Solving Mixed-Integer Linear and Nonlinear Network Optimization Problems by Local Reformulations and Relaxations

Lösungsmethoden für gemischt-ganzzahlige lineare und nichtlineare Netzwerkoptimierungsprobleme basierend auf lokalen Reformulierungen und Relaxierungen

Der Naturwissenschaftlichen Fakultät

der Friedrich-Alexander-Universität Erlangen-Nürnberg

zur

Erlangung des Doktorgrades Dr. rer. nat.

vorgelegt von

Maximilian Merkert

aus Kaiserslautern

Als Dissertation genehmigt
von der Naturwissenschaftlichen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der mündlichen Prüfung:	02.11.2017
Vorsitzender des Promotionsorgans:	Prof. Dr. Georg Kreimer
Gutachterin:	Prof. Dr. Frauke Liers
Gutachter:	Prof. Dr. Rüdiger Schultz
Gutachter:	Prof. Dr. Christoph Helmberg

Acknowledgements

First of all, I would like to thank my supervisors Frauke Liers and Alexander Martin. They enabled me to work in a fruitful environment on a number of interesting projects with various academic and industrial partners. I also wish to express my deepest gratitude for their guidance not only on mathematical issues but also on more general topics regarding life in academia.

I am also grateful to Johannes Jahn, Rüdiger Schultz and Christoph Helmberg for agreeing to be involved in the examination process.

During my time in Erlangen, I have experienced that mathematical research is team work to a great extend. Therefore, I wish to thank my coauthors Andreas Bärmann, Thorsten Gellermann, Frauke Liers, Alexander Martin, Nick Mertens, Dennis Michaels, Oskar Schneider, Christoph Thurner, Robert Weismantel and Dieter Weninger for the pleasant and productive collaborations. Moreover, many thanks to all my colleagues at FAU Erlangen-Nürnberg and from the Collaborative Research Center TRR 154 for many valuable discussions as well as the supportive atmosphere and many social events. It was a pleasure to work with you!

Furthermore, I want to thank Nina Gunkelmann, Andreas Bärmann, Lena Hupp, Nick Mertens, Dennis Michaels and Dieter Weninger for proof-reading parts of this thesis and helpful remarks.

I gratefully acknowledge the computing resources provided by the group of Michael Jünger and the technical support by Thomas Lange in Cologne as well as by Denis Aßmann and Thorsten Gellermann in Erlangen. Further thanks go to Christina Weber, Beate Kirchner and Gabriele Bittner for administrative aid.

Last but not least, my special thanks go to my parents for all their support ever since I can remember, and to my partner for her great continual believe in me.

Abstract

Since the beginnings of network optimization, the number of use cases has grown enormously and can be expected to further expand in an increasingly interconnected world. The wide range of modern applications include optimization tasks on energy networks, telecommunication networks and in public transport, just to name a few. Although many traditional network optimization problems are NP-hard in their basic version, applications pose additional challenges due to more complicated—often nonlinear—dependencies or the sheer size of the network.

In this thesis, we develop methods that help to cope with those challenges. A common strategy will be to improve mathematical programming formulations locally by modeling substructures in an integrated way. The resulting reformulations and relaxations will allow for global methods that either solve the problem to exact optimality or up to a predefined precision.

For large-scale network expansion problems, a solution method is proposed that is based on iterative aggregation. Starting with an initial aggregation, we solve a sequence of network design problems over increasingly fine-grained representations of the original network. This is done until the whole network is represented sufficiently well in the sense that an optimal solution to the aggregated problem can easily be extended to an optimal solution of the original problem. Global optimality is guaranteed by a subproblem that computationally is less expensive and either proves optimality or gives an indication of where to refine the representation. In this algorithmic scheme, locally relaxing the problem allows us to focus on the critical part of the network.

In many optimization problems on transportation networks—especially those arising from energy applications—the main challenge is connected to the problem's nonlinear features, arising, for example, from laws of physics. Gas networks represent a typical example for such a nonlinear network flow setting that we will repeatedly refer to throughout this work. A common and established solution approach consists of constructing a piecewise linear approximation or relaxation. We study how to strengthen the resulting mixed-integer programming formulation for specific substructures in the network. We find effective cutting planes and derive a complete description for induced paths of arbitrary length—using graph-theoretic arguments related to perfect graphs.

A generalization of key properties of this special case leads to an abstract definition in terms of clique problems on a specific type of graph. This abstract setting also comprises a basic version of the project scheduling problem and still allows us to give totally unimodular reformulations that are of linear size. Moreover, questions regarding recognizability of this structure will be discussed.

We also discuss the concept of simultaneous convexification that can be seen as a continuous counterpart to our approach for piecewise linearized problems. The resulting reformulations can improve relaxations employed by general-purpose MINLP solvers, which usually rely on convexifying nonlinear functions separately.

Computational results demonstrate the practical impact of the methods developed in this thesis, in many cases using real-world data sets.

Zusammenfassung

Seit den Anfängen der Netzwerkoptimierung ist die Zahl der Anwendungsfälle immens gewachsen und angesichts einer zunehmend vernetzten Welt ist ein weiterer Anstieg zu erwarten. Die Spannbreite moderner Anwendungen umfasst Optimierungsprobleme auf Energienetzen, Telekommunikationsnetzen und Verkehrsnetzen, um nur einige zu nennen. Auch wenn viele traditionelle Netzwerkoptimierungsprobleme bereits in ihrer Grundversion NP-schwer sind, stellen Anwendungen weitere Anforderungen aufgrund komplexerer - oftmals nichtlinearer - Abhängigkeiten oder der schieren Größe der zugrunde liegenden Netzwerke.

In dieser Arbeit werden Methoden entwickelt, um mit diesen Herausforderungen umzugehen. Die wesentliche Strategie wird darin bestehen, mathematische Problemformulierungen lokal zu verstärken, indem ausgewählte Substrukturen als Ganzes erfasst und modelliert werden. Die resultierenden Reformulierungen und Relaxierungen unterstützen globale Methoden, die entweder exakte oder bis auf eine vordefinierte Genauigkeit optimale Lösungen finden.

Für große Netzausbauprobleme wird eine Lösungsmethodik basierend auf iterativer Aggregation entworfen. Beginnend mit einer Startaggregation lösen wir eine Folge zunehmend detaillierter Vergröberungen des ursprünglichen Netzwerks, bis die Darstellung hinreichend genau ist, sodass seine Optimallösung leicht auf das ursprüngliche Problem übertragen werden kann. Exaktheit wird dabei durch ein Subproblem sichergestellt, das entweder Optimalität bestätigt oder Ansatzpunkte zur Verfeinerung der Darstellung liefert. In diesem Schema erlaubt somit eine lokale Relaxierung die Fokussierung auf kritische Teile des Netzwerks.

In vielen Optimierungsproblemen auf Transportnetzen, insbesondere für Energieträger, besteht die wesentliche Herausforderung in den auftretenden Nichtlinearitäten, die beispielsweise physikalischen Gesetzen geschuldet sind. Gasnetwerke sind hierfür ein typisches Beispiel, auf das wir uns mehrfach in dieser Arbeit beziehen werden. Ein etablierter Lösungsansatz besteht in der Konstruktion stückweise linearer Approximationen oder Relaxierungen. Es wird untersucht, wie die entstehende Formulierung für bestimmte Substrukturen verstärkt werden kann. Dabei finden wir effektive Schnittebenen und leiten mittels graphentheoretischer Argumente eine vollständige Beschreibung für induzierte Pfade her.

Eine Abstraktion wesentlicher Eigenschaften dieses Spezialfalls führt auf Cliquenprobleme auf bestimmten Graphen. Dieser abstrakte Rahmen umfasst auch eine Basisversion des Projektplanungsproblems und erlaubt es weiterhin, eine total unimodulare Reformulierung von linearer Größe nachzuweisen. Weiterhin werden Fragen zur Erkennbarkeit dieser Struktur behandelt.

Außerdem wird das Konzept der simultanen Konvexifizierung diskutiert, das als kontinuierliches Gegenstück zu unserem Ansatz für stückweise linearisierte Probleme angesehen werden kann. Die entstehenden Reformulierungen verstärken Relaxierungen, auf die allgemeine MINLP-Löser typischerweise angewiesen sind.

Rechenergebnisse unter Einbeziehung realer Datensätze zeigen den praktischen Einfluss der in dieser Arbeit entwickelten Methoden.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of network optimization covers a wide range of problems regularly faced in real-world applications and, consequently, has attracted a lot of attention among researchers. In particular, whenever an application involves routing some kind of quantity in a feasible or optimal way on a discrete structure, we may regard the problem as a network flow problem where 'flow' may refer to different types of physical (e.g. water, natural gas, industrial goods, passengers) as well as notional quantities (information). This makes this type of problem a powerful concept for unifying problems from different areas.

In many modern applications, the network optimization problems involved add a lot of intricacy to classical network optimization problems. The main challenges—in addition to the problem simply being NP-hard—are often due to huge infrastructures that we wish to look at in an integrated way, or nonlinear dependencies, e.g. due to modeling laws of physics. A direct approach for solving such a problem in practice consists of modeling it as a mixed-integer linear or nonlinear program and handing it to a general-purpose solver.

In this thesis, solution methods are developed that are based on strong reformulations and/or relaxations of standard formulations. They are tailored for specific situations that we try to keep as general as possible. However, the presentation of a method, especially when it comes to proof-of-computations, might focus on specific example applications or instances that can be considered typical and particularly well-suited. Possibilities for extensions to more general cases are sketched at various occasions.

A common theme of our methods will be to consider substructures of the underlying network in an integrated way. The formulation for the target substructure is locally improved, exploiting the structure of the network and/or other features of the problem type. Here, 'improving the formulation' may also result in a relaxation that simplifies the solver's task but still gives strong bounds—as long as global (approximate) optimality is ensured by other means within the overall algorithmic scheme. In contrast, a *reformulation* is usually expected to give a formulation that is equivalent to the original one—though we will not apply any rigorous definition to this term. However, reformulations can be beneficial for general-purpose solvers in a number of ways, some of which will be represented in this work: reformulations may streng-

then relaxations naturally employed by solvers within their branch-and-bound process. The most prominent example are cutting-plane methods that strengthen the problem's linear relaxation. For nonlinear programs, strong convex relaxations are of similar relevance. In other cases, the strength of reformulations may have emerged empirically, where explanations are related to sparsity or better chances for a more balanced branch-and-bound tree.

Many chapters contain computational experiments to show the potential of the proposed methods in practice. To construct test instances, we incorporate data from real-world applications. In particular, as we concentrate on network structure, a realistic network topology is important as, for instance, transportation networks tend to be relatively sparse. Test examples will be obtained from public transport and energy applications. In particular, we use optimization problems on transportation networks for natural gas as an example for mixed-integer nonlinear network flow problems multiple times throughout this work.

The Structure of this Thesis

This thesis is organized as follows: Chapter 2 introduces some basic theoretical background for the following chapters to build upon, concerning two major topics: on the one hand, Section 2.1 presents a selection of problems in network optimization—of course biased with respect to their importance for this work. In particular, basics of gas network optimization will be covered in Subsection 2.1.5. On the other hand, Section 2.2 covers modeling techniques for piecewise linear functions; those represent a strong tool for tackling nonlinear problems.

In Chapter 3, we present an exact approach for solving network design problems that is based on an iterative graph aggregation procedure. After an introductory discussion on aggregation and its role in optimization literature in Section 3.1, we briefly recall the network expansion problem in Section 3.2. In Section 3.3, we present a detailed description of the iterative aggregation scheme for network expansion problems, including details on the master problems and subproblems used in each iteration. In Subsection 3.3.4, we also relate our method to *Benders decomposition*. Then, in Section 3.4, we report some implementation details and describe three different iterative aggregation algorithms. In Section 3.5, we show computational results on single-commodity instances of the network expansion problem on random scale-free as well as realistic network topologies. Possible extensions to more complicated network expansion problems are discussed in Section 3.6 for mixed-integer linear problems and in Section 3.7 for gas networks.

In Chapter 4, we study polyhedra in the context of network flow problems, in which the flow value on each arc lies in one of several predefined intervals. This is motivated by nonlinear problems on transportation networks, where nonlinearities are handled by piecewise linear approximation or relaxation. After introducing the setting and our polytope of interest in Section 4.1, we study the geometric structure of that polytope when the problem is defined on simple network structures in Section 4.2.

Starting with two adjacent arcs, we move on to the cases of paths and stars. For the former, a complete description is derived in Subsection 4.2.2; our proof relies on a new class of perfect graphs. We also show how to obtain corresponding results for a formulation based on the *Incremental Method* in Subsection 4.2.3. Section 4.3 presents empirical studies on the performance of the derived cutting planes, showing that they lead to a significant improvement when used within a state-of-the-art MIP-solver. The chapter concludes with some further remarks in Section 4.4.

Chapter 5 deals with clique problems with multiple-choice constraints, which are introduced in Section 5.1. The definition of staircase compatibility in Section 5.2 generalizes common properties of two applications as we will also see in that section, one of which may arise from the setting in Subsection 4.2.2. Following that, in Section 5.3 two totally unimodular formulations for the clique problem with multiple-choice constraints under staircase compatibility are presented. In Section 5.4, we evaluate our reformulations from a computational point of view by applying them to two different real-world applications. These include energy-efficient railway timetabling in addition to piecewise linearized network flow problems on gas networks. Moreover, Section 5.5 addresses several questions on the recognizability of staircase relations.

Chapter 6 gives an outlook on potentials and possibilities for computing strong convex relaxations for optimization problems on gas networks in the spirit of Chapter 4, i.e. we strengthen the formulation by considering substructures of the network consisting of more than two nonlinear functions simultaneously. In Section 6.1, we give some theoretical background on simultaneous convexification, i.e. computing the convex hull of vector-valued functions. After that, we focus on optimization problems on gas networks in Section 6.2, in particular to the simultaneous convex hull of functions related to a junction in the network. In Section 6.3, the potential of our approach is discussed on the basis of computational experiments on a small test network. An outlook is given in Section 6.4.

Finally, Chapter 7 summarizes the results, highlights some key observations and mentions some open questions for further investigation.

Incorporation of Collaborative Work

Parts of this thesis are based on joint work with other authors that has been published elsewhere.

Chapter 3 is based on the article *Solving Network Design Problems via Iterative Aggregation*, published in *Mathematical Programming Computations* [BLM$^+$15]. It is joint work with Andreas Bärmann, Frauke Liers, Alexander Martin, Christoph Thurner and Dieter Weninger. It was a team effort from the very beginning and the resulting methods emerged from many fruitful discussions we had on a day-to-day basis. I contributed significantly to all aspects of this work. Additionally, in this thesis extensions of the algorithms from [BLM$^+$15] (other than a direct multi-commodity version) to more general cases of network expansion problems are discussed in Sections 3.6 and Sections 3.7, and a new version of the algorithms for the single commodity network

expansion problem has been enabled by Theorem 3.4.

Chapter 4 is based on the article *Structural Investigation of Piecewise Linearized Network Flow Problems*, which is joint work together with my supervisor Frauke Liers, and has been published in the *SIAM Journal on Optimization* [LM16]. I had the freedom to work independently on that topic to a large extend, complemented by regular joint discussions.

Major parts of Chapter 5 are based on the paper *Staircase Compatibility and its Applications in Scheduling and Piecewise Linearization*, which is joint work with Andreas Bärmann, Thorsten Gellermann and Oskar Schneider. It has been published as a technical report [BGMS16] and a corresponding journal article has been submitted. This project emerged when Andreas and I together realized striking parallels between piecewise linearized flow problems and problems in energy-efficient timetabling he had been working on, and we eventually came up with the definition of staircase compatibility. Many theoretical results were worked out in joint sessions between all authors. Oskar, who did his master thesis in this context, developed the implementation for the computations in Subsection 5.4.1. In the present work, I extended the chapter by Section 5.5, which goes beyond [BGMS16]and addresses recognizability issues.

Finally, Chapter 6 is part of unpublished work within a joint project in progress together with Frauke Liers, Alexander Martin, Nick Mertens, Dennis Michaels and Robert Weismantel. The computational experiments in that chapter have been performed by myself.

A brief note on collaborative work will also be given in the introduction of the respective chapters.

Chapter 2

Preliminaries

In this chapter, we will introduce some basic notions and concepts we will work with throughout the subsequent chapters. On the one hand, we will discuss some typical problems on transportation networks together with suitable mathematical programming models; on this occasion, we will also introduce notations and conventions related to networks that will be used throughout this work. On the other hand, as several chapters deal with piecewise linearization (or feature related models), the concepts of piecewise linear approximation—including several important formulations for modeling a piecewise linear function—will be covered as well. Readers who have a strong background in those topics may skip this chapter for now (at their own risk) and only come back on demand. Notes referring back to this chapter will be found in a number of passages throughout this work.

In any case, it is assumed that the reader has basic knowledge in some essential areas of discrete optimization such as linear and integer programming, polyhedral theory, complexity theory as well as basic notions of graph theory. Otherwise he or she may want to consult classic textbooks addressing those topics, e.g. [GLS88] (algorithmic aspects of linear and convex programming, polyhedral theory, among others), [GJ79] (NP-completeness, featuring many famous NP-complete problems), [KV07] (combinatorial optimization), [Sch86] (a large 3-Volume compendium dealing with various topics in discrete optimization), [AMO93] (especially for classical network flow algorithms), [BMMN95] (network design models), depending on the area needing a refreshment or a lookup reference. However, some definitions which are beyond the basic notions of the respective fields mentioned will be given at the respective passages.

2.1 A Selection of Problems on Transportation Networks

Problems on transportation networks will be a repetitive theme throughout this work, as almost all chapters aim at devising methods to solve a transportation problem of some kind—or can be motivated by such a problem. Therefore, we will now introduce a few typical problems on networks. Starting with classical linear problems, we will

work our way up to nonlinear nonconvex problems. The problems discussed are often studied in different contexts and can be found in many textbooks e.g. [BMMN95]. Most problems will be of importance in one or several of the subsequent chapters, being either directly targeted by methods developed in this work or appearing as subproblems. Also a couple of related problems or variations are mentioned that appear interesting in the given context, even if their role in this work is not central. The following list of problems is of course by no means comprehensive, neither is the selection meant to reflect a problem's popularity or overall importance outside of this work.

2.1.1 The Basic Linear Network Flow problem—Notations

Network flow problems are defined on some network, which will be given as a directed graph $G = (V, A)$ with a set of vertices (or nodes) V and a set of edges or arcs (or directed edges) $A \subseteq V \times V$. The problem asks for finding a feasible or optimal flow of goods through the network such that a certain demand pattern is satisfied or optimized with respect to an objective function—or asks for (optimal) decisions that enable such flows.

We will discuss some notation using the example of the following basic version: together with the network, we are given demands d_v for each node $v \in V$ and nonnegative capacities c_a for each arc $a \in A$. The aim is to find a feasible routing of continuous quantities of a single good through the network such that each node's demand is satisfied and the flow on each arc does not exceed the arc's capacity. This problem is referred to as *b-transshipment*, e.g. in [Sch86, Chapter 11, Volume A] (though 'd-transshipment' would be more suitable for our notation). It is described by the following linear programming model:

$$\text{find} \quad x \tag{2.1a}$$

$$\text{s.t.} \quad \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a = d_v \qquad (\ \forall v \in V\ ) \tag{2.1b}$$

$$x_a \leq c_a \qquad (\ \forall a \in A\ ) \tag{2.1c}$$

$$x \in \mathbb{R}_+^{|A|}. \tag{2.1d}$$

For each arc $a \in A$, there is a nonnegative variable x representing the flow along that arc. Equations (2.1b) ensure *flow conservation*. Here, $\delta^+(v)$ denotes the set of arcs leaving node v, while the set of nodes entering node v is denoted by $\delta^-(v)$. In this work, we interpret d_v as a surplus (rather than an amount requested, as may be the convention used in other sources), hence we call a node with positive demand a *source* and a node with negative demand a *sink*. It is of course necessary to have

$$\sum_{v \in V} d_v = 0 \tag{2.2}$$

for the problem to be feasible. Equation (2.2) will be called *flow balance*.

