	45	287	40	1	200	40	0	190	40	3	336	40	1	313	40	10	384	40	0	290
	50	322	40	1	240	40	53	445	40	0	311	40	45	639	40	139	718	40	94	720
	60	392	40	1	297	40	11	492	40	255	862	40	125	962	37	638	988	38	162	1250
	70	462	40	5	436	40	42	1018	28	1808	2539	40	5	818	38	698	2030	40	708	1735
	80	532	40	7	547	40	16	1172	30	1142	3588	37	364	2302	31	459	2628	37	852	2544
	90	602	40	119	977	39	381	2066	40	20	2045	37	1090	3705	34	1081	3823	33	944	5423
	100	672	40	213	1782	35	806	4401	40	163	3770	33	1439	7469	12	1232	7347	18	1730	7706
zero-one	10	42	40	0	18	40	0	18	40	0	19	40	0	24	40	0	27	40	0	20
	15	77	40	0	21	40	0	35	40	0	33	40	0	52	40	0	39	40	0	35
	20	112	40	0	67	40	0	62	40	0	93	40	0	80	40	0	82	40	0	90
	25	147	40	0	83	40	0	76	40	1	211	40	0	111	40	0	117	40	0	137
	30	182	40	0	87	40	0	106	40	0	118	40	0	141	40	0	181	40	4	329
	35	217	40	0	115	40	0	118	40	0	163	40	0	119	40	0	245	40	0	247
	40	252	40	0	149	40	0	231	40	0	213	40	0	321	40	0	376	40	1	285
	45	287	40	0	154	40	0	114	40	0	230	40	0	213	40	2	332	40	0	169
	50	322	40	0	184	40	1	372	40	0	211	40	1	506	40	6	600	40	4	565
	60	392	40	0	217	40	0	346	40	2	667	40	14	1063	40	37	1169	40	3	873
	70	462	40	0	315	40	2	681	40	37	2718	40	1	540	40	12	2010	40	7	1245
	80	532	40	0	390	40	3	842	40	19	2896	40	7	1682	40	317	3399	40	13	2352
	90	602	40	1	506	40	4	1062	40	7	1365	40	21	2756	40	19	2546	40	30	3733
	100	672	40	6	1131	40	16	2380	40	16	2391	40	58	6271	31	924	14748	39	100	7632

Table 6.8: Computational results on the Hose `PA` instances.

PA instances

In our first experiment, we run the cut-set based Branch-and-Cut algorithm for the `sRND-H` problem on the `PA` instances and report the aggregated results in Table 6.8. The algorithm was limited to four hours of time and 3GB of memory.

The table shows that for all three interval distributions (`geometric`, `uniform` and `zero-one`), the running time of our algorithm increases with the instance size and density. The higher the density and the higher the size, the fewer instances can be solved, the more running time is needed to solve the instances and the more Branch-and-Cut nodes are generated. Still, we are able to solve almost all `PA-geometric` and `PA-uniform` instances with up to 100 nodes if $\beta \in \{2, 3, 4\}$. Equally, we can solve all instances with at most 50 nodes, regardless of their density. The `PA-geometric` and `PA-uniform` instances with more than 80 nodes and a density parameter of $\beta = 6, 7$ cannot be solved reliably. In comparison, the `PA-uniform` instances require slightly more running time than the `PA-geometric` instances on average, and not as many of the instances can be solved. This is true independently of size and density. On the other hand, we were able to solve all but one of the `PA-zero-one` instances in less than two minutes. Still, also on these instances, more effort is needed as