Equations (2.1c) restrict the arc flows to the corresponding capacities. Our modeling at this point assumes one-way arcs. If the graph is undirected and flow can go in both directions, we can model this by having two arcs per undirected edge, both with the same capacity, obtaining a *bidirected graph*. This is the case for the majority of graphs in Chapter 3. Alternatively, we may allow potentially negative flow in the interval $[-c_a, c_a]$. This will be more convenient in Chapter 4. Both alternatives are equivalent as long as it can be assured that there is an optimal solution that does not simultaneously send positive flow along the forward and backward arc of an edge (for which there is no equivalent in the version that relaxes (2.1d)).

If we dropped Equations (2.1c), flow would always be sent along a shortest path and it suffices to determine which sources serve which sinks. This is usually called the *Transportation Problem*. As the structure of the network is not of importance for this problem, we will not discuss its usual modeling here in more detail. However, I would like to mention that the early work [Bal65], which inspired the aggregation algorithms developed in Chapter 3, aims to solve transportation problems (the concept of graph aggregation will be defined in that chapter).

As it is stated above, (2.1) is a feasibility problem (indicated by '$\text{find } x$', instead of a maximization or minimization). Some very classic problems of combinatorial optimization are obtained when considering specific objectives and/or small variations in the constraints.

The Maximum Flow Problem One may give a designated source-sink pair (s, t) of nodes and ask to maximize the flow from s to t through the network. In this setting, usually the demand at all other nodes is 0 (if not, this can be easily achieved by a standard trick, see below). Formally, the demand d at the source node s is then a variable whose value is to be maximized. The problem can be modeled as follows:

(2.3)
$$\begin{aligned} \max \quad & d \\ \text{s.t.} \quad \sum_{a\in\delta^+(v)} x_a - \sum_{a\in\delta^-(v)} x_a \quad &= \quad \begin{cases} d & \textit{if} \quad v = s \\ -d & \textit{if} \quad v = t \\ 0 & \textit{otherwise} \end{cases} && (\ \forall v \in V\ ) \\ x_a \quad &\leq \quad c_a && (\ \forall a \in A\ ) \\ x \quad &\in \quad \mathbb{R}_+^{|A|} \\ d \quad &\in \quad \mathbb{R}_+. \end{aligned}$$

The Minimum Cost Flow Problem By adding a linear objective

$$\min \sum_{a\in A} k_a x_a$$

to (2.1) we obtain the *Minimum Cost Flow Problem*. The (usually nonnegative) objective coefficients k_a can be interpreted as costs associated with routing a unit of flow along

arc a. A variation with additional lower and upper bounds on the flow values is called the *Circulation Problem*.

Both the Maximum Flow Problem and the Minimum Cost Flow Problem are well known to be solvable in strongly polynomial time and a number of sophisticated combinatorial algorithms exist for obtaining good bounds on the degree of the polynomial. For the Maximum Flow Problem, there is the *Preflow-Push Algorithm*. The generic version has a running time of $\mathcal{O}(n^2m)$, where as usual $n = |V|$ and $m = |A|$ denote the number of vertices and arcs of the network, respectively. This bound can be further improved by using special data structures and clever analysis. The implementation of the *FIFO Preflow-Push Algorithm* by Goldberg and Tarjan [GT88] achieves a bound on the time complexity of $\mathcal{O}(nm \log(\frac{n^2}{m}))$. Details on polynomial maximum flow algorithms can be found e.g. in [AMO93, Chapter 7].

The Minimum Cost Flow Problem can be also solved in strongly polynomial time by a number of combinatorial algorithms. The most famous is probably the *Minimum Mean Cycle-Canceling Algorithm* [GT89], which has a worst-case time complexity of $\mathcal{O}(n^2m^3 \log(n))$; it is presented together with other efficient minimum cost flow algorithms (many of which involve scaling techniques) in [AMO93, Chapter 10]. The fact that the Maximum Flow Problem and the Minimum Cost Flow Problem can be written as a linear program also allows to solve them in polynomial time by the *Ellipsoid Method*—although this is not competitive in practice. For details on this method, consult e.g. [GLS88, Chapter 3] or [KV07, Chapter 4].

However, whenever we have to solve linear network flow problems as subproblems in this work—which happens to quite some extend in Chapter 3, we will use a standard LP-solver that relies on the *Simplex Method*. Although theoretically it has exponential worst-case complexity, the Simplex Method is fast in practice, and a potential speedup by using specialized implementations is negligible as the implementation only spends a tiny fraction of the total running time with solving network flow subproblems.

Model (2.1) represents the *flow formulation* (or *arc-based formulation*) for network flow problems that models flow by explicit flow variables on each arc. Besides being probably the most popular and intuitive formulation, it is well established and extremely flexible in the sense that it is applicable (and practical) for a large variety of network flow problems. Models using a flow formulation will be our choice throughout this work for computational experiments, in particular for demonstrating the practical impact of the aggregation algorithms developed in Chapter 3. However, it should be mentioned that there are alternatives, e.g. *path formulations* and the *cut-set formulations*. Path formulations state problem constraints in terms of variables associated with feasible source-sink paths; whereas the cut formulations—when applicable—fully define the problem in terms of flow conservation equations for certain subsets of nodes. We do not want to go into detail here and just point out that those alternative formulations can be particularly well suited for some special types of more complicated network problems. For instance, path formulations tend to be popular for multi-commodity flow problems [BHJS94], whereas cut-based formulations are more common in the

context of network design problems [Bar96]. Advantages of the latter formulation for multi-scenario network design problems will also be remarked in the respective paragraph in Subsection 2.1.4 below. We will briefly discuss the applicability of our aggregation algorithms to those formulations in Section 3.6.

2.1.2 Analyzing Infeasibility of the Linear Network Flow Problem

If Problem (2.1) is infeasible, one may want to localize the main reason for infeasibility in the network. Such information is very valuable for guiding disaggregation in Chapter 3. For linear network flow problems this is nicely possible by using duality theory. Therefore, we will write (2.1) as a maximum flow problem.

Firstly, Problem (2.1) can be transformed to have a single source and a single sink. We introduce a super source s and a super sink t together with additional artificial arcs from s to all source nodes, and from all sink nodes to t. The idea is that s supplies all sources with their original demands d_v, and t collects all incoming flows from the sinks. Let V_+ denote the set of sources, i.e. nodes v with $d_v > 0$, and V_- denote the set of sinks of Problem (2.1). Then consequently, an arc (s, v) from the super source to some source $v \in V_+$ has capacity $|d_v|$, the same goes for an arc (v, t) from some sink $v \in V_-$ to t. After that, we can solve the problem as a maximum flow problem. This has the advantage that by the well-known correspondence between maximum flows and minimum cuts we can determine a set of edges in the network that limit the flow in case (2.1) is infeasible. At this point, we assume that s and t are connected in the network.

We obtain the following model:

(2.4) $\max \quad d$

$$
\begin{aligned}
\text{s.t.} \quad \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a &= \begin{cases} d & \textit{if} \quad v = s \\ -d & \textit{if} \quad v = t \\ 0 & \textit{otherwise} \end{cases} && (\ \forall v \in V \cup \{s, t\}\ ) \\
x_a &\leq c_a && (\ \forall a \in A\ ) \\
x_{(s,v)} &\leq d_v && (\ \forall v \in V_+\ ) \\
x_{(v,t)} &\leq -d_v && (\ \forall v \in V_-\ ) \\
x &\in \mathbb{R}_+^{(|A|+|V_+|+|V_-|)} \\
d &\in \mathbb{R}_+.
\end{aligned}
$$

It is easy to see that Problem (2.1) is feasible if and only if (2.4) has a solution where the optimal objective d^* is equal to $D := \sum_{v \in V_+} d_v$, the total amount of flow to be routed. By construction, this is only possible if all artificial arcs are saturated.

Otherwise, if $d^* < D$, we compute the minimum cut corresponding to the optimal flow. Consider the dual linear program of (2.4), which is given below. Let $\tilde{A}$ be the set of arcs of the underlying augmented graph of Problem (2.4), including the artificial arcs. If $a = (s, v)$ or $a = (v, t)$ is an artificial arc, c_a is interpreted as $|d_v|$ according to

(2.4) above.

$$
\begin{aligned}
(2.5) \qquad \min \quad & \sum_{a \in \tilde{A}} c_a \omega_a & & & \\
\text{s.t.} \quad & \pi_u - \pi_v + \omega_a & \geq & 0 & (\ \forall a = (u,v) \in \tilde{A}\ ) \\
& \pi_t - \pi_s & \geq & 1 & \\
& \pi_v & \in & \mathbb{R} & (\ \forall v \in V \cup \{s,t\}\ ) \\
& \omega_a & \in & \mathbb{R}_+ & (\ \forall a \in \tilde{A}\ )
\end{aligned}
$$

In (2.5), π_v denote dual variables corresponding to the flow conservation constraint at some node $v \in V \cup \{s,t\}$. The value of those variables is also called the *node potential* at node v. For an arc $a \in \tilde{A}$, the variable ω_a represents the dual variable corresponding to the capacity constraint belonging that arc. It expresses the *potential drop* along arc a. We will stick to this notation for duals of linear network flow problems throughout this work. Now, arcs a with $\omega_a > 0$ form a cut in the graph for any feasible solution. By complementary slackness, we know that $\omega_a > 0$ implies $x_a = c_a$, so this gives a minimum cut that limits the flow from s to t.

In Chapter 3, where this will be of importance, we do not have to implement (2.5) explicitly in practice. Dual information is comfortably available from LP-solvers after the primal has been solved successfully, so we can just read off a minimum cut from the dual of the of the capacity constraints ('shadow prices'). If the minimum cut is not unique, the node potential π might not drop from 1 directly to 0 (from t to s); instead, we may observe $0 < \omega_a < 1$ for some a, with the interpretation that the potential drops stepwise over several minimal cuts. In this case, the set $\{a \in \tilde{A} \mid \omega_a > 0\}$ would strictly contain minimum cuts, which might not be what we want. However, the constraint matrix of linear network flow problems is well known to be totally unimodular [Sch86, Chapter 13, Volume A], and hence, all *basic* feasible solutions are naturally integer. Since we generally use the Simplex Algorithm for solving linear programming subproblems (as already mentioned above), we will obtain a basic feasible solution that is integral, which in this case means that $\{a \in \tilde{A} \mid \omega_a > 0\}$ is an *elementary* minimum cut, i.e. removing any arc from the cut leaves the network connected.

If we want to interpret this cut $\tilde{C} \subseteq \tilde{A}$ back in Problem (2.1), we have to ignore artificial cut arcs but only consider arcs $C := \tilde{C} \cap A$ of the original network. We can guarantee the following:

Proposition 2.1. *If* (2.1) *is infeasible, there is at least one source-sink pair* (v_+, v_-), $v_+ \in V_+, v_- \in V_-$ *such that there is no path from* v_+ *to* v_- *in the graph* $G' = (V, A \backslash C)$.

Proof: If $\tilde{C} \supseteq \delta^+(s)$, we know by the Max-Flow Min-Cut Theorem that the maximum flow in problem (2.4) is equal to $\sum_{a \in \delta^+(s)} c_a = \sum_{v \in V_+} d_v = D$. Therefore, (2.1) is feasible. The same follows if $\tilde{C} \supseteq \delta^-(t)$. Hence, there must be $v_+ \in V_+, v_- \in V_-$ with $(s, v_+) \notin \tilde{C}$ and $(v_-, t) \notin \tilde{C}$. As we know that $\tilde{C}$ cuts all paths from s to t in the augmented graph, there can be no path from v_+ to v_- in the original graph. □

Hence, if the cut $\tilde{C}$ consists of artificial arcs only, then either

a) Problem (2.1) is feasible.

b) In the original graph not every source is connected to every sink.

In Chapter 3 we can exclude b) by the structure of the instances considered, and therefore ensure that we always obtain an interpretable cut if the input problem of type (2.1) is infeasible.

In formulation (2.5), we see that any feasible solution $(\hat{\pi}, \hat{\omega})$ is dominated by the solution $(\hat{\pi}, \omega)$ with $\pi = \hat{\pi}$ and

$$\omega_a = \max\{\hat{\pi}_v - \hat{\pi}_u, 0\} \qquad (\ \forall a = (u,v) \in A\ )$$

If the graph is undirected—expressed by either modeling discussed in Subsection 2.1.1, we have the equations

$$\pi_u - \pi_v + \omega_a = 0 \qquad (\ \forall a = (u,v) \in A\ )$$

instead of inequalities in the dual problem. Hence, the ω-variables can directly be eliminated from formulation (2.5) and we obtain a dual network flow problem of the following form:

$$\begin{aligned}
&\text{(2.6a)} && \min && \sum_{v \in V \cup \{s,t\}} \tilde{c}_v \pi_v && && \\
&\text{(2.6b)} && \text{s.t.} && \pi_v - \pi_u \ \geq\ 0 && (\ \forall a = (u,v) \in \tilde{A}\ ) \\
&\text{(2.6c)} && && \pi_t - \pi_s \ \geq\ 1 && \\
&\text{(2.6d)} && && \pi_v \ \in\ \mathbb{R} && (\ \forall v \in V \cup \{s,t\}\ )
\end{aligned}$$

for suitable $\tilde{c}_a$, where the ω-variables have been replaced in (2.6a). Constraints (2.6b) remain to represent the nonnegativity of ω.

Such a representation using only π-variables will be used in Chapter 5 to show that a certain formulation represents a dual network flow problem and is therefore totally unimodular. As total unimodularity only depends on the constraint matrix, we can allow a slightly more general version with arbitrary right-hand side in constraints of type (2.6b):

$$\begin{aligned}
&\text{(2.7)} && \min && \tilde{c}^T \pi && && \\
& && \text{s.t.} && \pi_v - \pi_u \ \leq\ k_a && (\ \forall a = (u,v) \in A\ ) \\
& && && \pi_v \ \in\ \mathbb{R} && (\ \forall v \in V \cup \{s,t\}\ )
\end{aligned}$$

Such a formulation may be obtained by dualizing a Minimum Cost Flow Problem.

2.1.3 The Network Design Problem

The next problem we consider involves making a decision on how to design the network in order to enable a certain flow. As before, each arc $a \in A$ possesses initial arc capacities $c_a \geq 0$. In addition, each arc can be upgraded by installing a module with an upgrade capacity of C_a at a price of k_a per unit, available in integral multiples y_a. The aim is to determine a feasible routing of a specified demand vector $d \in \mathbb{R}^{|V|}$ that incurs a minimal-cost upgrade of the network while respecting the capacities of the arcs. A mixed-integer programming (MIP) formulation of the single-commodity flow network design problem is given by

$$
\begin{aligned}
(2.8) \qquad \min \quad & \sum_{a \in A} k_a y_a & & & \\
\text{s.t.} \quad & \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a & = & \; d_v & \quad (\ \forall v \in V\ ) \\
& x_a & \leq & \; c_a + C_a y_a & \quad (\ \forall a \in A\ ) \\
& x & \in & \; \mathbb{R}_+^{|A|} & \\
& y & \in & \; \mathbb{Z}_+^{|A|}. &
\end{aligned}
$$

The above formulation only considers a single type of additional module per network arc. Note, however, that an extension to multiple types of additional modules with varying cost and capacity is possible in a straightforward way.

In contrast to the problems on networks that we examined so far, the capacitated network design problem (NDP) is well known to be NP-hard [JLK78]. A standard solution method is Lagrangian relaxation, which was first proposed in [Geo74]. Reference [Lem01] evaluates advanced theoretical results and numerical aspects, and relates it to other techniques such as column generation. In [KR79], relationships to surrogate duality in integer programming are investigated. Another approach that has been used intensively for capacitated network design problems is Benders decomposition. This method was first proposed in [Ben62]. More theoretical background can be found in [MD77] and [HO03]. The work [Cos05] gives a broad survey on the application of Benders decomposition to fixed-charge network design problems. A favorable cut selection criterion for Benders cuts is proposed and analyzed computationally in [FSZ10].

Note that some or even all of the initial arc capacities c_a may have a value of zero. This basically means that the arc is nonexistent unless it is constructed for the setup cost of k_a. In this case, the variable y_a is often restricted to be binary rather than in $\mathbb{Z}_+$, with the interpretation that y_a determines whether the connection a is established or not. However, in practical applications, it is very common that a relatively developed network has to be upgraded in order to allow for the routing of additional demand requirements. In this case, we call (2.8) a *network expansion problem*, though this of course is not a formal definition.

In Chapter 3, an algorithmic scheme for solving network expansion problems is proposed that is based on iterative graph aggregation.

2.1.4 Further Extensions of Linear Network Design Problems

There is an abundance of additional constraints that network design problems—or network flow problems in general—may have. We briefly discuss some of them here that we will revisit in Chapter 3 in connection with the question of applicability of our aggregation algorithms. In this subsection we only consider extensions of the network design problems that are still representable as an MILP, whereas the subsequent subsection will introduce gas network optimization as an example of a nonlinear network flow problem.

Multi-Commodity Flows Instead of having a single commodity and scalar demands at each node, the problem may feature multiple commodities $i \in \mathcal{I}_d$, where $\mathcal{I}_d$ is some discrete index set, that have to be routed through the network simultaneously. Therefore, the demand of a node is a vector specifying that node's demand for each commodity. We can extend (2.8) to model multi-commodity flow problems by increasing the dimension of the flow variables in order to track the flow on that arc for each commodity, and adapting the constraints as follows:

$$
\begin{aligned}
(2.9) \qquad \min \quad & \sum_{a \in A} k_a y_a \\
\text{s.t.} \quad & \sum_{a \in \delta^+(v)} x_{a,i} - \sum_{a \in \delta^-(v)} x_{a,i} = d_{v,i} && (\ \forall v \in V,\ \forall i \in \mathcal{I}_d\ ) \\
& \sum_{i \in \mathcal{I}_d} x_{a,i} \leq c_a + C_a y_a && (\ \forall a \in A\ ) \\
& x \in \mathbb{R}_+^{|A \times \mathcal{I}_d|} \\
& y \in \mathbb{Z}_+^{|A|}.
\end{aligned}
$$

The combination of network extensions together with multiple commodities actually requires additional modeling decisions regarding the intended interaction between network extensions and forward and backward arcs. The reason is that in contrast to all previous models we cannot guarantee that there is always an optimal solution that does not simultaneously send positive flow along the forward and backward arc of an edge. Forward and backward flow simply may belong to different commodities and therefore can not be treated as canceling out each other. Formulation (2.9) implies the following choice: upgrading an arc increases the capacity only for one direction. The alternative can also be reasonable depending on the application, e.g. when dealing with a telecommunication network. This can be modeled by merging all variables y_a where a represents the same edge connection. This has been the choice in the computations on multi-commodity flow network design in Chapter 3, Subsection 3.6.1. A common special case of (2.9) restricts each demand to be nonzero only at two nodes, i.e. the demand is given by source-sink pairs. This is motivated e.g. by applications in logistics, where passengers or communication messages have a single origin and destination.