	$\|V\|$	$\|E\|$	lp-gap (in %)	#solved (m)	cputime	#nodes	root-gap (in %)	relax-time (m)	sep-time	ip-sep-time	ipsep calls (in %)	heur-time
geometric	10	39	0.32	40 (0)	0	25	0.00	0 (0)	0	0	14.07	0
	15	69	0.24	40 (0)	0	53	0.03	0 (0)	0	0	10.45	0
	20	99	0.17	40 (0)	0	74	0.03	0 (0)	0	0	10.42	0
	25	129	0.22	40 (0)	0	114	0.03	0 (0)	0	0	9.33	0
	30	159	0.08	40 (0)	0	160	0.02	0 (0)	0	0	10.27	0
	35	189	0.16	40 (0)	3	260	0.08	0 (0)	2	2	12.60	0
	40	219	0.10	40 (0)	3	269	0.04	0 (0)	3	2	9.92	0
	45	249	0.19	40 (0)	10	368	0.09	0 (0)	8	6	13.10	0
	50	279	0.15	40 (0)	344	572	0.10	2 (0)	300	284	16.88	0
	60	339	0.13	39 (0)	579	897	0.11	6 (0)	517	486	16.26	0
	70	399	0.12	37 (0)	357	1829	0.09	15 (0)	330	303	8.31	0
	80	459	0.08	38 (0)	547	2417	0.06	26 (0)	481	440	11.93	0
	90	519	0.13	37 (0)	1041	3937	0.10	43 (0)	897	751	8.74	0
	100	579	0.07	19 (0)	645	7995	0.06	219 (0)	519	373	3.93	0
uniform	10	39	0.28	40 (0)	0	28	0.01	0 (0)	0	0	14.68	0
	15	69	0.31	40 (0)	0	50	0.05	0 (0)	0	0	13.50	0
	20	99	0.32	40 (0)	0	91	0.06	0 (0)	0	0	11.70	0
	25	129	0.16	40 (0)	0	115	0.05	0 (0)	0	0	11.59	0
	30	159	0.37	40 (0)	1	173	0.08	0 (0)	0	0	13.47	0
	35	189	0.20	40 (0)	5	277	0.11	0 (0)	4	3	14.83	0
	40	219	0.19	40 (0)	4	327	0.08	0 (0)	3	2	13.90	0
	45	249	0.11	40 (0)	10	383	0.07	1 (0)	8	7	13.06	0
	50	279	0.16	40 (0)	139	717	0.11	4 (0)	127	120	18.38	0
	60	339	0.13	37 (0)	638	987	0.10	9 (0)	570	543	18.25	0
	70	399	0.11	38 (0)	698	2029	0.09	17 (0)	631	572	12.69	0
	80	459	0.06	31 (0)	459	2627	0.05	38 (0)	413	382	11.66	0
	90	519	0.08	34 (0)	1081	3822	0.07	50 (0)	962	849	10.00	0
	100	579	0.12	12 (0)	1232	7346	0.10	391 (0)	1065	877	4.57	0
zero-one	10	39	1.53	40 (0)	0	26	0.00	0 (0)	0	0	15.81	0
	15	69	2.03	40 (0)	0	38	0.31	0 (0)	0	0	17.27	0
	20	99	1.11	40 (0)	0	81	0.10	0 (0)	0	0	12.87	0
	25	129	0.22	40 (0)	0	116	0.09	0 (0)	0	0	10.31	0
	30	159	0.66	40 (0)	0	180	0.07	0 (0)	0	0	10.64	0
	35	189	0.19	40 (0)	0	244	0.08	0 (0)	0	0	8.17	0
	40	219	0.05	40 (0)	0	375	0.05	0 (0)	0	0	7.13	0
	45	249	0.51	40 (0)	2	331	0.16	0 (0)	2	0	10.17	0
	50	279	0.22	40 (0)	6	599	0.10	2 (0)	5	3	9.70	0
	60	339	0.49	40 (0)	37	1168	0.24	10 (0)	33	25	13.97	0
	70	399	0.07	40 (0)	12	2009	0.00	11 (0)	9	3	3.52	0
	80	459	0.32	40 (0)	317	3398	0.32	47 (0)	293	269	14.46	0
	90	519	0.00	40 (0)	19	2545	0.00	17 (0)	13	2	2.00	0
	100	579	0.26	31 (0)	924	14747	0.17	604 (0)	832	713	6.98	0

Table 6.9: Detailed computational results on the Hose `PA` instances with $\beta = 6$.

instance size and density increase.

In conclusion, we are able to solve instances of non-trivial size in all three interval distribution models. The running time of our algorithm clearly depends on both instance size and density. The `zero-one` distribution of the intervals seems to be easier for our algorithm than the other two.

In Table 6.8 our algorithm performs worst for $\beta = 6$. We consider the computational results for this case in more detail in Table 6.9. The more detailed table reveals that on the depicted `PA` Hose instances, memory is no longer the limiting factor (as it was in the `sRND-F` case): If the algorithm failed to solve an instance, then it did so because of the time limit of four hours. Also, the time needed for the separation increases with the instance size and makes up a large part of the overall running time of the algorithm. This is in contrast to the `sRND-F` case as well. The `ip-sep-time` column shows how much time was spent by solving the separation MIP for Hose cut-set inequalities. The `ip-sep-time` increases with the instance size and we see that in almost all cases, the Hose MIP separation makes the largest contribution to both the separation time and the overall running time. In the

			geometric			uniform			zero-one		
	$\|V\|$	$\|E\|$	#solved	cputime	#nodes	#solved	cputime	#nodes	#solved	cputime	#nodes
pdh	11	34	39	5	30	40	0	33	40	1	45
newyork	16	49	40	19	58	38	61	59	40	0	98
ta1	24	55	40	154	74	39	331	73	40	0	70
france	25	45	31	13	63	30	38	64	40	0	54
norway	27	51	38	189	109	39	34	114	40	0	84
cost266	37	57	38	15	183	37	423	217	40	7	203
germany50	50	88	31	411	498	30	239	662	40	172	575
ta2	65	108	39	558	525	39	38	510	40	0	413

Table 6.10: Computational results on the Hose SNDLib instances.

ipsep-calls column, we see the proportion of the number of Hose MIP cut-set separations to the total number of cut-set separations (i.e., if k calls to the cut-set separation were made and i out of these calls required to solve a MIP, then the column depicts the value $(i/k) \cdot 100$). The proportion of MIP calls decreases with increasing instance size; on instances with up to 70 nodes, roughly every 10th to 7th call requires the algorithm to solve a MIP. The table also reveals that the algorithm closes less of the lp-gap the larger the instances grow. On all instances, the running time consumed by the heuristics is negligible. All these observations are independent of the interval distribution.

In total, the MIP cut-set separation is clearly less efficient than the polynomial cut-set separation from the previous section. This had to be expected. However, even though the MIP separation solves an $\mathcal{NP}$-hard problem *and* is called at least once per Branch-and-Bound node, it still allows us to solve a large part of the instances. Here, clearly the combination with the polynomial time separation algorithm pays off. In light of these observations, the running times of the algorithm are very reasonable. The increased running time of the separation algorithm also seems to be the reason why memory is no longer a problem: with more time spent in the separation, the Branch-and-Bound nodes are processed more slowly and thus do not fill the memory as quickly. We conjecture that the ip-sep-calls decrease on the larger instances because only those instances that require few calls can be solved.

SNDLib instances.

We close this section with a brief experiment which shows that our algorithm is useful on real-world instances as well. To this aim, we try to solve the SNDLib instances within four hours time and 3GB of memory. Table 6.10 shows the results of the experiments. We see from the table that apart from germany50 and france, most of the SNDLib instances can be solved reliably in less than 10 minutes and with less than 700 Branch-and-Bound nodes. Even on germany50 and france, we are able to solve at least 75% of the instances. These two instances are problematic in both the uniform and the geometric case, while the zero-one case seems to be generally easier.

We conclude that also on the real world instances of the SNDLib, our algorithm is able to quickly and reliably solve large parts of the instance set. As before, the zero-one case seems to be easier than the other cases, although it does not necessarily use fewer Branch-and-Bound nodes.

Conclusion

The experiments confirm that the Branch-and-Cut algorithm can solve realistic instances of the `sRND-H` problem reliably and in reasonable time. In particular, the MIP cut-set separation is practical, even though a significant number of calls to the separation routine are made. As before, the difficulty of an instance depends on its size and on its density. Additionally, we can observe a dependency on the distribution of the Hose bounds: Independently of the instance class, the `uniform` distribution of the bounds seems to be hardest for our algorithm. Instances with a `geometric` Hose bound distribution are slightly easier. The `zero-one` Hose bound distribution seems to be easy to solve. Our 3-partition inequalities continue to be effective.

Conclusion

In this thesis, we have considered the task to design single-commodity networks that work in different traffic scenarios. The scenario set for the problem is a polytope that can be given in an explicit vertex based description or in an implicit description by linear inequalities.

Our new cut-set IP formulation is based only on capacity variables and works in both cases. This is possible because the formulation does not need flow variables for every vertex of the scenario set. The new formulation allows us to find 3-partition inequalities as new problem specific cutting planes and to develop a Branch-and-Cut algorithm. Our experiments show that the algorithm is practical, that it improves on other approaches and that the new cutting planes are essential for its success. This is despite the fact that for scenario sets in a linear description, an $\mathcal{NP}$-hard separation problem must be solved.

This result is interesting in the context of multi-commodity network design. Here, capacity formulations exist as well and in principle, they allow us to cope with scenario sets that are given in an implicit linear description. However, these formulations require solving the separation problem for metric inequalities. With the current state-of-the-art, this is only possible by solving a non-convex quadratic problem [Mat13]. So far, this quadratic problem can only be avoided by switching to the static routing model that produces more conservative solutions. Our algorithm provides another alternative: If the application allows for it, we can switch to a single-commodity flow model where the separation problem can be solved by a simple MIP. However, it is not likely that this simple MIP separation can be translated to the multi-commodity case: The key result that allowed us to develop this separation procedure is a simplified description of the right-hand side of a cut-set inequality. This simplified right-hand side turns out to be a closed-form formula for a static maximum flow problem on a path. In the multi-commodity case, this flow problem must be solved on a bipartite graph, making it much more difficult to find a closed-form solution.

Even though we were able to show that both cut-set inequalities and 3-partition inequalities induce facets of the `sRND` polyhedron, we have not yet fully understood its structure. In particular, it is an open question if the 3-partition inequalities are exactly the rank-1 $\{0, \frac{1}{2}\}$-cuts of the cut-set formulation. It is also not clear if higher rank $\{0, \frac{1}{2}\}$-cuts have a similar combinatorial interpretation: We conjecture that $\{0, \frac{1}{2}\}$-cuts of rank k correspond to $(k+2)$-partition inequalities.

In the multi-commodity case, all valid inequalities for the capacity formulation are tight metric inequalities. The analogous inequality for the single-commodity robust network design problem on an instance $(V, E, \mathfrak{B})$ is given by

$$\sum_{\{i,j\} \in E} \mu_{ij} u_{ij} \geq \max_{b \in \mathfrak{B}} \mathrm{MCF}(V, E, b, \mu)$$

where $\mathrm{MCF}(V, E, b, \mu)$ is the value of an uncapacitated minimum cost b-flow with respect to costs defined by a metric μ. This inequality is valid, but it is not clear if it is also facet-inducing. We do not know a separation algorithm for this inequality class.