Routing Costs In network design problems we are mainly interested in the network's minimal extension that enables a feasible routing whereas the costs of realizing the routing itself are negligible in comparison. However, if we want to account for routing costs in our model formulation, we only have to modify the objective function

$$\min \sum_{a \in A} f_a x_a + k_a y_a,$$

where f_a specifies the charge per unit of flow along arc a. Though this seems like a minor change, it can have severe consequences for solution algorithms. One thing to mention is that the cut formulation is no longer directly applicable, as it has no representation for the actual flows.

Instationary flows via time-expanded graphs Flows in transportation networks are often instationary in their essence. Stationary models as the ones stated above are often times still useful, as their solutions represent steady states that a network can attain over a longer period of time, and which are therefore desirable. However, we will briefly describe how to model instationary flows via time-expanded graphs. Suppose network arcs have nonnegative travel times $\Delta t_a \in \mathbb{R}_+$ that indicate the time that the flow needs to bridge the distance from the source node of arc a to its target node. Given a time horizon of T, the task is to find a network expansion of minimal cost that allows all flow to be routed within time T. For this problem we consider an equidistant time-discretization of the time horizon into a set of possible starting times $\mathcal{T} = \{0, \dots, T\}$ such that $t + \Delta t_a$ is either an element in $\mathcal{T}$ or greater than the time horizon T, for all $t \in \mathcal{T}$ and $a \in A$. For the sake of simplicity we assume $\mathcal{T} = \mathbb{Z} \cap [0, T]$.

This can be represented as a classical network design-problem on the so-called time-expanded graph $G_{\text{texp}} = (V_{\text{texp}}, A_{\text{texp}})$. It is constructed by creating $|\mathcal{T}|$-many copies of G, one for each time step, i.e. $V_{\text{texp}} = V \times \mathcal{T}$. The arc set is defined by

$$a = ((v_1, t_1), (v_2, t_2)) \in A_{\text{texp}} \Leftrightarrow (v_1, v_2) \in A \text{ and } t_1 + \Delta t_{(v_1, v_2)} = t_2,$$

i.e. traveling from v_1 to v_2 is feasible in G and the travel time of that arc is equal to the difference of the time indices of both nodes in G_{texp}. In addition, we have the arc $((v, t), (v, t+1)) \in A_{\text{texp}}$ with infinite capacity for all $v \in V,\ t \in \mathcal{T} \backslash \{T\}$ if 'waiting' at a node is feasible in the application context. In this case, a node's original demand d_v is assigned to its 'latest' copy (v, T). As before, there are arc capacities that must not be exceeded at any time step in $\mathcal{T}$, so they can be assigned to the corresponding arcs in A_{texp}.

This way, we can solve instationary network design problems by solving a problem of type (2.8), but on a larger graph. However, important properties of G might of course not transfer to G_{texp}, e.g. if G is bidirected (as in Chapter 3), the time-expanded graph is usually not, as sending flow 'to the past' is not feasible.

Multiple Scenarios In applications we often have a fixed given network topology, on which there are several *scenarios* to be solved that vary in the node demands—or sometimes other data such as e.g. initial arc capacities or the type of expansion modules. However, we will focus on demands here. We speak of a *Multi-Scenario Problem* if we seek a decision (network expansion) that allows for a feasible routing of *all* given scenarios of a set $\mathcal{U}$ of possible scenarios. We obtain a problem formulation of the following type:

$$
\begin{aligned}
(2.10) \qquad \text{find} \quad & y \\
\text{s.t.} \quad \forall S \quad & \in \quad \mathcal{U} \quad \exists x \in \mathbb{R}_+^{|A|} : (x, y) \text{ is feasible for (2.8)} \\
y \quad & \in \quad \mathbb{Z}_+^{|A|}.
\end{aligned}
$$

This is a bilevel problem, where we have to make a first-stage decision on the expansion y before the uncertainty is observed, and a second-stage decision on the routing that may depend on the scenario. Using the terminology of robust optimization, we call $\mathcal{U}$ the *uncertainty set*. It may not necessarily be finite, but separating over $\mathcal{U}$ (i.e. determining a scenario $S \in \mathcal{U}$ that does not allow for a feasible routing or determining that no such scenario exists) should be possible if we want to stand a chance for solving (2.10) to optimality. This is true e.g. if $\mathcal{U}$ is polyhedral, which leads to the so-called *Hose polytope*. For investigations on the complexity of single-commodity network design with a Hose polytope as the scenario set, see e.g. [CJL+16]. The paper makes use of the fact that one can remove the second stage by switching to a cut-formulation, since this formulation can express feasibility without flow variables.

Alternatively, whenever we have a finite uncertainty set $\mathcal{U} = \{S_1, \dots, S_k\}$, a well-known technique for resolving the second stage is constructing the so-called *deterministic equivalent* of (2.10). This is done by creating a copy $x_{a,i}$ of flow variable x_a for every possible scenario $S_i,\ i = 1, \dots, k$, with the meaning that $x_{a,s}$ represents an optimal choice for x_a in case of scenario s. This leads to the following formulation:

$$
\begin{aligned}
(2.11) \qquad \min \quad & \sum_{a \in A} k_a y_a \\
\text{s.t.} \quad & \sum_{a \in \delta^+(v)} x_{a,s} - \sum_{a \in \delta^-(v)} x_{a,s} \quad = \quad d_v \qquad (\ \forall v \in V,\ \forall s \in \mathcal{U}\ ) \\
& x_{a,s} \quad \leq \quad c_a + C_a y_a \qquad (\ \forall a \in A,\ \forall s \in \mathcal{U}\ ) \\
& x \quad \in \quad \mathbb{R}_+^{|A \times \mathcal{U}|} \\
& y \quad \in \quad \mathbb{Z}_+^{|A|}.
\end{aligned}
$$

Note that throughout this work, a demand pattern constituting an instance of the problem will be called a *scenario* even if we do not consider the problem as a multi-scenario problem—especially in Chapter 6, where we consider different demand scenarios and objective functions on the same gas network.

It goes without saying that this list of linear extensions of network design problems introduced above—in order to be referred to in Chapter 3—is not exhaustive. Plenty

of variants and additional constraint can be found across the literature and of course also combinations of them are possible.

2.1.5 Gas Network Optimization

As an example of a nonlinear network flow problem, we will consider optimization tasks on gas networks. In gas network optimization flow may not be split arbitrarily at a node, but is driven by a potential, namely the pressure. Gas flow is bound to be directed from higher to lower pressure following certain physical laws. In this regard, gas networks are similar to water supply networks or electricity networks [GMMS12, KW84].

In order to model gas networks, we introduce additional variables p_v describing the pressure at node v. Though p_v can be seen as a node potential, we do not use π-variables as in the dual flow problem (2.5) to be consistent with the literature. Moreover, p_v does not have the same interpretation as a shadow price that is valid for π in the linear case. Following a convention, the flow on an arc a will be denoted by q_a (instead of x_a) in the context of gas network optimization throughout this work. Furthermore, arcs will also be called pipes and are naturally undirected. Therefore we allow q-variables to be negative, with the interpretation that there is positive flow against the pipe's formal direction.

Passive Networks

An important source of nonlinearity is due to the pressure loss along pipes. It is the only source of nonlinearity in so-called *passive* networks, which consist of pipes only and do not feature elements to actively operate the network. Therefore, gas flow is fully determined by the laws of physics.

At a high level of detail, the pressure loss along pipes can be modeled by the *Euler Equations*, a set of nonlinear hyperbolic partial differential equations that are suitable to describe compressible fluids [Fei93]. They consist of the *Continuity Equation*, the *Momentum Conservation Equation* and the *Energy Conservation Equation* and involve additional quantities as the gas temperature, density and velocity, among others.

In order to obtain an algebraic pressure loss equation, we have to make some simplifying assumptions. As pipes in Germany are usually well beneath the ground, we may assume constant temperature. Also, if we are interested in solutions that are stable over time, we may impose *stationarity*, which is also an implicit assumption in most linear network flow models discussed so far. This means that all time derivatives vanish. Further simplifications include horizontal pipes and the so-called compressibility factor being constant.

After applying the simplifications, the set of differential equations can be solved analytically, leading to an algebraic approximation that is commonly used in practice, called the *Weymouth Equation* (see [KHPS15, Equation (7.2)]), in our case for the special

case of horizontal pipes). It has the form

$$\Delta(p^2) = \lambda q|q|$$

for some parameter $\lambda \in \mathbb{R}_+$ that depends on pipe properties as length, diameter and roughness. Here, $\Delta(p^2)$ denotes the reduction of the squared pressure along a pipe (cf. formulation (2.12) below). A detailed derivation of this algebraic model can be found in [Gei11b, Chapter 6 & Appendix B] or [KHPS15, Section 2.3.1]. Though the model is used in the computations mentioned in Chapter 3, Section 3.7, the principal approach is mainly independent from the chosen pipe model—as long as it can be handled by a tractable global optimization method. On the contrary, in Chapter 6 we will explicitly make use of the quadratic nature of this model. In addition, fixed flow directions will be assumed there in order to get rid of the non-smoothness. The setting in Chapter 4 is suited to deal with problems featuring low-dimensional nonlinearities as the right-hand side of the above equation, though the abstract setting in that chapter is more general and no particular structure on the nonlinearity is assumed.

The above approximation yields an example for a nonconvex problem on transportation networks, where nonlinearities are sufficiently 'well-behaved' to allow for structure-exploiting approaches such as simultaneous convexification in Chapter 6.

We can give the following nonlinear programming formulation for a gas network feasibility problem on passive networks:

(2.12)
$$\begin{aligned} \text{find} \quad & q, p \\ \text{s.t.} \quad & \sum_{a \in \delta^+(v)} q_a - \sum_{a \in \delta^-(v)} q_a &&= d_v && (\ \forall v \in V\ ) \\ & p_u^2 - p_v^2 &&= \lambda_a q_a |q_a| && (\ \forall a = (u,v) \in A\ ) \\ & q_a &&\in [\underline{q_a}, \overline{q_a}] && (\ \forall a \in A\ ) \\ & p_v &&\in [\underline{p_v}, \overline{p_v}] && (\ \forall v \in V\ ). \end{aligned}$$

Source and sink nodes (in this context also called *entries* and *exits*, respectively) usually have a minimum and/or maximum pressure requirement, denoted by $\underline{p_v}$ and $\overline{p_v}$ respectively, that is specified by the gas provider or consumer. Still, also pressure bounds for inner nodes, i.e. nodes that are neither sources nor sinks, can be stated by technical limitations. There are also lower and upper bounds $\underline{q_a}$ and $\overline{q_a}$ for the flow on each pipe, though they usually are not a limiting factor in the network. The absolute value of a flow of course could be reformulated by a binary variable, obtaining a mixed integer nonlinear programming formulation with smooth nonlinearities. Note that for passive networks we may replace all pressure variables by variables representing the squared pressure, thus directly removing some nonlinearities.

So far, all constraints are derived from physical laws such that it is not surprising that for a given demand vector d_v, fixing a single pressure value in a connected network already guarantees uniqueness of a solution. A proof is given in [RMWSB02].

Active Elements

Real-world gas networks have *active elements* that allow operation of the gas network, e.g. by shutting off connections or raising the pressure. We will briefly discuss different important types of active elements in the following. However, we will not give full details as the specific modeling will not be important for the methods developed in the subsequent chapters. Instead, their focus clearly is on the network part and the nonlinearities that are present on every arc. In Chapter 3, Section 3.7 the presented adaptation of the iterative aggregation scheme is meant to deal with networks featuring active elements. Yet, the treatment of active elements by the solver is not influenced by the algorithm apart from assigning them either to the aggregated master problem or to the subproblem. Readers interested in a high level of detail on modeling gas network elements are referred to [KHPS15, Chapter 2.3].

Active elements are usually modeled as a special type of arc. **Valves** are the simplest type of active element. A valve is either open or closed. An open valve essentially is a pipe that causes no pressure drop, hence the pressure values on both endpoints of the valve have to agree. If the valve is closed, no flow is allowed to pass and the pressure values on the valve's endpoints are decoupled.
Valves are discrete structures that essentially allow modification of the network topology. They are usually modeled via a binary variable that indicates whether the valve is open or closed. In a network design setting as in Chapter 3, Section 3.7, they can also be used to model whether a new pipe should be built rather than being physically present in the network already. In this case, the candidate pipe's building costs are attached to the choice of the valve being open.

Control Valves are a type of directed active element that allow to reduce the pressure for gas flowing in a specific direction. Similar to valves, control valves are either open or closed, and if closed they prevent any flow from passing along both directions. Consequently, pressure values at the control valve's endpoints are decoupled. If a control valve is open, flow is allowed to pass in a given direction, and the pressure is reduced by a controllable amount in a given range that is determined by the valve specification. That means that they introduce a continuous degree of freedom for the network operator. In practice, control valves are used to regulate pressure at transition points between network components that typically have different overall pressure levels. They can be modeled by several linear equations with an additional binary variable that represents whether or not the control valve is open [Gei11b].

Compressors are directed active elements that allow to increase gas pressure. The possible pressure values at the target node of the compressor depend on the inflow and the inlet pressure. The operating range of a compressor is determined by its so-called *characteristic diagram*, which originates from least-square fits of measured data points. There are different approximation levels of the operating range. In this work, we will just assume that the compressor model can be handled by a global mixed integer nonlinear programming solver (possibly after piecewise relaxation). For example, we might think of a polyhedral approximation model in terms of bounds for the com-

pression ratio, pressure increase and power consumption, which has been the choice in [Gei11b]. In a simplified model, all those quantities can be described in terms of the inflow and the inlet and outlet pressure values, in the case of the power consumption by a nonlinear inequality (that might as well be approximated linearly). In real-world gas networks, compressors are essential to compensate for the pressure loss over long distances due to pipe friction. Also, compressors can be operated in *bypass* mode, where flow may pass in both directions without being compressed. Depending on the modeling, compressors induce quadratic or more complicated nonlinearities in addition to a binary variable that indicates whether the compressor is operated in bypass mode. As compressors consume energy, this is a common choice to optimize in real-world instances.

Compressors are usually grouped—conceptually as well as geographically—into **Compressor stations** that comprise several compressor machines. Those can be interconnected in various different ways, e.g. a subgroup of compressors may be run in series or in parallel. Usually not every conceivable interconnection diagram is technically realizable or sensible in realistic cases. Therefore, a discrete set of feasible configurations may be decided on beforehand such that the operator only has to choose one of those possibilities to specify routing inside a compressor station.
The task of choosing optimal configurations adds a lot of complexity to the problem, which makes large compressor stations a main computational challenge on the discrete-optimization side.

Real-world gas networks contain additional elements that cause pressure loss, even though it is not their dedicated purpose, e.g. filtration plants, measuring devices, gas-preheaters and gas-coolers. While their functionality is usually not included in most models, we may account for the pressure loss by modeling them as **Resistors**. Each resistor may be modeled as increasing pressure reduction, either flow-dependent or by a given constant that only depends on the particular element. Moreover, the pressure loss due to a resistor may depend on the direction of transition. Resistors can lead to a pressure drop behavior that is different from that of a pipe. In particular, it cannot be compensated by adapting the pipe's λ-parameter.

A full MINLP-model extending (2.12) can be found in [Gei11b, Chapter 6]. As already mentioned, the reader interested in the topic of modeling and solving gas networks optimization will find detailed examination in [KHPS15].

The Network Design Problem on Gas Networks

As for the linear case, we may pose the problem of optimally extending a given network for gas networks as well. However, in order to model this problem in a sensible way, upgrading an arc should—in contrast to (2.8)—not simply effect the arc's 'capacity', i.e. the corresponding flow bound in (2.12). First of all, the upgrade should reflect the effect of adding an appropriate arc parallel to the existing one, and increasing the flow bound does not implement this concept for gas networks. Secondly, flow bounds are often times not the limiting factor in a gas network optimization problem, so up-

grading it may not lead to feasibility for any upgrade (see [KHPS15, Chapter 11] for experiments on that topic).

Instead, for a network design problem on a gas network we use a formulation that models extensions as ordinary network arcs but with the difference that additional arcs come together with a valve each such that access to the new arc is only granted by opening its valve. That way, we can associate the cost of creating the arc to a binary variable that represents whether the valve is open. Remember that closed valves effectively change the topology such that the connection may as well not exist.

For the sake of clarity, the model below assumes a passive network and accordingly only considers network extensions consisting of additional arcs. Though note that upgrade candidates that involve the construction of additional active elements may be modeled in the same way.

Given a set A_{ext} of possible upgrade arcs—that may or may not be parallel to already existing arcs—we extend (2.12) as follows: for each extension arc $e \in A_{\text{ext}}$ connecting nodes u and v with associated building cost k_e, we create an artificial network node n_e. This node is connected to the remaining network via a valve $a_{\text{valve},e} = (u, n_e)$ and a pipe $a_{\text{pipe},e} = (n_e, v)$. The modified node and pipe sets including those additional constructions are denoted by A' and V' respectively. New valves are subsumed in the set A_{valve}, where each valve $a \in A_{\text{valve}}$ has an associated binary variable y_a that indicates the valve's status. There is no need for shutting off new arcs at both endpoints. This is because no solution can send nonzero flow from the reverse side into the new arc due to the flow conservation at its inner node. We may use the following model for extending (2.12) to a network design problem:

$$
\begin{aligned}
&\text{(2.13a)} && \min && \sum_{a \in A_{\text{valve}}} k_a y_a && && \\
&\text{(2.13b)} && \text{s.t.} && \sum_{a \in \delta^+(v)} q_a - \sum_{a \in \delta^-(v)} q_a &= d_v && (\ \forall v \in V'\ ) \\
&\text{(2.13c)} && && p_u^2 - p_v^2 &= \lambda_a q_a |q_a| && (\ \forall a = (u,v) \in A'\ ) \\
&\text{(2.13d)} && && q_a &\leq \overline{q_a} y_a && (\ \forall a = (u,n) \in A_{\text{valve}}\ ) \\
&\text{(2.13e)} && && q_a &\geq \underline{q_a} y_a && (\ \forall a = (u,n) \in A_{\text{valve}}\ ) \\
&\text{(2.13f)} && && p_u - p_n &\leq (\overline{p_u} - \underline{p_n})(1 - y_a) && (\ \forall a = (u,n) \in A_{\text{valve}}\ ) \\
&\text{(2.13g)} && && p_u - p_n &\geq (\underline{p_u} - \overline{p_n})(1 - y_a) && (\ \forall a = (u,n) \in A_{\text{valve}}\ ) \\
&\text{(2.13h)} && && q_a &\in [\underline{q_a}, \overline{q_a}] && (\ \forall a \in A'\ ) \\
&\text{(2.13i)} && && p_v &\in [\underline{p_v}, \overline{p_v}] && (\ \forall v \in V'\ ) \\
&\text{(2.13j)} && && y_a &\in \{0,1\} && (\ \forall a \in A_{\text{valve}}\ ).
\end{aligned}
$$

Equations for extension pipes are integrated in (2.13c), whereas Equations (2.13d) to (2.13g) model the behavior of a valve. The pressure bound at all artificial nodes n_e can be copied from the extension's start node, while the flow bounds for $a_{\text{pipe},e}$ are determined by that of the extension candidate.

Problems of type (2.13) will be considered towards the end of Chapter 3.