The Branch-and-Cut algorithm also invites future research questions. First, we conjecture that symmetry breaking branching rules have a great potential for a speedup of the algorithm. We observe that the difficult hypercube instances from Chapter 3 have a large number of symmetric solutions that are induced by disjoint paths between terminals. Exploiting symmetries should lead to a better branching rule. Second, the performance of the cut-set separation in the linear description case could be enhanced by further heuristic separation procedures. This should be done such that, ideally, the MIP separation would be called only at the end of each Branch-and-Bound node. Third, in this case, the MIP models a special Max-Cut problem. Using more sophisticated solution algorithms from the Max-Cut literature would probably speed up the separation.

Bibliography

[AABP07] A. Altın, E. Amaldi, P. Belotti and M. Ç. Pınar, *Provisioning virtual private networks under traffic uncertainty*, *Networks*, vol. 49, no. 1, pp. 100–115, 2007.

[ACDJ+12] E. Álvarez-Miranda, V. Cacchiani, T. Dorneth, M. Jünger, F. Liers, A. Lodi, T. Parriani and D. R. Schmidt, *Models and Algorithms for Robust Network Design with Several Traffic Scenarios*, in *ISCO 2012, Revised Selected Papers*, A. Ridha Mahjoub, V. Markakis, I. Milis and V. T. Paschos, Eds., ser. Lecture Notes in Computer Science, vol. 7422, Springer, 2012, pp. 261–272.

[ACF07] G. Andreello, A. Caprara and M. Fischetti, *Embedding $\{0, \frac{1}{2}\}$-Cuts in a Branch-and-Cut Framework: A Computational Study*, *INFORMS Journal on Computing*, vol. 19, pp. 229–238, 2007.

[Aga06] Y. K. Agarwal, *k-Partition-based facets of the network design problem*, *Networks*, vol. 47, no. 3, pp. 123–139, 2006.

[AMO93] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[AMS04] P. Avella, S. Mattia and A. Sassano, *Metric Inequalities and the Network Loading Problem*, in *Integer Programming and Combinatorial Optimization*, D. Bienstock and G. Nemhauser, Eds., ser. Lecture Notes in Computer Science, vol. 3064, Springer, 2004, pp. 401–421.

[Ata02] A. Atamtürk, *On Capacitated Network Design Cut-Set Polyhedra*, *Mathematical Programming Series B*, vol. 92, no. 3, pp. 425–437, 2002.

[AYP11] A. Altın, H. Yaman and M. Ç. Pınar, *The Robust Network Loading Problem Under Hose Demand Uncertainty: Formulation, Polyhedral Analysis, and Computations*, *INFORMS Journal on Computing*, vol. 23, no. 1, pp. 75–89, 2011.

[BA99] A.-L. Barabási and R. Albert, *Emergence of Scaling in Random Networks*, *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[BAK05] W. Ben-Ameur and H. Kerivin, *Routing of Uncertain Traffic Demands*, *Optimization and Engineering*, vol. 6, pp. 283–313, 3 2005.

[Bar96] F. Barahona, *Network Design Using Cut Inequalities*, *SIAM Journal on Optimization*, vol. 6, no. 3, pp. 823–837, 1996.

[BCGT98] D. Bienstock, S. Chopra, O. Günlük and C.-Y. Tsai, *Minimum cost capacity installation for multicommodity network flows*, *Mathematical Programming*, vol. 81, no. 2, pp. 177–199, 1998.

[Ben62] J. F. Benders, *Partitioning procedures for solving mixed-variables programming problems*, *Numerische Mathematik*, vol. 4, no. 1, pp. 238–252, 1962.

[BG96] D. Bienstock and O. Günlük, *Capacitated Network Design – Polyhedral Structure and Computation*, *INFORMS Journal on Computing*, vol. 8, no. 3, pp. 243–259, 1996.

[BLO08] C. Buchheim, F. Liers and M. Oswald, *Local cuts revisited*, *Operations Research Letters*, vol. 36, no. 4, pp. 430–433, 2008.

[BLS11] C. Buchheim, F. Liers and L. Sanità, *An Exact Algorithm for Robust Network Design*, in *Proceedings of the INOC*, J. Pahl, T. Reiners and S. Voß, Eds., ser. INOC'11, Springer, 2011, pp. 7–17.

[BS03] D. Bertsimas and M. Sim, *Robust discrete optimization and network flows*, *Mathematical Programming Series B*, vol. 98, pp. 49–71, 2003.

[BS04] ——, *The Price of Robustness*, *Operations Research*, vol. 52, no. 1, pp. 35–53, 2004.

[BTEN09] A. Ben-Tal, L. El Ghaoui and A. S. Nemirovski, *Robust Optimization*, ser. Princeton Series in Applied Mathematics. Princeton University Press, Oct. 2009.

[BTN99] A. Ben-Tal and A. Nemirovski, *Robust solutions of uncertain linear programs*, *Operations Research Letters*, vol. 25, no. 1, pp. 1–13, 1999.

[CCG09] A. M. Costa, J.-F. Cordeau and B. Gendron, *Benders, metric and cutset inequalities for multicommodity capacitated network design*, *Computational Optimization and Applications*, vol. 42, no. 3, pp. 371–392, 2009.