2.2 Modeling Piecewise Linear Functions

In the last section, we have seen network problems involving nonlinearities. A common and established approach for dealing with those nonlinearities consists of constructing piecewise linearizations or relaxations of the involved nonlinear functions. The resulting MIP can then be solved by any general-purpose MIP solver. This is especially attractive for problems like (2.12), in which nonlinearities can be modeled as low-dimensional nonlinear functions. For this problem, the nonlinear constraint on the pressure loss along a pipe is separable, and it is sufficient to linearize the function

$$f\colon I \to \mathbb{R}, \quad q \mapsto q|q|, \tag{2.14}$$

where $I \subset \mathbb{R}$ is a compact interval, in order to obtain an MIP as we can use variables for squared pressure.

In this section, we will first discuss several methods for modeling piecewise linear functions, including some interesting properties and interconnections. In the course of this, we concentrate on univariate functions. Most of the popular methods can be extended to arbitrary dimensions; we will discuss the principal idea behind this briefly in Subsection 2.2.5 and otherwise refer the reader to [GMMS12] for more details. The concepts of piecewise linear approximation and relaxation will be introduced afterwards, in Subsection 2.2.6. In the following, let

$$\phi\colon I \to \mathbb{R}, \quad x \mapsto \phi(x)$$

be a piecewise linear one-dimensional function on a connected compact domain $I = [l, u] \subset \mathbb{R}$ with breakpoints $B_1 = l, B_2, \dots, B_k, B_{k+1} = u$. In particular, ϕ is linear on each *segment* $[B_i, B_{i+1}]$, $i = 1, \dots, k$. In the following, we implicitly assume ϕ to be continuous, though non-continuous functions can be modeled in the same style, as long as the function value at each breakpoint B_i may be chosen freely by the model within the highest and lowest function value in a neighborhood of B_i (or the correct choice is enforced by the objective function anyway). In Chapter 4, we will deal, among other things, with piecewise constant—but not continuous—objective functions that way.

Most of the formulations used in practice are locally ideal, i.e. for a single piecewise linear function their linear relaxation is equal to the convex hull of feasible points. However, this property is usually lost as soon as multiple piecewise linear functions are considered simultaneously. Chapter 4 deals with strengthening the formulation for multiple functions in the context of network flow problems for specific substructures. The choice of linearization method in practice mainly relies on empirical investigations for the particular application. For exemplary fields of application, the reader is referred to the references within [SLL13, Table 1].

2.2.1 The Multiple Choice Method

We start with the *Multiple Choice Method* (MCM) [JL84]. In this context we have a binary variable $z_i,\ i = 1, \ldots, k$ for each segment that indicates whether the value of x is contained in that segment $[B_i, B_{i+1}]$. In addition, there is a 'copy' x_i of x for every segment that is only activated if $z_i = 1$, and forced to zero otherwise. The following MIP describes the piecewise linear function ϕ, i.e. it has exactly the set of points on the function graph of ϕ, $\{(x, y) \mid y = \phi(x)\}$, as the feasible set.

$$\text{(2.15a)} \qquad \text{find} \quad x, y, z$$

$$\text{(2.15b)} \qquad \text{s.t.} \quad x = \sum_{i=1}^{k} x_i$$

$$\text{(2.15c)} \qquad y = \sum_{i=1}^{k} \phi(B_i) z_i + (x_i - B_i z_i) \frac{\phi(B_{i+1}) - \phi(B_i)}{B_{i+1} - B_i}$$

$$\text{(2.15d)} \qquad x_i \geq B_i z_i \qquad (\ \forall i \in \{1, \ldots, k\}\ )$$

$$\text{(2.15e)} \qquad x_i \leq B_{i+1} z_i \qquad (\ \forall i \in \{1, \ldots, k\}\ )$$

$$\text{(2.15f)} \qquad \sum_{i=1}^{k} z_i = 1$$

$$\text{(2.15g)} \qquad x_i \in [B_i, B_{i+1}] \qquad (\ \forall i \in \{1, \ldots, k\}\ )$$

$$\text{(2.15h)} \qquad x, y \in \mathbb{R}$$

$$\text{(2.15i)} \qquad z_i \in \{0, 1\} \qquad (\ \forall i \in \{1, \ldots, k\}\ )$$

In this formulation, (2.15b) establishes the intended connection between x and the copies $x_i,\ i = 1, \ldots, k$; y can then be computed using the formula in (2.15c). Note that it is linear as B_i and $\phi(B_i)$ are constants. Constraints (2.15d) and (2.15e) ensure that x_i is zero unless segment i is active, which corresponds to $z_i = 1$. Finally, exactly one segment can be active, which is encoded in (2.15f). The Multiple Choice Method leads to *locally ideal* formulations (see [VAN10]).

2.2.2 The Convex Combination Method

Another intuitive formulation is the *Convex Combination Method* (CCM), also known as *λ-Method* [Dan60]. The idea behind this method is that each point on a linear segment can be represented as a convex combination of the segment's two endpoints. Again, we have a binary variable $z_i,\ i = 1, \ldots, k$ for each segment. In addition, for each breakpoint we introduce an auxiliary continuous variable—typically named λ—that regulates the weight of that breakpoint for averaging.

$$\text{(2.16a)} \qquad \text{find} \quad x, y, \lambda, z$$

$$
\begin{aligned}
&\text{(2.16b)} && \text{s.t.} \quad x &&= \sum_{i=1}^{k+1} B_i \lambda_i && \\
&\text{(2.16c)} && y &&= \sum_{i=1}^{k+1} \phi(B_i) \lambda_i && \\
&\text{(2.16d)} && \lambda_1 &&\leq z_1 && \\
&\text{(2.16e)} && \lambda_{i+1} &&\leq z_i + z_{i+1} && (\ \forall i \in \{1, \ldots, k\}\ ) \\
&\text{(2.16f)} && \lambda_{k+1} &&\leq z_k && \\
&\text{(2.16g)} && \sum_{i=1}^{k+1} \lambda_i &&= 1 && \\
&\text{(2.16h)} && \sum_{i=1}^{k} z_i &&= 1 && \\
&\text{(2.16i)} && \lambda_i &&\in [0,1] && (\ \forall i \in \{1, \ldots, k+1\}\ ) \\
&\text{(2.16j)} && x, y &&\in \mathbb{R} && \\
&\text{(2.16k)} && z_i &&\in \{0,1\} && (\ \forall i \in \{1, \ldots, k\}\ )
\end{aligned}
$$

We have to make sure that only breakpoints adjacent to the active segment may be used for the convex combination. That means that at most two λ-variables may be nonzero, and those have to be adjacent. This is known as λ forming a *Special Ordered Set Of Type II (SOS2)*. The SOS2 condition can be enforced by a special branching scheme as proposed in [BT70]. However, it can also be modeled by mixed-integer constraints; in CCM this is traditionally implemented by (2.16d) to (2.16f).

However, this basic version of CCM is not locally ideal. Instead, a locally-ideal improved variant is proposed in [Pad00]. It asks for replacing (2.16d) to (2.16f) by the constraints

$$
\begin{aligned}
\sum_{i=1}^{j} \lambda_i &\leq \sum_{i=1}^{j} z_i && (\ \forall j \in \{1, \ldots, k\}\ ) \\
\sum_{i=j}^{k} \lambda_{i+1} &\leq \sum_{i=j}^{k} z_i && (\ \forall j \in \{1, \ldots, k\}\ )
\end{aligned}
$$

in order to model the SOS2 condition.

The Convex Combination Method and the Multiple Choice Method both share the same logic with respect to the meaning of the binary variables: namely, there is a binary variable for every segment and the active segment is indicated by its corresponding binary variable having a value of 1, while all others have a value of 0. The main part in Chapter 4 is designed to fit this setting, though we will also discuss transferability of the results to another important formulation method in Section 4.2.3.

2.2.3 The Incremental Method

In contrast to MCM and CCM, the *Incremental Method* (or *δ-Method*) [MM57] uses a different logic to encode the active segment. It introduces continuous δ-variables in addition to the usual binary variables—again one for each segment—that start from the leftmost breakpoint and 'fill up' from there until a particular point x is reached.

$$\text{find} \quad x, y, \delta, z \tag{2.18a}$$

$$x = B_1 + \sum_{i=1}^{k} (B_{i+1} - B_i)\delta_i \tag{2.18b}$$

$$y = \phi(B_1) + \sum_{i=1}^{k} (\phi(B_{i+1}) - \phi(B_i))\delta_i \tag{2.18c}$$

$$\delta_i \leq z_i \qquad (\ \forall i \in \{1, \dots, k\}\ ) \tag{2.18d}$$

$$z_{i+1} \leq \delta_i \qquad (\ \forall i \in \{1, \dots, k-1\}\ ) \tag{2.18e}$$

$$\sum_{i=1}^{k} z_i = 1 \tag{2.18f}$$

$$\delta_i \in [0, 1] \qquad (\ \forall i \in \{1, \dots, k\}\ ) \tag{2.18g}$$

$$x, y \in \mathbb{R} \tag{2.18h}$$

$$z_i \in \{0, 1\} \qquad (\ \forall i \in \{1, \dots, k\}\ ) \tag{2.18i}$$

By construction, the z-variables are decreasing, where a 'jump' $z_i = 1, z_{i+1} = 0$ means that x lies in the i-th segment $[B_i, B_{i+1}]$. In this formulation, we use normalized variables δ_i that indicate the fraction of segment i that is on the left side of x. Constraints (2.18d) and (2.18e) represent the so-called *filling condition*. Together they control that δ_i may only start filling, i.e. have a nonzero value, if the previous segment is already filled completely, i.e. $\delta_{i-1} = 1$ or $i = 1$. Finally, x and y are determined using (2.18b) and (2.18c), respectively, which both form telescope sums. The Incremental Method also yields locally ideal formulations.

In Chapter 4, we will consider the projection of polyhedra related to modeling piecewise linear functions to the integer variables. In this regard, polyhedra obtained from a modeling according to MCM and CCM agree, as they use the same logic with respect to the integer variables. Concerning the Incremental Method, there is a well-known linear bijection T that establishes a connection between its binary variables and those of the other two methods. It is given by

$$T : P_\delta \to P_{\text{MCM}}, \quad T(z)_i = \begin{cases} z_i - z_{i+1}, & i \in \{1, \dots, n-1\} \\ z_n, & i = n \end{cases} .$$

with inverse

$$T^{-1} : P_{\text{MCM}} \to P_\delta, \quad T^{-1}(z)_i = \sum_{j=i}^{n} z_j,$$

where P_{MCM} denotes the polytope from the formulation according to the Multiple Choice Method (or the Convex Combination Method), and P_δ the polytope corresponding to the incremental formulation, cf. [Vie15].

The existence of such a transformation implies a one-to-one correspondence between the extreme points of P_{MCM} and P_δ. Also, following [Vie15], we obtain a complete description of P_δ by taking a complete description of P_{MCM} and for each inequality, replacing every occurrence of a z-variable by $T(z)$.

We will use this connection in Chapter 4, Section 4.2.3, to translate polyhedral results to a formulation based on the Incremental Method. Moreover, in Chapter 5 the transformation T will play an important role to turn a formulation into an equivalent dual flow problem (see (2.5))—although the formulation did not originate from the context of modeling piecewise linear functions.

Although all of the above methods lead to locally ideal formulations (except the basic version of CCM) and there being a close connection between the binary variables of those formulations, their performance in practice may vary significantly, depending on the application. The Incremental Method has proven very useful for certain applications and is widely used in practice, e.g. for optimization problems on gas networks [Gei11b]. A recent in-depth computational study for piecewise linear functions in the context of gas network optimization [CPSM14], sees the Incremental Method coming out on top, outperforming the Multiple Choice Method even by several orders of magnitude for some test sets. One possible explanation for the success of this method is that it leads to more balanced branch-and-bound trees. A clear performance gap is also observed in the computations in Chapters 4 and 5, where it turns out that the facets of the respective polyhedra are overall sparser when described in terms of the Incremental Method.

2.2.4 Logarithmic Models

One further method—or rather, technique to modify other linearization methods—should be mentioned, namely *logarithmic modeling*. All methods so far use binary variables to somehow indicate the active segment, and have one such binary variable for each segment. Those formulations involve redundancy in the sense that knowing the value of one binary variable can allow to derive the value of a lot of other variables. In contrast, as one can distinguish 2^n cases by using n binary variables, it should be possible to have a formulation that uses much fewer binary variables than the formulations above. Consequently, formulations using only a logarithmic number of variables have been proposed [VN11]. For example, there is a logarithmic version for CCM. The idea behind it is to encode the active segment using a *Gray Code*. It encodes every segment

$i = 1, \dots, k$ by a binary code $c(i)$ such that the code for adjacent segments only differs by one digit. This allows to force the SOS2 condition with logarithmically many variables $z_1, \dots, z_{\lceil \log_2(k) \rceil}$ directly by the following constraints:

$$\sum_{i=1}^{j-2} \lambda_i + \sum_{i=j+1}^{k} \lambda_i \quad \leq \quad \sum_{\{l | c(j)_l = 1\}} (1 - z_l) + \sum_{\{l | c(j)_l = 0\}} z_l \qquad (\ \forall j \in \{1, \dots, k\} \ ),$$

cf. [GMMS12]. For a more general presentation of a logarithmic convex combination model see [VAN10].

Despite having drastically fewer variables, logarithmic models are often times not computationally superior to methods with linearly many binary variables. In particular, for solving gas network optimization problems, [Gei11b, Chapter 8] and [CPSM14] came to the conclusion that the Incremental Method is preferable.

2.2.5 Nonseparable Multivariate Functions

Based on the modeling techniques for piecewise linear univariate functions presented in the previous section, there are also models for nonseparable multivariate functions $\phi : D \to \mathbb{R}$. In this context, it is usually assumed that the domain D is compact, and that there is a triangulation of D into simplices (or more general polytopes) such that ϕ is linear on each simplex. Furthermore, ϕ should be continuous, though—as for the univariate case—there exist extensions to so-called semi-continuous functions.

With simplices taking the role of segments, MCM is relatively straightforward to extend, while still leading to locally ideal formulations. CCM can also be extended, though one has to be careful in order to obtain a locally ideal formulation (the so-called *disaggregated convex combination model*). Extending the Incremental Method is relatively elaborate as one has to determine an ordering of the simplices that has to satisfy some consistency condition—which is trivially satisfied for the canonical ordering in dimension 1. However, finding such a consistent ordering is possible in arbitrary dimension, a detailed presentation can be found in [GMMS12]. For more details and exact model formulations (for polytopes instead of simplices) see [VAN10].

2.2.6 Piecewise Linear Approximations and Relaxations

For constructing a *piecewise linear approximation* of a univariate function f, one selects breakpoints B_i, $i = 1, \dots, k+1$ and replaces f by the unique piecewise linear function ϕ that agrees with f on all breakpoints, i.e. $\phi(B_i) = f(B_i)$ for all $i = 1, \dots, k+1$. In case f is multivariate, ϕ is not uniquely determined by this condition. Hence, in addition one has to fix a triangulation of the selected breakpoints into simplices. On each simplex S, the restriction $\phi|_S$ is then uniquely determined as the linear function that agrees with f on all the vertices of S. The piecewise linear approximation function ϕ can be modeled by one of the modeling methods reviewed above, thus removing the nonlinearity f from the problem formulation.

If one wants to obtain a *piecewise linear relaxation* of the original problem formulation rather than an approximation, one may add or subtract a sufficiently large error term $\epsilon > 0$ (that may depend on the segment) to f's piecewise linear approximation ϕ such that the resulting area contains the function graph of f, i.e. $\phi(x) - \epsilon \leq f(x) \leq \phi(x) + \epsilon$ for all $x \in D$. In the resulting MIP formulation, the variable y representing the function value $f(x)$ in Formulations (2.15), (2.16) and (2.18) is relaxed to satisfying

$$y \in [\phi(x) - \epsilon, \phi(x) + \epsilon]. \tag{2.20}$$

In contrast, piecewise linear approximation requires $y = \phi(x)$ (see Figure 2.1). Condition (2.20) leads to a proper relaxation of the original problem which comes with the usual advantages, e.g. its optimal value provides a dual bound for the original problem.

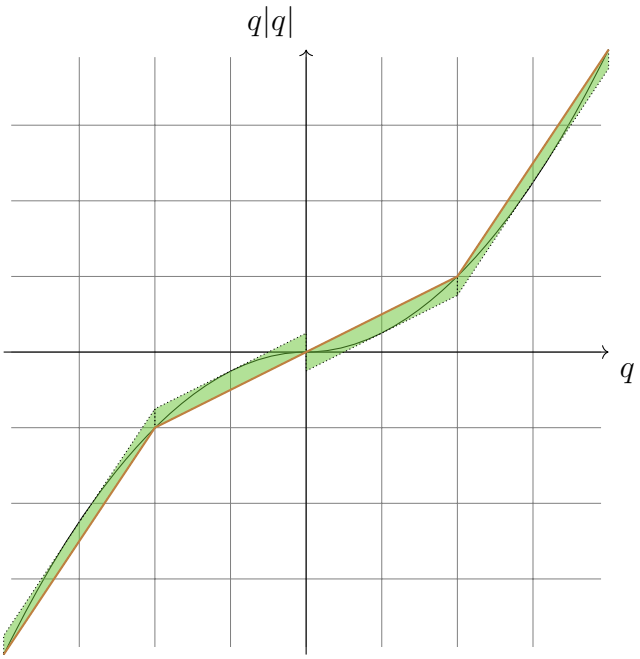


Figure 2.1: Piecewise linear approximation (brown function) and relaxation (green area) of the function (2.14) on the interval $[-4, 4]$.

In order to determine suitable values for ϵ, one has to be able to compute the maximum overestimation and the maximum underestimation of f by ϕ, i.e. $\max_{x \in S}(f(x) - \phi(x))$ and $\max_{x \in S}(\phi(x) - f(x))$, respectively, for every segment (or simplex) S. As f is nonlinear, this task already is hard in general, but can be computed efficiently for certain classes of nonlinear functions, including (2.14). The density of breakpoints may be chosen such that an a-priori defined ϵ can be used. Condition (2.20) can be directly incorporated into all common piecewise linear modeling methods (see [Gei11b, Section 4.4]).

This is also possible in arbitrary dimension with a suitable extension of piecewise linear modeling to the multivariate case. However, one should note that this approach

becomes less attractive in higher dimensions as the number of simplices needed for a fine-grained triangulation—e.g. such that the diameter of each simplex is below a given threshold—grows exponentially in the dimension.

For a detailed description of using piecewise linear relaxations for solving non-convex MINLPs, together with computational results, see [GMMS12]. It is one of the methods connected with recent advances on the so called nomination validation problem on gas networks, which can nowadays be solved satisfactorily for country-size real-world gas networks [PFG+15].

Chapter 3

Solving Network Expansion Problems by Iterative Graph Aggregation

Mathematical network optimization tasks often times have to be solved for very large underlying networks resulting from country size instances or a high level of detail for the network's representation. In case of NP-hard problems as e.g. the network design problem (see Subsection 2.1.3), this poses challenges to state-of-the-art solution approaches to the extend that some practical instances cannot be solved with the techniques that are currently available within an acceptable time frame. A question that arises naturally is whether the problem sizes can be reduced in practice. In this chapter, we focus on aggregation methods. Major parts of the chapter are based on joint work with Andreas Bärmann, Frauke Liers, Alexander Martin, Christoph Thurner and Dieter Weninger, published in [BLM+15].