[CF96] A. Caprara and M. Fischetti, $\{0, \frac{1}{2}\}$-*Chvátal-Gomory cuts*, *Mathematical Programming*, vol. 74, no. 3, pp. 221–235, 1996.

[CG95] B. V. Cherkassky and A. V. Goldberg, *On implementing push-relabel method for the maximum flow problem*, in *Proceedings of IPCO*, E. Balas and J. Clausen, Eds., ser. Lecture Notes in Computer Science, vol. 920, Springer, 1995, pp. 157–171.

[Chi60] R. T. Chien, *Synthesis of a Communication Net*, *IBM Journal of Research and Development*, vol. 4, no. 3, pp. 311–320, 1960.

[Chv73] V. Chvátal, *Edmonds polytopes and a hierarchy of combinatorial problems*, *Discrete Mathematics*, vol. 4, no. 4, pp. 305–337, 1973.

[CJL+14] V. Cacchiani, M. Jünger, F. Liers, A. Lodi and D. R. Schmidt, *Single-Commodity Robust Network Design with Finite and Hose Demand Sets*, Universität zu Köln, Technical Report, 2014, Also appeared as technical report OR-14-11 at the University of Bologna, Italy.

[CL08] T. Christof and A. Löbel, *PORTA — POlyhedron Representation Transformation Algorithm*, `http://typo.zib.de/opt-long_projects/Software/Porta/`, 2008.

[CR94] S. Chopra and M. R. Rao, *The Steiner tree problem I: Formulations, compositions and extension of facets*, *Mathematical Programming*, vol. 64, pp. 209–229, 1 1994.

[CSOS07] C. Chekuri, B. F. Shepherd, G. Oriolo and M. Scutellà, *Hardness of robust network design*, *Networks*, vol. 50, no. 1, pp. 50–54, 2007.

[Dan51] G. B. Dantzig, *Maximization of a linear function of variables subject to linear inequalities*, in *Activity Analysis of Production and Allocation*, T. C. Koopmans, Ed., Wiley, 1951, pp. 339–347.

[DGG+99] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan and J. E. van der Merwe, *A flexible model for resource management in virtual private networks*, in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, ser. SIGCOMM '99, ACM, 1999, pp. 95–108.

[Dij59] E. W. Dijkstra, *A note on two problems in connexion with graphs*, *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[Dor12] T. Dorneth, *Ein Branch-and-Cut-Verfahren für robustes Netzwerkdesign*, Diplomarbeit, Universität zu Köln, Nov. 2012.

[EGJR01] M. Elf, C. Gutwenger, M. Jünger and G. Rinaldi, *Branch-and-Cut Algorithms for Combinatorial Optimization and Their Implementation in ABACUS*, in *Computational Combinatorial Optimization*, ser. Lecture Notes in Computer Science, M. Jünger and D. Naddef, Eds., vol. 2241, Springer, 2001, pp. 157–222.

[EIS76] S. Even, A. Itai and A. Shamir, *On the Complexity of Timetable and Multicommodity Flow Problems*, *SIAM Journal on Computing*, vol. 5, no. 4, pp. 691–703, 1976.

[Eis99] F. Eisenbrand, *On the membership problem for the elementary closure of a polyhedron*, *Combinatorica*, vol. 19, no. 2, pp. 297–300, 1999.

[ER04] T. Erlebach and M. Rüegg, *Optimal Bandwidth Reservation in Hose-Model VPNs with Multi-Path Routing*, *Proceedings of the INFOCOM*, vol. 4, pp. 2275–2282, 2004.

[FF54] L. R. Ford Jr. and D. R. Fulkerson, *Maximal flow through a network*, The RAND Corporation, Santa Monica, California, Research Memorandum RM-1400, 1954, published in [FF56].

[FF56] ——, *Maximal Flow through a Network*, *Canadian J. of Mathematics*, vol. 8, pp. 399–404, 1956.

[FF58] ——, *A Suggested Computation for Maximal Multi-Commodity Network Flows*, *Management Science*, vol. 5, no. 1, pp. 97–101, 1958.

[FL07] M. Fischetti and A. Lodi, *Optimizing over the first chvátal closure*, *Mathematical Programming B*, vol. 110, no. 1, pp. 3–20, 2007.

[FST97] J. A. Fingerhut, S. Suri and J. S. Turner, *Designing Least-Cost Nonblocking Broadband Networks*, *Journal of Algorithms*, vol. 24, no. 2, pp. 287–309, 1997.

[Gal57] D. Gale, *A theorem on flows in networks. Pacific Journal of Mathematics*, vol. 7, no. 2, pp. 1073–1082, 1957.

[GH61] R. E. Gomory and T. C. Hu, *Multi-terminal Network Flow*, *SIAM Journal on Applied Mathematics*, vol. 9, pp. 551–570, 1961.

[GH62] R. Gomory and T. Hu, *An Application of Generalized Linear Programming to Network Flows*, *Journal of the Society for Industrial and Applied Mathematics*, vol. 10, no. 2, pp. 260–283, 1962.

[GH64] R. E. Gomory and T. C. Hu, *Synthesis of a Communication Network*, *Journal of the Society for Industrial and Applied Mathematics*, vol. 12, no. 2, pp. 348–369, 1964.

[GJ00] E. Gawrilow and M. Joswig, *polymake: a Framework for Analyzing Convex Polytopes*, in *Polytopes – Combinatorics and Computation*, G. Kalai and G. M. Ziegler, Eds., Birkhäuser, 2000, pp. 43–74.

[GKK+01] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi and B. Yener, *Provisioning a virtual private network: a network design problem for multicommodity flow*, in *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, ser. STOC '01, ACM, 2001, pp. 389–398.

[GLS81] M. Grötschel, L. Lovász and A. Schrijver, *The ellipsoid method and its consequences in combinatorial optimization*, *Combinatorica*, vol. 1, no. 2, pp. 169–197, 1981, **corrigendum:** [GLS84].

[GLS84] ——, *Corrigendum to our paper "The ellipsoid method and its consequences in combinatorial optimization"*, *Combinatorica*, vol. 4, no. 4, pp. 291–295, 1984.

[Gom58] R. E. Gomory, *Outline of an algorithm for integer solutions to linear programs*, *Bulletin of the American Mathematical Society*, vol. 64, no. 5, pp. 275–278, Sep. 1958.

[GOS08] N. Goyal, N. Olver and B. Shepherd, *The VPN conjecture is true*, *Proceedings of the STOC*, pp. 443–450, 2008.

[Grö04] M. Grötschel, *Lineare Optimierung*, `http://www.zib.de/groetschel/teaching/skriptADMII.pdf`, Lecture script in the version of February 27th, 2004. In german., 2004.

[GT88] A. V. Goldberg and R. E. Tarjan, *A New Approach to the Maximum-flow Problem*, *Journal of the ACM*, vol. 35, no. 4, pp. 921–940, 1988.

[Gus90] D. Gusfield, *Very Simple Methods for All Pairs Network Flow Analysis*, *SIAM Journal on Computing*, vol. 19, no. 1, pp. 143–155, 1990.

[Gün02] O. Günlük, *A New Min-Cut Max-Flow Ratio for Multicommodity Flows*, in *Proceedings of IPCO*, ser. Lecture Notes in Computer Science, W. J. Cook and A. S. Schulz, Eds., vol. 2337, Springer, 2002, pp. 54–66.

[Gün07] ——, *A New Min-Cut Max-Flow Ratio for Multicommodity Flows*, *SIAM Journal on Discrete Mathematics*, vol. 21, no. 1, pp. 1–15, 2007.

[Gün99] ——, *A branch-and-cut algorithm for capacitated network design problems*, *Mathematical Programming*, vol. 86, pp. 17–39, 1 1999.

[HHW88] F. Harary, J. P. Hayes and H.-J. Wu, *A survey of the theory of hypercube graphs*, *Computers & Mathematics with Applications*, vol. 15, no. 4, pp. 277–289, 1988.

[Hoc97] D. S. Hochbaum, Ed., *Approximation Algorithms for NP-hard Problems*. PWS Publishing, 1997.

[Iri70] M. Iri, *On an Extension of the Maximum-Flow Minimum-Cut Theorem to Multicommodity Flows*, *Journal of the Operations Research Society of Japan*, vol. 13, no. 3, 1970.

[Jai01] K. Jain, *A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem*, *Combinatorica*, vol. 21, no. 1, pp. 39–60, 2001.

[JMP00] D. S. Johnson, M. Minkoff and S. Phillips, *The Prize Collecting Steiner Tree Problem: Theory and Practice*, in *Proceedings of the SODA*, ser. SODA '00, SIAM, 2000, pp. 760–769.

[JT00] M. Jünger and S. Thienel, *The ABACUS System for Branch-and-Cut-and-Price Algorithms in Integer Programming and Combinatorial Optimization*, *Software: Practice and Experience*, vol. 30, no. 11, pp. 1325–1352, 2000.

[Kar72] R. M. Karp, *Reducibility Among Combinatorial Problems*, in *Complexity of Computer Computations*, 1972, pp. 85–103.

[Kar75] ——, *On the complexity of combinatorial problems*, *Networks*, no. 5, pp. 45–68, 1975.

[KKR13] A. M. C. A. Koster, M. Kutschka and C. Raack, *Robust network design: Formulations, valid inequalities, and computations*, *Networks*, vol. 61, no. 2, pp. 128–149, 2013.

[KOR+09] A. M. C. A. Koster, S. Orlowski, C. Raack, G. Baier, T. Engel and P. Belotti, *Branch-and-cut techniques for solving realistic two-layer network design problems*, in *Graphs and Algorithms in Communication Networks*, Springer, 2009, pp. 95–118.

[KP80] R. M. Karp and C. Papadimitriou, *On linear characterizations of combinatorial optimization problems*, in *Foundations of Computer Science, 1980., 21st Annual Symposium on*, Oct. 1980, pp. 1–9.

[KP82] ——, *On Linear Characterizations of Combinatorial Optimization Problems*, *SIAM Journal on Computing*, vol. 11, no. 4, pp. 620–632, 1982, first appeared as [KP80].