3.1 What is Aggregation?

The term *aggregation* exists in many disciplines and has been used in many different contexts in the literature. Compliant with the very general idea of 'A whole formed by combining several separate elements'(given by *Oxford Dictionary* [Oxf] for the general noun *aggregate*), we use the term *aggregation* to describe a coarsening process that condenses data and omits details. Important reasons for performing aggregation on problem data include that the problem might simply be too large for solving it accurately in its full size, or that expensive computations are not reasonable for part of the data, e.g. because they will likely not matter. Aggregation techniques typically combine parts of the original problem to obtain an aggregated problem and solve this aggregated instance. Though aggregation is meant to ensure a global view on the complete problem, the resulting method represents a heuristic one, if no estimation of the error due to aggregation is available. However, in this chapter we will only consider algorithms that still solve the original problem to optimality. Such methods typically involve disaggregation. This can be seen as an inverse procedure that reintroduces more detailed information, leading to a modified aggregated problem. The process is

iterated until some the stopping criteria are satisfied.

Aggregation is naturally part of intuitive human decision making in complicated scenarios. As an example, when planning to travel to an overseas location, naturally the first thing to look for is a suitable flight connection. Only in a second step, one checks how to complete the route by connections to and from airports—often even after the flight has been booked. The reasoning behind this is simple: while reaching nearby airports will most likely not pose major problems regardless of the flight, availability and associated costs (in terms of money and travel time) of flight connections depend heavily on the time of departure and are considered 'critical' for the connection.

Aggregation techniques have frequently been investigated. In [Bal65], a solution method for large-scale transportation problems is suggested that does not consider all data simultaneously. Instead, a sequence of aggregated problems is solved while more and more data is reintroduced during the algorithm until the optimal solution is found, which is certified by a duality-based optimality criterion. This is conceptually very similar to the algorithms developed in this chapter and [Bal65] was indeed a main inspiration for this work.

An survey of features that are characteristic for aggregation and disaggregation techniques can be found in [RPWE91], though the authors quote from [Iji71]:

> ...it is difficult to systematize various aggregation issues that have been raised in the literature according to subject matter unless the nature of the subject is considerably limited ...

Reference [Zip77] derives a posteriori and a priori bounds for the linear programming case. A survey on aggregation techniques has been given in [DRV87] and there is a book about aggregation in the context of large-scale optimization [LT03]. In addition, aggregation techniques are applied to a wide field of applications, including the optimization of production planning systems [Lei95, Lei98], and gas network optimization [RMWSB02]. Various articles on aggregation are surveyed in [Fra85]. There are only few results about the usability of aggregation techniques in discrete problem settings. In [Ros74], aggregation of equations is described, [CH77] analyzes aggregation of inequalities and [HS90] presents column aggregation in integer programming. Aggregation has proven useful for handling highly symmetric problems. In [LMT09], it is one of several tools to grind a very hard problem instance from coding theory; [SW12] uses aggregation to form a master problem of a decomposition method for multi-activity shift scheduling. Especially, shortest path algorithms based on graph contractions [Gei11a] are very successful in practice. Recent examples for the use of aggregation are [MAdC+11] for a vehicle routing application with time windows and [NK07] for scheduling the excavation at an underground mine.

In practical applications, it is very common that a relatively developed network has to be upgraded in order to allow for the routing of additional demand requirements. In such a network expansion problem, typically only a small percentage of the arcs

has to be upgraded. These arcs are frequently referred to as *bottlenecks*. Bottleneck arcs constitute the limiting factor for additional demands to be routed on top of those that can already be accommodated. As an example application, in its initial state in 2010, the German railway network was able to accommodate about 80 % of the forecasted demand for the year 2030. An appropriate upgrade requires capacity-increasing measures on less than 20 % of the tracks. This fact is a motivation to devise an algorithm that continuously updates a set of potential bottleneck arcs, whereas non-bottleneck arcs are aggregated.

3.2 The Single-Commodity Network Expansion Problem

In the following, we will develop an exact approach based on iterative aggregation for solving single-commodity network expansion problems. This type of problem has been introduced in Subsection 2.1.3, where the term *expansion* refers to the fact that we assume that we already have a relatively developed network that has to be upgraded.

We will work on the mixed-integer programming formulation (2.8), restated here for convenience:

$$
\begin{aligned}
(3.1) \qquad \min \quad & \sum_{a \in A} k_a y_a & & & \\
\text{s.t.} \quad & \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a & = & \; d_v & \quad (\ \forall v \in V\ ) \\
& x_a & \leq & \; c_a + C_a y_a & \quad (\ \forall a \in A\ ) \\
& x & \in & \; \mathbb{R}_+^{|A|} & \\
& y & \in & \; \mathbb{Z}_+^{|A|}. &
\end{aligned}
$$

We assume that the underlying graph $G = (V, A)$ is bidirected. Each arc $a \in A$ possesses initial arc capacities $c_a \geq 0$. In addition, it can be upgraded by installing a module with an upgrade capacity of C_a at a price of k_a per unit, available in integral multiples y_a. The flow on arc a is represented by the variable x_a. The aim is to determine a feasible routing of a specified demand vector $d \in \mathbb{R}^{|V|}$ that incurs a minimal cost upgrade of the network while respecting the capacities of the arcs. Although the method is not restricted to single-commodity network design problems, we focus on this case in the following for ease of exposition.

The algorithmic idea presented in the following can be extended to more complex problems as e.g. the problems discussed in Subsection 2.1.4. In some cases, for instance for multi-commodity flow problems, this can be done in a very straightforward way to obtain a first raw version of an aggregation algorithm, though there might be room (and need) for improvement by incorporating problem specific additional enhancements. We will sketch some ideas for extending our algorithms in Sections 3.6 and 3.7.

3.3 An Iterative Graph Aggregation Scheme

The proposed aggregation scheme for solving (3.1) works as follows:

1. Choose an initial aggregation by partitioning the node set of the graph into components.
2. Solve the network expansion *master problem* over the aggregated graph.
3. Solve *subproblems*, to check whether the optimal solution of the master problem can be extended to a solution on the original graph;
   (a) In case of feasibility: terminate and return a network expansion.
   (b) In case of infeasibility: refine the partition and go to Step 2.

The above procedure can be seen as a method that computes a reduced version of the network consisting of bottlenecks only. We will prove in Subsection 3.3.3 that at termination, the returned solution is globally optimal for the original network, which makes the devised method an exact algorithm for solving network expansion problems. The algorithm is detailed in the following.

3.3.1 Graph Aggregation and the Aggregated Master Problem

The quite intuitive concept of graph aggregation can be formalized as follows:

Definition 3.1 (Graph Aggregation)**.** Let $G = (V, A)$ be a directed graph and $\varphi : V \to \{1, \dots, k\}$ be a surjective *clustering function* for some positive integer k. Then the graph obtained by *graph aggregation* of G with respect to φ, denoted by $\mathcal{G}_\varphi = (\mathcal{V}_\varphi, \mathcal{A}_\varphi)$, has the node set

$$\mathcal{V}_\varphi = \{V_1, \dots, V_k\}, \text{where } V_i = \varphi^{-1}(i) \subseteq V,\ i \in \{1, \dots, k\}.$$

The sets $V_i,\ i = 1, \dots, k$ will be referred to as *(aggregate) components*. The arc set $\mathcal{A}_\varphi$ of $\mathcal{G}_\varphi$ is defined by

$$\mathcal{A}_\varphi = \{(V_{\varphi(u)}, V_{\varphi(v)}) \mid a = (u, v) \in A \text{ with } \varphi(u) \neq \varphi(v)\},$$

i.e. $\mathcal{A}_\varphi$ can be interpreted as a subset of A. To put it differently, $\mathcal{G}_\varphi$ results from contracting all edges $a = (u, v) \in A$ with $\varphi(u) = \varphi(v)$.

Note that G as well as $\mathcal{G}_\varphi$ are allowed to contain multiple arcs between the same two nodes and that parallel arcs may be introduced during aggregation. Figure 3.1 illustrates the above definition.

In order to define an aggregated version of Problem (3.1) on the aggregated graph $\mathcal{G}_\varphi$, we have to specify the demands of aggregate nodes as well as the capacity and upgrading capability of aggregate arcs: The aggregated demand vector d_φ is defined as the total net demand inside a component, i.e.

$$d_{V_i} = \sum_{v \in V_i} d_v,\ i = 1, \dots, k.$$

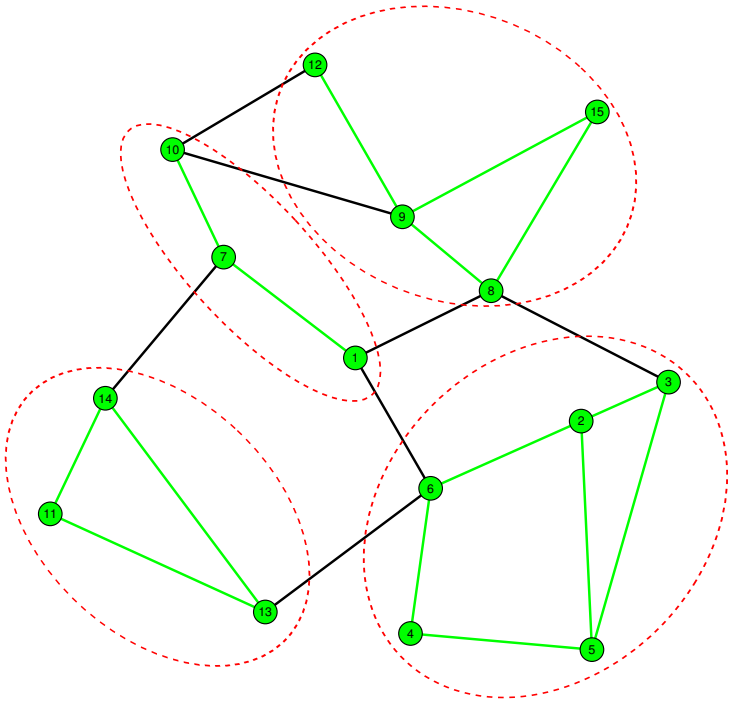


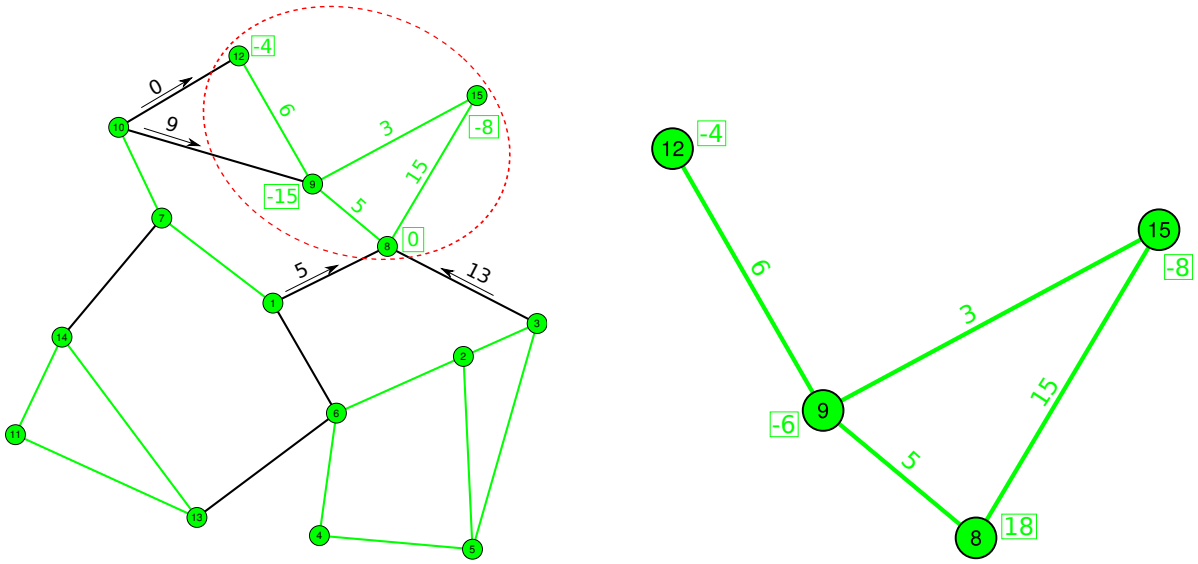
Figure 3.1: Aggregation of a graph with respect to a clustering function φ with $\varphi^{-1}(1) = \{1, 7, 10\}$, $\varphi^{-1}(2) = \{2, 3, 4, 5, 6\}$, $\varphi^{-1}(3) = \{8, 9, 12, 15\}$, and $\varphi^{-1}(4) = \{11, 13, 14\}$. Nodes in the same component are encircled.

The capacity of an arc $a = (V_{\varphi(u)}, V_{\varphi(v)}) \in \mathcal{A}_\varphi$ is simply identical to that of the corresponding original one in A, and also installable upgrades of an arc $a \in \mathcal{A}_\varphi$ are inherited. In order to simplify notation, we identify a component $V_i \in \mathcal{V}_\varphi$ with its index i and identify each arc $a \in \mathcal{A}_\varphi$ with the corresponding original one in A.

The master problem with respect to $\mathcal{G}_\varphi$ can then be stated as follows:

$$
\begin{aligned}
(3.2) \qquad \min \quad & \sum_{a \in \mathcal{A}_\varphi} k_a y_a \\
\text{s.t.} \quad & \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a = d_v && (\ \forall v \in \mathcal{V}_\varphi\ ) \\
& x_a \leq c_a + C_a y_a && (\ \forall a \in \mathcal{A}_\varphi\ ) \\
& x \in \mathbb{R}_+^{|\mathcal{A}_\varphi|} \\
& y \in \mathbb{Z}_+^{|\mathcal{A}_\varphi|}.
\end{aligned}
$$

Note that by definition of the aggregated demand vector, the flow conservation constraint at an aggregate node $i \in \mathcal{V}_\varphi$ is exactly the sum of the original flow conservation constraints of all nodes $v \in V_i$. As a result, any solution of (3.1) is also a solution of (3.2) with the same objective function value, and hence (3.2) is a relaxation of (3.1). Consequently, the optimal value of (3.2) with respect to an arbitrary clustering function



(a) The induced demands for component $\{8, 9, 12, 15\}$.

(b) The associated feasibility subproblem for component $\{8, 9, 12, 15\}$.

Figure 3.2: Illustration of a subproblem of the aggregation scheme: Figure 3.2a shows part of the solution of the master problem which sends flow into component $\{8, 9, 12, 15\}$ (encircled) via several arcs. The corresponding subproblem for this component is depicted in Figure 3.2b.

gives a lower bound on the optimal value of the original problem.

3.3.2 The Local Subproblems and Graph Disaggregation

The solution of (3.2) induces new demands within the aggregated components. We define subproblems whose purpose is to validate whether these demands can be routed without additional capacity upgrades inside the components. Moreover, in case of a negative answer, we obtain information on where to refine the representation of the graph in a subsequent master problem.

Let $H_i = (V_i, A_i)$ be the subgraph of $G = (V, A)$ induced by component V_i of the partition of V according to φ, i.e. $V_i = \{v \in V \mid \varphi(v) = i\},\ i = 1, \ldots, k$ and A_i is the restriction $A_i = A|_{V_i \times V_i}$.

Checking extendibility of an optimal solution of (3.2) can be done by solving a maximum-flow problem within each component as follows. The nodes V_i of H_i have an original demand of $d_v,\ v \in V_i$. The optimal flows obtained from the master problem induce new demands within H_i as each flow x_a on an arc $a = (u, v) \in \mathcal{A}_\varphi$, where $u \in V_i$ and $v \in V_j$, changes the demand of u to $\tilde{d}_u := d_u + x_a$ and that of v to $\tilde{d}_v := d_v - x_a$.

An example of this situation is depicted in Figure 3.2.

The resulting problem for component i is the following basic network flow feasibi-

lity problem (cf. (2.1)):

$$
\begin{aligned}
(3.3) \qquad & \text{find} \quad x \\
& \text{s.t.} \quad \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a \;=\; \tilde{d}_v \qquad (\ \forall v \in V_i\ ) \\
& \qquad\qquad x_a \;\leq\; c_a + C_a \tilde{y}_a \qquad (\ \forall a \in A_i\ ) \\
& \qquad\qquad x \;\in\; \mathbb{R}_+^{|A_i|},
\end{aligned}
$$

where $\tilde{y}$ as the network design solution determined by the master problem is fixed and $\tilde{d}_v$ denotes the adapted demands as described above.

As explained in Subsection 2.1.2, we can formulate such a problem as a maximum-flow problem with a single source and a single sink. This reformulation is helpful for determining a disaggregation in case of infeasibility.

If the subproblem for a component V_i is feasible, the component requires no further examination in the current iteration. The algorithm terminates as soon as all subproblems are feasible. In contrast, an infeasible subproblem indicates that the master problem mistakenly neglected the capacity limitations within the component. When an infeasible subproblem is encountered, the partition is refined in order to consider additional arcs in the master problem. This arc set is chosen as a minimum cut that limits the flow. Details of this have been covered in Subsection 2.1.2. Note that we assume G to be connected and bidirected and that the disaggregation will be done in a way that ensures that also all components are strongly connected. Therefore, Proposition 2.1 guarantees that this cut always contains edges of H such that we get a directive on where to refine the representation. An example illustration of the disaggregation step is shown in Figure 3.3a.

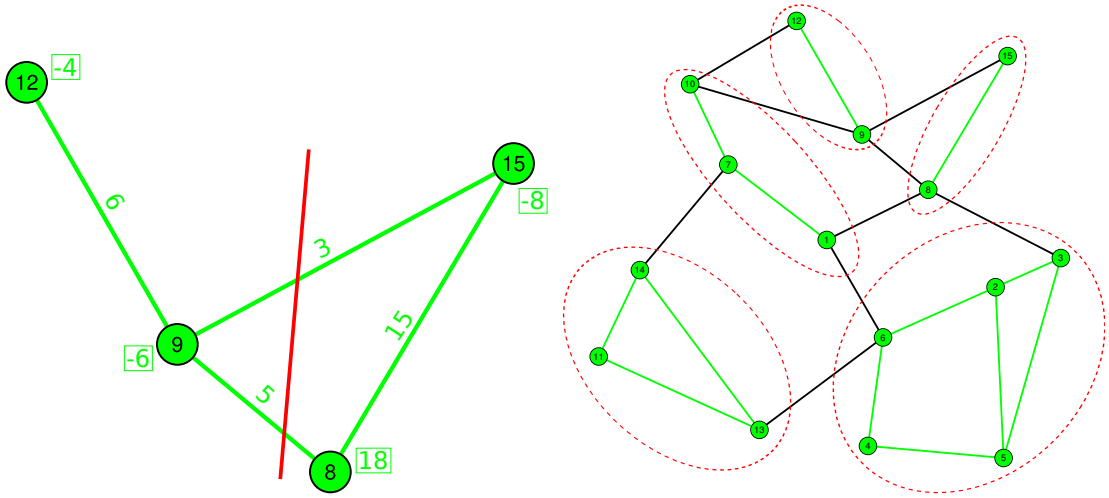
Updating the master problem is done by disaggregating the infeasible component along this minimum cut. Let without loss of generality V_k be an infeasible component and let $V_k^1, \ldots, V_k^l$ be the components into which V_k disaggregates. We define a new clustering function $\overline{\varphi} : V \to \{1, \ldots, k+l-1\}$ with $\overline{\varphi}(v) = \varphi(v)$ if $v \notin V_k$ and $\overline{\varphi}(v) = k+i$ if $v \in V_k^i \subset V_k$, see Figure 3.3b. The resulting refined aggregated master problem therefore is a relaxation not only of the original problem but also of all aggregated master problems of previous iterations.