[KV12] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 5th, ser. Algorithms and Combinatorics. Springer, 2012, vol. 21 2012.

[KYDN09] S. N. Kabadi, J. Yan, D. Du and K. P. K. Nair, *Integer exact network synthesis problem*, *SIAM Journal on Discrete Mathematics*, vol. 23, no. 1, pp. 136–154, 2009.

[LLP13] C. Lee, K. Lee and S. Park, *Benders decomposition approach for the robust network design problem with flow bifurcations*, *Networks*, vol. 62, no. 1, pp. 1–16, 2013.

[LPSG12] I. Ljubić, P. Putz and J.-J. Salazar-González, *Exact approaches to the single-source network loading problem*, *Networks*, vol. 59, no. 1, pp. 89–106, 2012.

[LR99] T. Leighton and S. Rao, *Multicommodity Max-flow Min-cut Theorems and Their Use in Designing Approximation Algorithms*, *Journal of the ACM*, vol. 46, no. 6, pp. 787–832, 1999.

[LSSW99] M. Labbé, R. Séguin, P. Soriano and C. Wynants, *Network Synthesis with Non-Simultaneous Multicommodity Flow Requirements: Bounds and Heuristics*, 1999.

[Mac87a] N. Maculan, *The Steiner problem in graphs*, *Annals of Discrete Mathematics*, vol. 21, 185—212, 1987, also appeared as [Mac87b].

[Mac87b] ——, *The Steiner Problem in Graphs*, in *Surveys in Combinatorial Optimization*, ser. North-Holland Mathematics Studies, S. Martello, G. Laporte, M. Minoux and C. Ribeiro, Eds., vol. 132, North-Holland, 1987, pp. 185–211.

[Mar01] A. Martin, *General mixed integer programming: computational issues for branch-and-cut algorithms*, in *Computational Combinatorial Optimization*, ser. Lecture Notes in Computer Science, M. Jünger and D. Naddef, Eds., vol. 2241, Springer, 2001, pp. 1–25, ISBN: 978-3-540-42877-0.

[Mat10a] S. Mattia, *The Robust Network Loading Problem with Dynamic Routing*, Università di Roma la Sapienza, Tech. Rep. 3, 2010.

[Mat10b] ——, *The Robust Network Loading Problem with Dynamic Routing*, `http://www.optimization-online.org/DB_FILE/2010/11/2826.pdf`, 2010.

[Mat12] ——, *Separating tight metric inequalities by bilevel programming*, *Operations Research Letters*, vol. 40, no. 6, pp. 568–572, 2012.

[Mat13] ——, *The robust network loading problem with dynamic routing*, *Computational Optimization and Applications*, vol. 54, pp. 619–643, 2013.

[MCL+14] E. Álvarez Miranda, V. Cacchiani, A. Lodi, T. Parriani and D. R. Schmidt, *Single-commodity robust network design problem: Complexity, instances and heuristic solutions*, *European Journal of Operational Research*, vol. 238, no. 3, pp. 711–723, 2014.

[Men27] K. Menger, *Zur allgemeinen Kurventheorie*, *Fundamenta Mathematicae*, vol. 10, no. 1, pp. 96–115, 1927.

[Min81] M. Minoux, *Optimum Synthesis of a Network with Non-Simultaneous Multicommodity Flow Requirements*, in *Annals of Discrete Mathematics (11) Studies on Graphs and Discrete Programming*, vol. 59, North-Holland, 1981, pp. 269–277.

[Min89] ——, *Networks synthesis and optimum network design problems: Models, solution methods and applications*, *Networks*, vol. 19, no. 3, pp. 313–360, 1989.

[Mir00] P. Mirchandani, *Projections of the capacitated network loading problem*, *European Journal of Operational Research*, vol. 122, no. 3, pp. 534–560, 2000.

[MM93] T. L. Magnanti and P. Mirchandani, *Shortest paths, single origin-destination network design, and associated polyhedra*, *Networks*, vol. 23, no. 2, pp. 103–121, 1993.

[MMV91] T. L. Magnanti, P. Mirchandani and R. Vachani, *Modeling and Solving the Capacitated Network Loading Problem*, MIT, Technical Report OR-239-91, 1991.

[MMV93] ——, *The convex hull of two core capacitated network design problems*, *Mathematical Programming*, vol. 60, no. 1–3, pp. 233–250, 1993.

[MMV95] ——, *Modeling and Solving the Two-Facility Capacitated Network Loading Problem*, *Operations Research*, vol. 43, no. 1, pp. 142–157, 1995.

[MOL08] S. Mudchanatongsuk, F. Ordóñez and J. Liu, *Robust solutions for network design under transportation cost and demand uncertainty*, *Journal of the Operational Research Society*, vol. 59, no. 5, pp. 652–662, 2008.

[MRR03] S. T. McCormick, M. R. Rao and G. Rinaldi, *Easy and difficult objective functions for max-cut*, *Mathematical Programming B*, vol. 94, pp. 459–466, 2003.