3.3.3 Correctness of the Algorithm

We show next that for nonnegative expansion costs, the above method always terminates with an optimal solution to the original network expansion instance.

Theorem 3.2 (Correctness of the Algorithm)**.** *For nonnegative expansion costs k in* (3.1)*, the proposed algorithmic scheme always terminates after a finite number of iterations with an optimal solution to the network expansion problem for the original graph.*

Proof: Termination follows from the fact that only finitely many disaggregation steps are possible until the original graph is reached. Clearly, the returned solution is feasible for the original network by the termination criterion.



(a) Disaggregation along a minimal cut (b) Resulting graph after disaggregation

Figure 3.3: Disaggregation of a component in case its subproblem is infeasible.

In order to prove optimality, let $(\tilde{x}, \tilde{y})$ be the optimum solution of the final master problem on $\mathcal{G}_\varphi$ with objective value $\tilde{z} = \sum_{a \in A} k_a \tilde{y}_a$. As (3.2) is a relaxation of (3.1) for nonnegative expansion costs k, we know that $\tilde{z}$ gives a lower bound $\tilde{z} \leq z^*$ on the optimum objective value z^* of the original problem. On the other hand $(\tilde{x}, \tilde{y})$ has been successfully extended by the subproblems to a feasible solution of (3.1) with the same objective value $\tilde{z}$ as no additional costs have been caused by this extension. Therefore, we also have $z^* \leq \tilde{z}$, which concludes the proof. □

3.3.4 Relation to Benders Decomposition

The aggregation procedure developed in this chapter possesses some obvious similarities to Benders decomposition. Both algorithms solve a succession of increasingly stronger relaxations of the original problem, which is achieved by introducing cutting planes. In the case of the aggregation framework, these are part of the primal (aggregated) flow conservation and capacity constraints. For Benders decomposition, these are the Benders feasibility and optimality cuts. Both algorithms stop as soon as the optimality of the relaxed solution is proven. Furthermore, the subproblem used in the aggregation approach coincides with the subproblem in Benders decomposition if the x-variables for the arcs in $\mathcal{A}_\varphi$ are chosen to belong to the Benders master problem.

However, there are also substantial differences. The aggregation scheme introduces both new variables and constraints in each iteration to tighten the master problem formulation. Contrary to this, Benders decomposition is a pure row-generation scheme. Equally important is the fact that the continuous disaggregation of the network graph leads to a shift in the proportions between the master and the subproblem. The master problem grows in size, while the subproblem tends to shrink as bottleneck

arcs are transferred from inside the components to the master graph. In comparison, Benders decomposition leaves these proportions fixed.

The following theorem details the relation between the subproblem information used in the two algorithms.

Theorem 3.3. *Let φ be a clustering function according to a given network graph G. For a disaggregation of G along a minimal cut, the primal constraints introduced to the master problem* (3.2) *in the proposed aggregation scheme strictly imply the Benders feasibility cut obtained from the corresponding subproblem.*

Proof: We prove the claim for the special case where the whole graph is aggregated to a single component, i.e. $\varphi \equiv 1$. The corresponding situation in Benders decomposition is that all arc flow variables are projected out of the master problem. The extension of the arguments to the general case is straightforward.

For both algorithms, the subproblem consists in finding a feasible flow in the network and thus a solution to the following feasibility problem of type (3.3).

Let π denote the dual variables of the flow conservation constraints and ω those of the capacity constraints. In case of infeasibility, Benders decomposition derives its feasibility cut from an unbounded ray of the dual subproblem

$$
\begin{aligned}
(3.4) \qquad \max \quad & \sum_{v \in V} d_v \pi_v - \sum_{a \in A} (c_a + C_a \tilde{y}_a) \omega_a \\
\text{s.t.} \quad & \pi_u - \pi_v - \omega_a \leq 0 \qquad (\ \forall a = (u,v) \in A\ ) \\
& \omega \in \mathbb{R}_+^{|A|}.
\end{aligned}
$$

where $\tilde{y}$ denotes the network design solution determined by the master problem. As already mentioned in Subsection 2.1.2, variables ω can be eliminated from the problem since any feasible solution $(\hat{\pi}, \hat{\omega})$ is dominated by the solution $(\tilde{\pi}, \tilde{\omega})$ with $\tilde{\pi} = \hat{\pi}$ and

$$\tilde{\omega}_a = \max\{\hat{\pi}_u - \hat{\pi}_v, 0\}$$

for $a = (u, v) \in A$. In case of an infeasible (primal) subproblem, the Benders cut can now be written as

$$\sum_{v \in V} \tilde{\pi}_v d_v \leq \sum_{a \in A} \max\{\tilde{\pi}_u - \tilde{\pi}_v, 0\}(c_a + C_a y_a),$$

for $\tilde{\pi}$ belonging to an unbounded dual ray $(\tilde{\pi}, \tilde{\omega})$. For the same infeasible master solution, the aggregation scheme adds the following system of inequalities to its master problem:

$$\sum_{a \in \delta^+_{V_i}} x_a - \sum_{a \in \delta^-_{V_i}} x_a = d_i \quad (\forall V_i \in \mathcal{V}_{\overline{\varphi}})$$

and

$$x_a \leq c_a + C_a y_a \quad (\forall a \in \mathcal{A}_{\overline{\varphi}} \setminus \mathcal{A}_{\varphi})$$

together with the new variables x_a for $a \in \mathcal{A}_{\overline{\varphi}} \setminus \mathcal{A}_\varphi$, where $\overline{\varphi}$ is the aggregation induced by the minimal cut. As stated above, the dual subproblem values $\tilde{\pi}_u$ and $\tilde{\pi}_v$ coincide if nodes u and v belong to the same component $V_i \in \mathcal{V}_\varphi$. Thus we can define $\pi_i := \tilde{\pi}_u$ for some $u \in V_i$. The claim then follows by taking the sum of the aggregated flow conservation constraints weighted with $-\tilde{\pi}_i$ and the aggregated capacity constraints weighted with $-\max\{\tilde{\pi_u} - \tilde{\pi}_v, 0\}$. For this weighting, the left hand side becomes

$$\sum_{V_i \in \mathcal{V}_\varphi} \tilde{\pi}_i \left(\sum_{a \in \delta^+_{V_i}} x_a - \sum_{a \in \delta^-_{V_i}} x_a \right) - \sum_{a=(u,v) \in \mathcal{A}_\varphi} \max\{\tilde{\pi}_u - \tilde{\pi}_v, 0\} x_a,$$

which can be transformed to

$$\sum_{a=(u,v) \in \mathcal{A}_\varphi} (\tilde{\pi}_u - \tilde{\pi}_v - \max\{\tilde{\pi}_u - \tilde{\pi}_v, 0\}) x_a.$$

This yields

$$\sum_{\substack{a=(u,v) \in \mathcal{A}_\varphi: \\ \tilde{\pi}_v \geq \tilde{\pi}_u}} (\tilde{\pi}_u - \tilde{\pi}_v) x_a,$$

which is nonpositive due to the nonnegativity of x. Thus, we find

$$\sum_{v \in V} \tilde{\pi}_v d_v = \sum_{V_i \in \mathcal{V}_\varphi} \tilde{\pi}_i d_i \leq \sum_{\substack{a=(u,v) \in \mathcal{A}_\varphi: \\ \tilde{\pi}_v \geq \tilde{\pi}_u}} (\tilde{\pi}_u - \tilde{\pi}_v) x_a + \sum_{a \in A} \max\{\tilde{\pi}_u - \tilde{\pi}_v, 0\}(c_a + C_a y_a),$$

which implies the Benders cut. Finally, this inequality is strict for all solutions to the problem, where flow is sent both along a certain arc as well as along its opposite arc. This completes the proof. □

The theorem above shows that each iteration of the aggregation scheme introduces more information to the master problem than a Benders iteration. Whereas Benders decomposition is often used to solve network design problems, it is widely known that the original Benders cuts are weak and numerically unstable already for small-scale networks. And this becomes even more problematic in the case of large-scale networks as they are considered here. Therefore, Benders decomposition is most commonly employed for smaller network problems with complicating constraints. However, numerical difficulties connected to Benders may remain manageable if not too many of them are needed, e.g. when starting with a very good (heuristic) primal solution. Benders cuts have been used together with aggregation in [Bä16] in order to incorporate routing costs.

3.3.5 The Global Subproblem

It should be mentioned that the current network expansion in terms of the y-variables might be optimal although the extendibility test described in Subsection 3.3.2 fails

for some component. This is because in the subproblems not only the expansions (y-variables) are fixed but also the flow on all edges contained in the master problem. To overcome this problem we can use a simple global test: we fix the expansion and check feasibility of the resulting flow problem on the complete graph, given by

$$
\begin{aligned}
(3.5) \qquad & \text{find} \quad x \\
& \text{s.t.} \quad \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a \;=\; d_v \qquad (\ \forall v \in V\ ) \\
& \qquad\qquad x_a \;\leq\; c_a + C_a \tilde{y}_a \qquad (\ \forall a \in A\ ) \\
& \qquad\qquad x \;\in\; \mathbb{R}_+^{|A|},
\end{aligned}
$$

where $\tilde{y}$ is fixed by the master problem. In case of the single-commodity network design problem, this global subproblem is even capable of completely replacing the local subproblems. For that, it is required that in case of infeasibility of the global subproblem, the minimum cut obtained does contain at least some edge that is not part of the master problem yet (and therefore the cut runs through one or more components). Indeed, we can show that the case in which this cut only consists of master problem edges can be excluded (despite this having been considered possible in [BLM+15]).

Theorem 3.4. *Let $(\tilde{x}, \tilde{y})$ be a feasible solution of* (3.1) *and let* (3.5) *be infeasible. Then any minimum cut restricting the flow in* (3.5) *contains edges that have not been part of* (3.1)*.*

Proof: Consider the dual problems of type (2.5) of the master problem (3.2) as well as for the global subproblem (3.5). We have identified them as minimum cut problems for the single-source versions of the respective flow problems in Chapter 2, Subsection 2.1.2. In the following, we will refer to the construction and notation of the super nodes and super sinks as well as artificial arcs described in that subsection.

The minimum cut problem for the master problem reads

$$
\begin{aligned}
(3.6) \qquad & \min \quad \sum_{a \in \tilde{\mathcal{A}}_\varphi} (c_a + C_a \tilde{y}_a)\omega_a \\
& \text{s.t.} \quad \pi_u - \pi_v + \omega_a \;\geq\; 0 \qquad (\ \forall a = (u,v) \in \tilde{\mathcal{A}}_\varphi\ ) \\
& \qquad\quad \pi_t - \pi_s \;\geq\; 1 \\
& \qquad\qquad\quad \pi_v \;\in\; \mathbb{R} \qquad (\ \forall v \in \mathcal{V}_\varphi \cup \{s,t\}\ ) \\
& \qquad\qquad\quad \omega_a \;\in\; \mathbb{R}_+ \qquad (\ \forall a \in \tilde{\mathcal{A}}_\varphi\ ),
\end{aligned}
$$

where—as a notational reminder—$\tilde{\mathcal{A}}_\varphi$ denotes the set of arcs $\mathcal{A}_\varphi$ together with any artificial arcs that have been introduced in order to obtain a single-source network flow problem.

We know that both (2.4) and its dual (2.5) are feasible for all networks as they allow for trivial feasible solutions. Hence, by strong duality, the optimal value of (3.6) is equal to the total demand to be routed, given by $\sum_{v \in \mathcal{V}_\varphi : d_v > 0} d_v$. However, for forming

aggregate demands, some demand values have canceled out. We compensate for that by adding extra auxiliary arc (s, V_i) from the super source s into the component nodes with a capacity equal to the total demand of that aggregate node minus its net demand. Similarly, arcs (V_i, t) from the components to the super sink t are added that have the same demand as their aforementioned counterparts. This raises the objective value of (3.6) to the total demand $D \coloneqq \sum_{v \in V : d_v > 0} d_v$ of the original problem.

For the global subproblem, the dual of the corresponding single-source flow problem is given by

$$
\begin{aligned}
(3.7) \qquad \min \quad & \sum_{a \in \tilde{A}} (c_a + C_a \tilde{y}_a) \omega_a \\
\text{s.t.} \quad & \pi_u - \pi_v + \omega_a \geq 0 && (\ \forall a = (u,v) \in \tilde{A}\ ) \\
& \pi_t - \pi_s \geq 1 \\
& \pi_v \in \mathbb{R} && (\ \forall v \in V \cup \{s,t\}\ ) \\
& \omega_a \in \mathbb{R}_+ && (\ \forall a \in \tilde{A}\ ).
\end{aligned}
$$

The global subproblem (3.5) is infeasible by assumption, which implies that the optimal value of (3.7) is strictly smaller than D. Now assume that a minimum cut limiting the flow contains no arc in $A \backslash \mathcal{A}_\varphi$. This implies that $\omega_a = 0$ for all $a \in A \backslash \mathcal{A}_\varphi$ by the way the cut is computed (again cf. Subsection 2.1.2). Since G is bidirected, we have $\pi_u = \pi_v$ for all nodes u, v with $(u, v) \in A$. Furthermore, due to connectedness of all components, the node potential is constant over each components. Thus, we may define $\pi_{V_i} = \pi_v$ for any $v \in V_i$, which transforms (3.7) equivalently into (3.6) together with the slight adaptation with respect to its total demand described above. This is a contradiction as by assumption the optimal value of the former problem is less than D, whereas the latter has a value of exactly D. □

Still, it is not clear whether a disaggregation strategy based on the global subproblem is superior. In fact, disaggregating in an overly conservative way may be detrimental to the algorithm as it tends to increase the number of iterations (see also the discussion on 'Disaggregation' in Section 3.4 below). An experiment on that question will be contained in Section 3.5.

3.4 Implementation

Three versions of the aggregation scheme have been implemented. The first one represents what has been described so far. It has the obvious drawback that only very limited information is used when moving from one iteration to the next. In order to partly overcome this, we integrate the aggregation scheme into a branch-and-bound framework. Finally, we study a hybrid of both.

3.4.1 Sequential Aggregation (SAGG)

The *Sequential Aggregation Algorithm* (SAGG) works in a strictly sequential manner: in each iteration, the network expansion master problem is solved to optimality. In case of feasibility of all subproblems, the algorithm terminates. Otherwise, the graph is disaggregated as described in the previous section (according to local or global subproblems), see Figure 3.4a for a schematic example.

For speeding up the first iterations, we employ the following variation: instead of solving the network expansion master problems, we solve their linear programming relaxations only and disaggregate according to the obtained optimal solutions. This is done until the optimum solution of the LP relaxation of the original problem is found. Only then, we solve the master problem as an MIP. Experiments suggest that the savings in runtime compensate for the potentially misleading first disaggregation decisions based on the LP relaxation. Note that the proof of Theorem 3.2 does not require y to be integral.

3.4.2 Integrated Aggregation (IAGG)

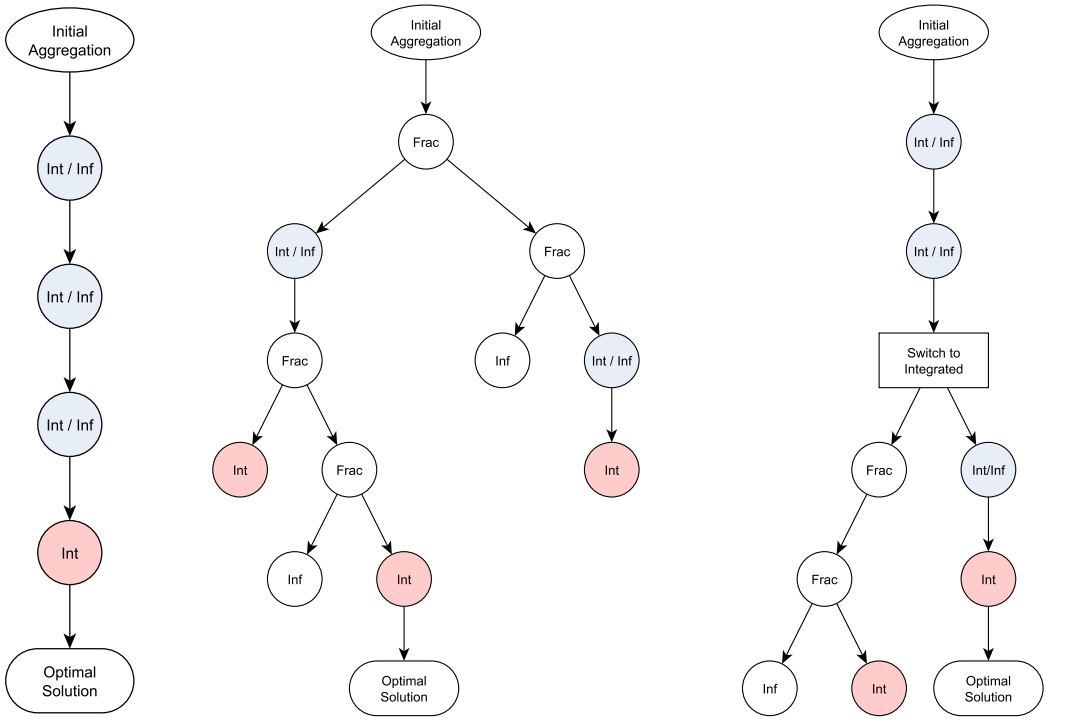
In SAGG, most information, including bounds and cutting planes, is lost when proceeding from one iteration to the next. Only the disaggregation is processed to the next iteration. The idea behind the *Integrated Aggregation Algorithm* (IAGG) is to use more of this information by embedding the disaggregation steps into a branch-and-bound tree.

We start with an initial aggregation and form the corresponding master problem. However, all constraints of (3.2) are formulated using the original variables of Formulation (3.1). Note that aggregation of a network can be performed by removing some capacity constraints and adding up some flow conservation constraints from the formulation of the original instance. Each integral solution (x, y) found during the branch-and-bound search is immediately tested for extendibility (realized by a callback). In case of feasibility, we keep (x, y) as an incumbent solution, otherwise we disaggregate the graph and reject (x, y), see Figure 3.4b for a schematic example. All constraints from an aggregated graph remain valid for the disaggregated network. Disaggregating a network amounts to inserting the flow conservation constraints for the new components and the arc capacity constraints for the arcs entering the master problem to the problem formulation.

3.4.3 The Hybrid Aggregation Algorithm (HAGG)

A natural composition of SAGG and IAGG is the *Hybrid Aggregation Algorithm* (HAGG). It starts with a number of sequential iterations such as in SAGG, and then switches to the integrated scheme (see Figure 3.4c). The idea is to have more information about the graph available at the root node of the branch-and-bound tree once we start to employ the integrated scheme. This is beneficial for the cutting planes generated at the root node.

Thus, for the first iterations, HAGG and SAGG behave exactly the same. Namely, we solve the LP relaxation of the master problem and proceed with the obtained fractional solution. As a heuristic rule, we switch from the sequential to the integrated scheme when the value of the LP relaxation of the master problem is equal to the value of the LP relaxation of the original problem. Then, the optimal fractional expansion is found and the LP relaxation does not give any further disaggregation information.