[MW81] T. L. Magnanti and R. T. Wong, *Accelerating Benders Decomposition: Algorithmic Enhancement and Model Selection Criteria*, *Operations Research*, vol. 29, no. 3, pp. 464–484, 1981.

[MW84] ——, *Network Design and Transportation Planning: Models and Algorithms. Transportation Science*, vol. 18, no. 1, pp. 1–55, 1984.

[netlib] *Netlib*, http://www.netlib.org/lp/.

[NW88] G. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. Wiley, 1988.

[OGDF] *The Open Graph Drawing Framework*, http://www.ogdf.net.

[OK71] K. Onaga and O. Kakusho, *On feasibility conditions of multicommodity flows in networks*, *IEEE Transactions on Circuit Theory*, vol. 18, no. 4, pp. 425–429, 1971.

[OPTW07] S. Orlowski, M. Pióro, A. Tomaszewski and R. Wessäly, *SNDlib 1.0–Survivable Network Design Library*, in *Proceedings of the INOC 2007*, http://sndlib.zib.de, extended version accepted in Networks, 2009., 2007.

[Orl13] J. B. Orlin, *Max flows in o(nm) time, or better*, in *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '13, Palo Alto, California, USA: ACM, 2013, pp. 765–774.

[OS81] H. Okamura and P. D. Seymour, *Multicommodity flows in planar graphs*, *Journal of Combinatorial Theory, Series B*, vol. 31, no. 1, pp. 75–81, 1981.

[OSZ13] G. Oriolo, L. Sanità and R. Zenklusen, *Network design with a discrete set of traffic matrices*, *Operations Research Letters*, vol. 41, no. 4, pp. 390–396, 2013.

[OW03] F. Ortega and L. A. Wolsey, *A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem*, *Networks*, vol. 41, no. 3, pp. 143–158, 2003.

[PR11] M. Poss and C. Raack, *Affine Recourse for the Robust Network Design Problem: Between Static and Dynamic Routing*, in *Network Optimization*, ser. Lecture Notes in Computer Science, J. Pahl, T. Reiners and S. Voß, Eds., vol. 6701, Springer, 2011, pp. 150–155.

[PR81] M. W. Padberg and M. R. Rao, *The Russian method for linear inequalities III: bounded integer programming*, INRIA, Tech. Rep. RR-0078, May 1981, also appeared as Research Report 81-39, New York University, 1981.

[PRU04] R. Pesenti, F. Rinaldi and W. Ukovich, *An exact algorithm for the min-cost network containment problem*, *Networks*, vol. 43, no. 2, pp. 87–102, 2004.

[RKOW11] C. Raack, A. M. C. A. Koster, S. Orlowski and R. Wessäly, *On cut-based inequalities for capacitated network design polyhedra*, *Networks*, vol. 57, no. 2, pp. 141–156, 2011.

[San09] L. Sanità, *Robust Network Design*, PhD thesis, Università La Sapienza, Roma, 2009.

[San13] ——, Private communication, 2013.

[Sch86] A. Schrijver, *Theory of Linear and Integer Programming*. Wiley, 1986.

[Soy73] A. L. Soyster, *Convex Programming with Set-Inclusive Constraints and Applications to Inexact Linear Programming*, *Operations Research*, vol. 21, no. 5, pp. 1154–1157, 1973.

[SS85] Y. Saad and M. H. Schultz, *Topological properties of hypercubes*, Yale University, Tech. Rep. YALEU/DCS/TR389, 1985, published in [SS88].

[SS88] ——, *Topological properties of hypercubes*, *IEEE Transactions on Computers*, vol. 37, no. 7, pp. 867–872, Jul. 1988, first appeared as [SS85].

Erklärung

Ich versichere, dass ich die von mir vorgelegte Dissertation selbständig angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe; dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie – abgesehen von unten angegebenen Teilpublikationen – noch nicht veröffentlicht worden ist sowie, dass ich eine solche Veröffentlichung vor Abschluss des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen der Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von Prof. Dr. Michael Jünger betreut worden.

Köln, 29. September 2014

D. Schmidt

Teilpublikationen

- E. Álvarez-Miranda, V. Cacchiani, T. Dorneth, M. Jünger, F. Liers, A. Lodi, T. Parriani and D. R. Schmidt, *Models and Algorithms for Robust Network Design with Several Traffic Scenarios*, in *ISCO 2012, Revised Selected Papers*, A. Ridha Mahjoub, V. Markakis, I. Milis and V. T. Paschos, Eds., ser. Lecture Notes in Computer Science, vol. 7422, Springer, 2012, pp. 261–272
- E. Álvarez Miranda, V. Cacchiani, A. Lodi, T. Parriani and D. R. Schmidt, *Single-commodity robust network design problem: Complexity, instances and heuristic solutions*, *European Journal of Operational Research*, vol. 238, no. 3, pp. 711–723, 2014
- V. Cacchiani, M. Jünger, F. Liers, A. Lodi and D. R. Schmidt, *Single-Commodity Robust Network Design with Finite and Hose Demand Sets*, Universität zu Köln, Technical Report, 2014, Also appeared as technical report OR-14-11 at the University of Bologna, Italy.