(a) The Sequential Algorithm (SAGG)

(b) The Integrated Algorithm (IAGG)

(c) The Hybrid Algorithm (HAGG)

Figure 3.4: Schematic outline of the three aggregation schemes. Nodes labeled *Int* correspond to feasible integral solutions of the current master problem, whereas *Int / Inf* indicates that such a solution is infeasible for the original problem, leading to disaggregation. For the branch-and-bound trees, white nodes labeled *Inf* or *Frac* represent infeasible and fractional branch-and-bound nodes (at which branching might occur), respectively.

3.4.4 Details of the Implementation

The Initial Aggregation The easiest choice is to first aggregate the whole graph to a single vertex such that the disaggregation is completely determined by the minimum-cut strategy. As discussed above, this might not be the most suitable choice for the

integrated scheme, which is the motivation for a hybrid scheme. In fact, one can view HAGG as being IAGG with a special heuristic for finding the initial aggregation.

Solving the Subproblems As already mentioned in Chapter 2, all maximum-flow subproblems are solved by a standard LP solver, which is fast in practice. A potential speedup by using specialized maximum-flow implementations is negligible as the implementation spends almost all of the time on solving the master problems.

Disaggregation We have seen how disaggregation works for a single component in Section 3.3. However, in case of several infeasible components, it is not clear beforehand whether all of them should be disaggregated, or, otherwise, which one(s) should be used for disaggregation. In our implementation, we always disaggregate all infeasible components, which aims at minimizing the number of iterations. Experiments with other disaggregation policies did not lead to significant improvement. In addition, we also split components that are not arc-connected into their connected components. This is needed in order to guarantee that we obtain useful cuts in case the subproblem is infeasible (cf. Subsection 3.3.2 above). Moreover, disconnected components are likely to be split anyway in a later iteration.

Global Subproblems The case where testing for extendibility using the global subproblem saves at least one iteration happened regularly in preliminary experiments. As an iteration is relatively expensive (especially for SAGG), using global subproblems for testing should definitely be included in the default settings. After finding a solution of the master problem, we first apply the global subproblem and in case of infeasibility we use the local subproblems to determine how to disaggregate. However, note that due to Theorem 3.4 we could waive the local subproblems completely. This policy in general would lead to more conservative disaggregation. An experiment on this version is included in Subsection 3.5.3. Having a distribution of roles between different types of subproblems adds more flexibility to the scheme. In this case, the purpose of the global subproblem lies solely in the extendibility test, while a possible diaggregation is decided on the basis of a local subproblem. Although this choice is arbitrary for the case of single-commodity network design, we will see this distribution of roles later in Section 3.7, where it is difficult to do without.

3.5 Computational Results

The computational experiments have been performed on a queuing cluster of Intel Xeon E5410 2.33 GHz computers with 12 MB cache and 32 GB RAM, running Linux in 64 bit mode. The framework has been implemented using the C++-API of Gurobi 5.5 [Gur17]. For IAGG and HAGG, it was necessary to adjust Gurobi's parameter settings, which involves a more aggressive cutting plane generation, a focus on improving the bound and downscaling the frequency of the heuristics. Additionally, since

those algorithms use *lazy cuts*, dual reductions had to be disabled in order to guarantee correctness. Implementation SAGG uses Gurobi's standard parameter settings. Each job was run on 4 cores and with a time limit of 10 hours.

We compare our aggregation schemes to the solution of the original network expansion integer program (3.1) using Gurobi 5.5 with standard parameter settings. These reference solution times are denoted by MIP. For MIP, experiments with different parameter settings did not lead to considerably better running times.

3.5.1 Benchmark Instances

The aggregation schemes are tested on different sets of benchmark instances. Network topologies include random scale-free networks according to the preferential attachment model [AB02], and instances created from the *rome99* graph from the 9th DIMACS challenge [DGJ06]. The vector d of demands as well as the vector c of initial arc capacities were drawn randomly. The initial capacities of each instance were scaled by a constant factor in order to obtain different percentages of initial demand satisfaction l, which was done by solving an auxiliary network flow problem. The parameter l indicates which portion of the demand can be routed given the initial state of the network. For different instance sizes, varying the initial capacities has a significant impact on the solution time and the solvability in general, and is therefore an important parameter for the forthcoming analysis.

Whenever the generation of instances included random elements, we generated 5 instances of the same size and demand satisfaction. The solution times then are (geometric) averages over those five instances. If only a subset of the 5 instances was solvable within the time limit, the average is taken over this subset only. We also state the number of instances that could be solved within the time limit.

3.5.2 Computational Results on Scale-Free Networks

The topology of the instances in this benchmark set has been generated according to a preferential attachment model. It produces so-called scale-free graphs [AB02], which are known to represent the evolutionary behavior of complex real networks well. Starting with a small clique of initial nodes, the model iteratively adds new nodes. Each new node is connected to m of the already existing nodes. This parameter m, the so-called *neighborhood parameter*, influences the average node degree. We set $m = 2$ in order to generate sparse graphs that resemble infrastructure networks. In preliminary experimental computations, choosing higher values of m did not influence the results significantly. Furthermore, we chose 80 % of the nodes as terminals, i.e. nodes with non-zero demand, in order to represent a higher but not overly conservative load scenario. The module capacities for these instances were chosen as 0.25 % of the total demand in order to obtain reasonable module sizes with respect to the scale of the demand. Varying these two parameters did not lead to significantly different results either. Finally, the module costs were drawn randomly.

Computational Results for Small Instances

In this subsection, we analyze the aggregation method for small instances with different levels l of initial demand satisfaction on random scale-free networks with 100 nodes, where we consider $l \in \{0, 0.05, 0.1, 0.2, \ldots, 0.8, 0.9, 0.95\}$. First, we determine which implementation of the aggregation scheme performs best. In a second step, we compare the best implementation to MIP.

l	IAGG		SAGG		HAGG	
	solved	time[s]	solved	time[s]	solved	time[s]
0	**3**	**196.10**	3	502.30	3	230.85
0.05	**5**	**22.44**	5	186.98	5	63.04
0.1	**5**	**111.15**	4	676.27	4	106.75
0.2	**5**	**34.57**	5	110.92	5	55.57
0.3	**5**	**8.02**	5	25.69	5	10.16
0.4	**5**	**8.01**	5	35.03	5	8.47
0.5	5	4.18	5	5.65	**5**	**2.71**
0.6	**5**	**5.63**	4	4.23	5	6.07
0.7	5	1.17	**5**	**0.81**	5	0.82
0.8	5	0.57	**5**	**0.49**	5	0.51
0.9	**5**	**0.23**	5	0.30	5	0.25
0.95	**5**	**0.12**	5	0.21	5	0.18

Table 3.1: Number of instances solved and average solution times[s] for the three aggregation algorithms for random scale-free networks with $|V| = 100$ nodes and varying level of initial demand satisfaction.

In Table 3.1, solution times are reported in seconds, each averaged over five instances with the same value of l. If not all instances could be solved within the time limit, the average is taken over the subset of solved instances. If a method could not solve any of the five instances for a given l, we denote this by an average solution time of '∞'. The fastest method in each row is emphasized with bold letters. We rank the methods first by the number of solved instances and second by the average solution time.

The results for the instances from Table 3.1 are also presented as a performance profile in Figure 3.5. For each aggregation method, the percentage of all solved scale-free instances with $|V| = 100$ is shown as a function of the solution time that is given in multiples of the time the fastest method needed in order to solve it. The information deduced from this kind of plot is twofold. First, the intercept of each curve with the left vertical axis shows the percentage of instances for which the corresponding method achieves the shortest solution time. Thus, the method attaining the highest intercept on the vertical axis is the one which 'wins' most instances. Second, for each value m on the horizontal axis, the plot shows the percentage of instances that a method was able to solve within m times the shortest solution time achieved by any of the methods. The

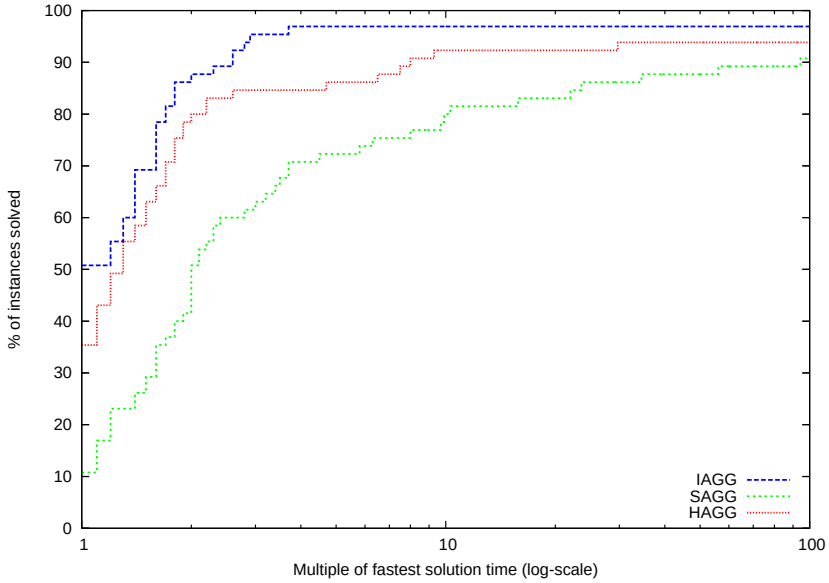


Figure 3.5: Performance profile for the three aggregation frameworks on random scale-free networks with $|V| = 100$ nodes.

interpretation of this information is how good a method is in catching up on instances for which it is not the fastest. A more detailed introduction to performance profiles can be found in [DM02].

We see that IAGG performs best for the scale-free networks with 100 nodes when compared to SAGG and HAGG. It solves the majority of instances within the shortest solution time and solves 97 % of the instances, which is the largest value among the three methods. Furthermore, there is no instance for which it requires more than 4 times the shortest solution time.

In Table 3.2, we thus compare IAGG with MIP. We see that the aggregation approach is beneficial whenever the instance cannot trivially be solved within a few seconds. For instances with small initial demand satisfaction, we observe significantly faster solution times for IAGG, and we see that it is able to solve more instances within the time limit. Even without any preinstalled capacities ($l = 0$), the aggregation scheme attains an average solution time which is 6 times smaller than that of MIP. From $l = 0.7$ upwards, the running times of both algorithms are negligible, and the tiny advantage for MIP can be attributed to the overhead caused by performing the aggregation scheme. The superior performance on instances with small initial demand satisfaction seems surprising. It contrasts the fact that the number of components in the final state of network aggregation in IAGG converges to the number of nodes in the original instance. This is presented in Figure 3.6, where we show the average number of components in the final iteration as a function of the percentage of initial demand satisfaction. The aggregation framework performs better than the standard approach MIP even in case of complete disaggregation. In order to determine what causes this behavior, we tested whether the aggregation approach could determine

l	MIP		IAGG	
	solved	time[s]	solved	time[s]
0	3	1373.11	**3**	**196.10**
0.05	5	266.07	**5**	**22.44**
0.1	4	904.29	**5**	**111.15**
0.2	5	136.43	**5**	**34.57**
0.3	5	29.30	**5**	**8.02**
0.4	5	28.16	**5**	**8.01**
0.5	**5**	**2.40**	5	4.18
0.6	5	19.81	**5**	**5.63**
0.7	**5**	**0.21**	5	1.17
0.8	**5**	**0.08**	5	0.57
0.9	**5**	**0.05**	5	0.23
0.95	**5**	**0.04**	5	0.12

Table 3.2: Number of instances solved and average solution times of MIP and IAGG for random scale-free networks with $|V| = 100$ nodes and varying values of l.

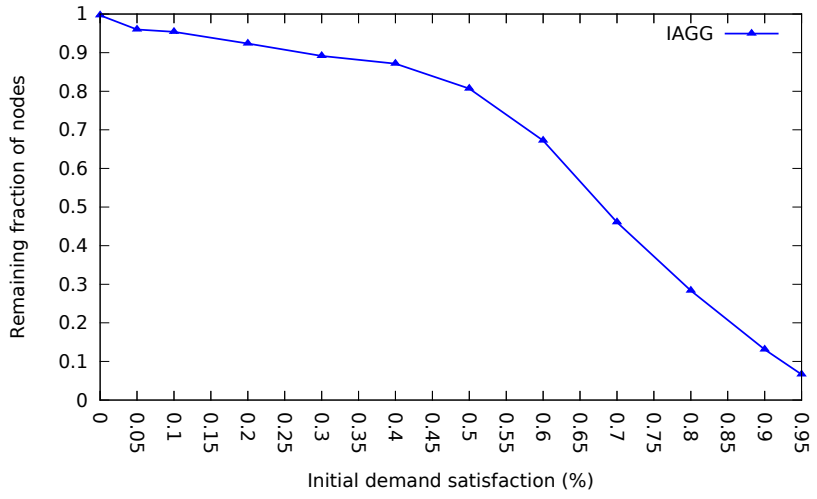


Figure 3.6: Average number of components in the last iteration of IAGG in relation to the original $|V| = 100$ nodes for random scale-free networks for varying level of initial demand satisfaction.

$\|V\| \setminus l$	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.92	0.94	0.96	0.98
1000	X	X	X	X	X	X					
2000				X	X	X	X	X			
3000					X	X	X	X	X		
4000						X	X	X	X	X	
5000						X	X	X	X	X	X
10000							X	X	X	X	X
15000										X	X
20000											X
25000											X

Table 3.3: Instance sizes $|V|$ and extension degrees l that comply with the selection rule for medium-sized and large instances are marked by an X in the table.

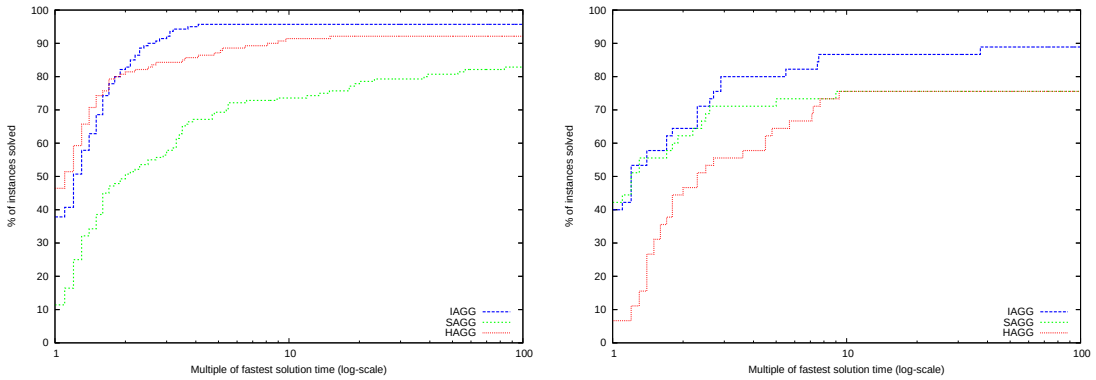
more effective branching decisions within the branch-and-bound procedure. To this end, in the standard solver MIP, we increased the branching priority on variables that enter the master problems of the aggregation scheme in early iterations. We found that these branching priorities did not lead to better running times for MIP. This suggests that the cutting planes generated within the aggregation procedure are more powerful than the ones generated within MIP.

Behavior for Medium-Sized and Large Instances

We now consider larger instances within a range of 1000 to 25000 nodes as well as an initial demand satisfaction from 60 % to 98 %. We applied a selection rule to sort out instances which are 'too easy' or 'too hard' to solve. We required that for at least three out of five instances per class, any of the four methods has a solution time in the time interval reaching from 10 seconds to the time limit of 10 hours. Table 3.3 lists the instances which comply with this selection rule. Note that the relevant instances can be located mainly on the diagonal as an increasing instance size requires an increasing level of initial capacities in order to remain solvable within the time limit.

Figure 3.7a shows the performance profile for the medium-sized instances with up to 5000 nodes. We observe that for these instances, the HAGG implementation performs best and IAGG is almost as good. Accordingly, these results suggest the choice of one of those two methods. However, the picture changes for the large instances with at least 10000 nodes, which have a high level of preinstalled capacities, see Figure 3.7b. Here, HAGG performs poorly and instead, SAGG solves a majority of the instances fastest ($\sim 42\%$), while IAGG performs only slightly worse.

As a result of Figures 3.7a and 3.7b, we come to the conclusion that the overall best choice is IAGG, as it is not much worse than HAGG on the medium-sized instances and much better on the large networks. Furthermore, it outperforms both other implementations when considering small multiples of the shortest solution times.



(a) Instances with up to $|V| = 5000$ nodes (b) Instances from $|V| = 10000$ nodes on

Figure 3.7: Performance profile for the large random scale-free networks from Table 3.3.

The comparison between IAGG and MIP on the instances from Table 3.3 is shown in Table 3.4. The instances are grouped by initial demand satisfaction. We see that IAGG is better comparing the average solution times for almost all instances under consideration. A special remark is to be made on the total number of solved instances, which is the first number in each cell. Here, we see that within the time limit, IAGG can always solve at least as many instances as MIP, often more. Furthermore, we note that even though the number of solved instances is larger for IAGG, the geometric mean of the solution times is still lower compared to the solution times of MIP. Thus, IAGG solves the instances significantly faster than the standard MIP approach.

These statements are underlined by the performance profile over the same instances, which is shown in Figure 3.8. The aggregation scheme IAGG clearly outperforms the standard approach MIP. It solves about 86 % of all instances fastest. In addition, IAGG was able to solve a higher percentage of the overall number of instances within the time limit when compared to MIP.

To investigate why the aggregation scheme solves the instances so much faster, we examine the average number of network components in the final iteration for four selected instance sizes, $|V| \in \{1000, 2000, 3000, 4000\}$, as this number strongly influences the size of the aggregated network design problem, see Figure 3.9.

The results are comparable to those for random scale-free networks with $|V| = 100$ nodes as shown in Figure 3.6. Due to the larger size, these instances could only be solved for higher levels of initial demand satisfaction, for example at least 60 % for graphs with 1000 nodes. The plot shows that the aggregation algorithms can indeed reduce the number of nodes significantly when compared to the number of nodes in the original graph.

As an example, Table 3.5 presents the results for the instances with $|V| = 3000$ nodes. For different values of l, average solution times of the aggregation schemes are compared with those of MIP.

In total, these results for medium and large instances confirm our findings for in-

$\|V\|$	l	MIP		IAGG	
		solved	time[s]	solved	time[s]
1000	0.75	4	142.31	**5**	**92.40**
2000	0.75	3	6715.29	**4**	**1583.72**
1000	0.8	4	52.75	**5**	**34.25**
2000	0.8	5	944.84	**5**	**143.04**
3000	0.8	2	6605.03	**3**	**1806.96**
1000	0.85	5	19.68	**5**	**9.31**
2000	0.85	5	140.75	**5**	**43.15**
3000	0.85	5	2783.47	**5**	**397.87**
4000	0.85	3	14164.16	**5**	**884.52**
5000	0.85	1	31214.39	**3**	**5484.70**
2000	0.9	5	24.39	**5**	**12.78**
3000	0.9	5	340.59	**5**	**35.94**
4000	0.9	5	1433.56	**5**	**115.57**
5000	0.9	5	3113.36	**5**	**195.83**
10000	0.9	0	∞	**3**	**11951.60**
2000	0.92	5	11.32	**5**	**10.14**
3000	0.92	5	63.97	**5**	**24.38**
4000	0.92	5	979.59	**5**	**51.89**
5000	0.92	5	1499.94	**5**	**61.39**
10000	0.92	0	∞	**3**	**6139.25**
3000	0.94	5	21.08	**5**	**13.17**
4000	0.94	5	121.86	**5**	**21.30**
5000	0.94	5	450.84	**5**	**36.23**
10000	0.94	3	9004.99	**5**	**338.51**
3000	0.96	**5**	**5.62**	5	11.76
4000	0.96	5	30.08	**5**	**8.76**
5000	0.96	5	65.41	**5**	**20.21**
10000	0.96	4	4012.20	**5**	**109.48**
15000	0.96	2	25343.05	**5**	**2395.53**
5000	0.98	**5**	**6.70**	5	9.53
10000	0.98	5	421.14	**5**	**33.07**
15000	0.98	5	3690.52	**5**	**90.50**
20000	0.98	5	10982.81	**5**	**470.47**
25000	0.98	2	22687.73	**4**	**4372.76**
25000	0.98	2	22687.73	**4**	**4372.76**

Table 3.4: Number of instances solved and average solution times[s] of MIP and IAGG for random scale-free instances with $|V|$ nodes and initial demand satisfaction l.

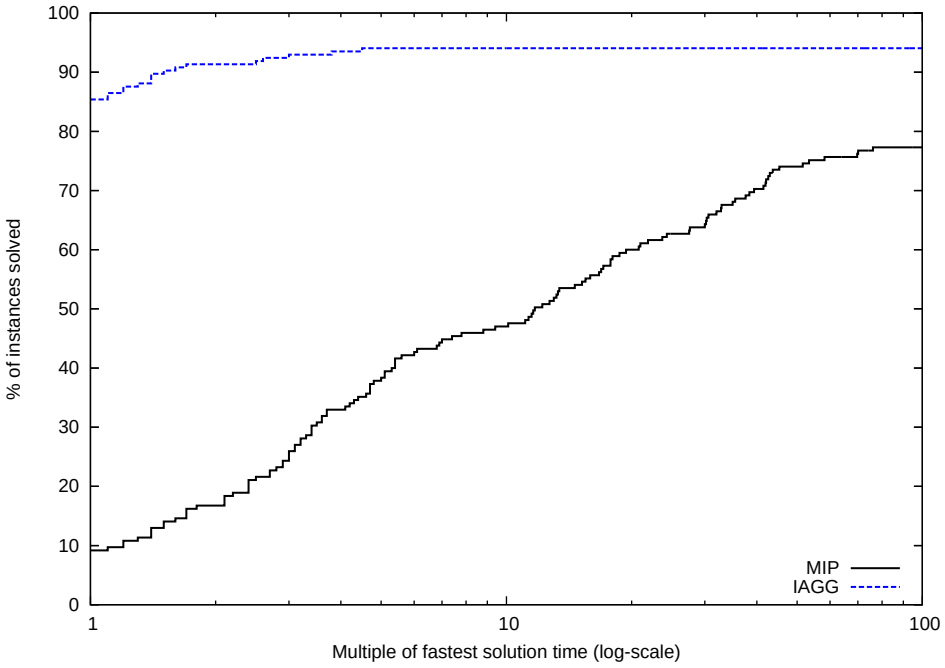


Figure 3.8: Performance profile for all instances from Table 3.3, comparing MIP and IAGG.

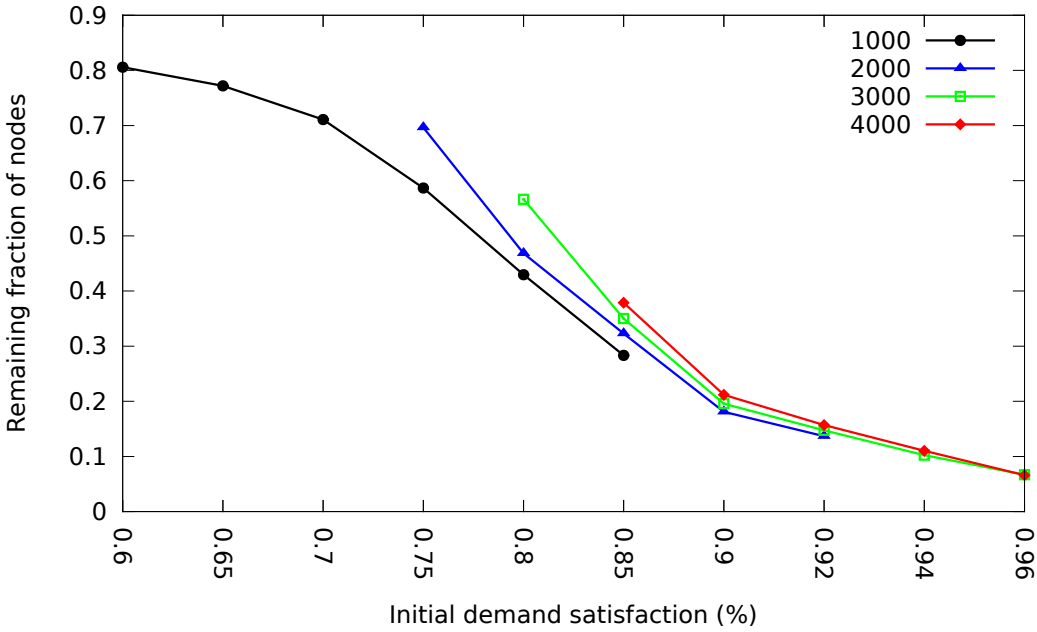


Figure 3.9: Average number of components in the last iteration of IAGG in relation to the original number of nodes for $|V| \in \{1000, 2000, 3000, 4000\}$ for random scale-free networks with varying level of initial demand satisfaction.

l	MIP		IAGG		SAGG		HAGG	
	solved	time[s]	solved	time[s]	solved	time[s]	solved	time[s]
0.8	2	6605.03	**3**	**1806.96**	2	5914.47	2	1299.07
0.85	5	2783.47	**5**	**397.87**	5	3761.89	5	623.59
0.9	5	340.59	5	35.94	5	58.96	**5**	**28.68**
0.92	5	63.97	5	24.38	**5**	**16.23**	5	18.00
0.94	5	21.08	5	13.17	**5**	**9.84**	5	10.44
0.96	**5**	**5.62**	5	11.76	5	7.99	5	7.81

Table 3.5: Number of instances solved and average solution times[s] for random scale-free networks with $|V| = 3000$ nodes and initial demand satisfaction l.

stances with $|V| = 100$ nodes. Namely, MIP is vastly outperformed by the aggregation schemes. IAGG generally performs best with respect to the number of solved instances and with respect to the solution times.

3.5.3 Disaggregation According to the Global Subproblem

Theorem 3.4 opened up a new version of the aggregation algorithms that relies on the global subproblem only. In this subsection, we want to test this version with respect to two questions: On the one hand, we ask whether more conservative disaggregation leads to a more compressed master problem at termination (or just to more iterations). On the other hand, we may compare the two policies with respect to runtime.

	components						
l	MIP	IAGG		SAGG		HAGG	
		global	local	global	local	global	local
0.85	3000.0	708.6	1056.4	703.0	884.8	**673.0**	688.4
0.90	3000.0	466.8	580.2	459.2	528.4	**447.2**	454.0
0.92	3000.0	371.6	438.8	369.4	408.6	**362.2**	365.8
0.94	3000.0	271.0	307.2	270.6	291.4	**268.6**	269.8
	time[s]						
0.85	344.66	76.25	127.00	77.53	116.04	62.87	**47.55**
0.90	48.32	15.21	17.64	10.15	14.85	12.28	**9.92**
0.92	22.41	10.70	13.27	**6.78**	8.88	9.50	10.36
0.94	8.02	6.84	8.02	**4.09**	4.97	5.44	5.43

Table 3.6: Number of components (arithmetic averages) at termination as well as solution times[s] (geometric averages) comparing disaggregation according to the global subproblem (column 'global') or local subproblems (column 'local') for random scale-free instances with $|V| = 3000$ nodes and initial demand satisfaction l.

Table 3.6 exemplarily shows results on these two questions on instances with 3000 nodes. It includes all instance sets from Table 3.3, where every instance has been solved by all four methods within the time limit. This test has been run on superior machines (Intel Xeon E5-2690 3.00 GHz computers with 25 MB cache and 128 GB RAM) as well as a later version of Gurobi (Version 6.00) which explains the faster solution times compared to Table 3.5.

We see that the number of components in the final master problem of IAGG and SAGG is indeed reduced by about 10% to 20% when using the 'global' setting. Moreover, slight savings in runtime can be observed for those methods. Remarkably, in contrast to IAGG, HAGG arrives at a final aggregation of pretty much the same size on average with either setting. Consequently, disaggregation according to the global sub-

problem does not save any runtime, it is even slightly detrimental. Moreover, HAGG consistently achieves the smallest aggregated master problems on average among the three aggregation methods. This fact further supports the general idea behind HAGG of using an integrated aggregation scheme together with a nontrivial initial aggregation. Furthermore, the development of good problem-specific heuristics for finding a good initial aggregation seems promising—although this is outside the scope of this work. Just imagine we could somehow guess the configuration of the master problem in the last iteration of SAGG: this would obviously lead to a vastly improved version.

3.5.4 Performance on a Real-World Street Network

The graph *rome99* from the 9th DIMACS Implementation Challenge on shortest path problems [DGJ06] describes a large portion of the road network of the city of Rome from 1999 (3353 vertices, 8870 directed edges). Its size is comparable to that of the scale-free networks with 3000 nodes. The corresponding results can be found in Table 3.5. It not only provides a realistic network topology but also comes with a distance measure on the edges. We use the latter values as module expansion costs, as a distance-proportional cost seems a plausible choice. The module sizes are of the size of 0.25 % of the total demand as for the previous test instances. The demands and initial capacities were again generated randomly.

Table 3.7 shows the solution times for each aggregation method as well as for plainly solving the network expansion problem via MIP for initial capacities ranging from 90 % up to 98 % in steps of one percent.

l	MIP		IAGG		SAGG		HAGG	
	solved	time[s]	solved	time[s]	solved	time[s]	solved	time[s]
0.90	0	∞	0	∞	0	∞	0	∞
0.91	1	11041.70	**1**	**1201.86**	1	6297.84	1	1894.52
0.92	1	94.92	1	130.39	1	157.05	**1**	**56.22**
0.93	2	1109.27	2	478.40	2	747.56	**2**	**169.59**
0.94	4	826.98	5	311.36	4	421.08	**5**	**269.42**
0.95	5	133.37	5	76.08	5	122.41	**5**	**58.84**
0.96	**5**	**11.9**	5	27.63	5	36.57	5	20.13
0.97	**5**	**2.57**	5	9.78	5	8.89	5	8.56
0.98	**5**	**1.53**	5	4.65	5	4.95	5	4.61

Table 3.7: Number of instances solved and average solution time[s] for a real street network with initial demand satisfaction l.

We see a very similar behavior as for the scale-free instances: MIP is fastest only for very easy instances and our aggregation algorithms start to take the lead from a certain level of difficulty onwards.

Remark 3.5. More computational results on the aggregation methods on real network topologies have been compiled. Computations on single-commodity adaptations of instances from the popular library of network design problems *SNDlib* [OPTW07] can be found in [BLM+15]. Among other things, they show that IAGG can be successful even if the network is almost completely disaggregated. Furthermore, computations on the German railway network are included in [Bä16].

3.6 Extending the Aggregation Scheme to More Complex Network Design Problems

An important issue is that many real-world applications include more complex features and cannot be modeled in the current single-commodity maximum-flow setting. Those features include e.g. multi-commodity flows, robustness/survivability, multiple scenarios, or nonlinearities. Respective models have been brought up already in Chapter 2, Subsections 2.1.4 and 2.1.5. There are natural extensions of our algorithms for those cases but also additional peculiarities that have to be considered in detail in each case. In this section, we will address cases in which the problem can still be modeled as an MILP. An example for a nonlinear network design problem will be discussed in Section 3.7.

The ideas shared in the following will hopefully help the reader to decide whether an aggregation approach as the one presented in this chapter is promising for his or her own network design problem.

3.6.1 Multi-Commodity Flow

The aggregation schemes can directly be extended to the multi-commodity case (see (2.9) for a formulation of a multi-commodity flow network expansion problem). However, since multi-commodity flow problems do not give a canonical minimum cut in case of infeasibility in the same way as single-commodity problems do, we have to specify how disaggregation is supposed to work. First of all, note that any disaggregation policy will lead to an exact algorithm as long as we do not erroneously terminate with a suboptimal (or infeasible) solution. As no canonical disaggregation method is available anymore, we use the following heuristic rule, which represents the most direct extension of our implementation so far: in case of an infeasible subproblem, we maximize the total throughput over all commodities and disaggregate along all arcs that limit this flow (defined by the dual variables of the respective capacity constraints). After that, we run a breadth-first search to determine the new components.

To demonstrate the extendibility to the multi-commodity case, we include some results for random scale-free networks with a few commodities. These instances are random scale-free networks with $|V| = 100$ and $l = 0.8$ as well as medium-sized networks with $|V| = 3000$ and $l = 0.96$. The short solution times for these network sizes obtained in the single-commodity case allow for a multi-commodity study with

a varying number of commodities b, for which the demands were again randomly drawn.

Table 3.8 compares the number of instances solved and the average solution times obtained by MIP and the three implementations of our aggregation scheme for the small instances with up to 25 commodities. We see that an increasing number of commodities increases the difficulty of the problem significantly. On the one hand, this is due to the obvious fact that the subproblems are now multi-commodity network flow problems. On the other hand, we also observe that the graphs tend to be disaggregated much further at termination. Nevertheless, implementations IAGG and HAGG both outperform MIP, and for a higher number of commodities, IAGG is preferable.

b	MIP		IAGG		SAGG		HAGG	
	solved	time[s]	solved	time[s]	solved	time[s]	solved	time[s]
5	5	7.13	5	9.72	**5**	**6.78**	5	6.93
10	5	258.43	**5**	**52.47**	5	126.54	5	86.12
15	4	166.98	5	147.35	5	334.15	**5**	**140.81**
20	3	966.64	**4**	**337.68**	4	1779.86	4	363.40
25	3	2358.80	**5**	**876.46**	3	2678.55	4	2258.37

Table 3.8: Number of instances solved and average solution time[s] for small multi-commodity instances with $|V| = 100$ nodes, an initial demand satisfaction of $l = 0.8$, and an increasing number of commodities b.

b	MIP		IAGG		SAGG		HAGG	
	solved	time[s]	solved	time[s]	solved	time[s]	solved	time[s]
2	5	120.01	5	54.13	5	59.98	**5**	**50.08**
3	5	337.98	5	152.21	5	156.87	**5**	**132.78**
5	5	4436.23	5	620.31	4	863.54	**5**	**533.49**
7	3	15302.56	4	2249.79	1	15708.12	**5**	**5169.41**
10	0	∞	**2**	**5465.29**	0	∞	0	∞

Table 3.9: Number of instances solved and average solution time[s] for medium-sized multi-commodity instances with $|V| = 3000$ nodes, an initial demand satisfaction of $l = 0.96$, and an increasing number of commodities b.

The corresponding results for the medium-sized instances in Table 3.9 show a similar picture. The dimension of the graph allows for fewer commodities to be considered, but the instances are still best solved by the aggregation schemes. In this case, IAGG does not only solve more instances to optimality than MIP; there even is no single instance which is solved faster by MIP than by IAGG.

However, please keep in mind that the instances examined in Tables 3.8 and 3.9 still feature relatively few commodities compared to the size of the network. If this relation shifts towards a high number of commodities (consider e.g. the instances from *SNDlib* [OPTW07]), the algorithms will likely result in complete disaggregation of the network. For a large number of commodities, further algorithmic enhancements should be included, such as methods that aggregate commodities, in addition to aggregating the network topology. Aggregation and disaggregation of commodities can be integrated into a branch-and-bound tree in essentially the same way as it has been done for topology disaggregation (see Section 3.4). In case of an infeasible subproblem, we need a procedure to decide when to disaggregate the topology and when to disaggregate demands. A direct choice would consist of using two types of local multi-commodity-flow subproblems, one using aggregated demands and one using disaggregated demands. In case the aggregated version is feasible, but the disaggregated one is not, this signals demand disaggregation. If both are infeasible, disaggregating the topology seems to be the natural choice. It would be interesting to test this idea in a suitable environment in further research.

3.6.2 Routing Costs

Considering routing costs in addition to the costs related to installing network upgrades (see (2.9)) does allows for an extension as straightforward as it was possible in the previous subsection. The reason is that in order to show correctness of the algorithm in Theorem 3.2, we needed the fact that no additional costs are induced in the subproblems. However, this seems to be completely unrealistic if routing costs exist for any arc. Thus, the consequence will be complete disaggregation.

There are, however, ways we may deal with (nonnegative) routing costs:

- Though considering routing costs in a problem implies that they are not completely negligible, additional routing costs caused by subproblems will likely only make up for a small fraction of the total cost. Hence, we may use the aggregation scheme as an approximation algorithm with predefined (total or relative) approximation error: we terminate as soon as all subproblems are feasible and the gap between the resulting primal solution (constructed by the subproblems) and the dual bound (given by the master problem) is within a given range. If having an approximate solution is acceptable in the application context, this seems to be the most straightforward extension approach.

- Routing costs may be projected onto the master problem using Benders optimality cuts. As mentioned in Subsection 3.3.4 those cuts are numerically challenging and should only be used at a carefully dosed rate. However, the approach has been used successfully on railway networks in [Bä16, Chapter 9].

Moreover, there are cases in which routing costs can be transformed into upgrades by suitable reformulation. This was the case e.g. for discrete lot-sizing instances in [Wen16, Chapter 6].

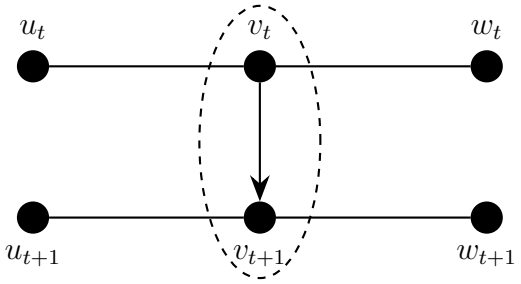


Figure 3.10: Illustration of a typical situation in a time-expanded graph: Nodes v_{t+1} and v_t (encircled) form an aggregated component that spans over multiple time periods.

3.6.3 Time-Expanded Networks

In Subsection 2.1.4, time-expanded graphs have been introduces as a way of reformulating instationary network flow problems. If such a formulation is used, our aggregation algorithms can be employed to solve instationary linear problems. Time-expanded graphs based on fine-grained time discretizations even represent a premier target for our algorithms as the increase in difficulty when compared to the stationary version mainly comes from an enormous graph size.

Aggregation of time steps is not new and has been used in many applications, though applying our scheme to time-expanded graphs corresponds to an aggregation in time and space simultaneously. In [Wen16, Chapter 7], our aggregation scheme has been applied to discrete lot-sizing instances, whose underlying graphs are particular time-expanded networks. However, the specific structure of such a graph deserves special consideration. Note that it is not bidirected anymore, which can lead to problems when identifying a suitable cut as has been mentioned before (see Subsection 3.3.2). We need to have a backup plan for performing disaggregation in case the local subproblems are infeasible but do not provide cut edges in G.

Apart from that, having one-way edges inside an aggregated component may lead to weak relaxations. For instance, consider the case depicted in Figure 3.10. The relaxation represented by the master problem may not resist the temptation to send flow 'back in time' from v_{t+1} to v_t. This is quite possible as the aggregated problem by design does not consider any restriction regarding flow between nodes of the same component. This will likely result in eventual diaggregation of that component.

However, we can overcome this by only incorporating the flow direction into the master problem, but not the capacity limitation of the arc from v_t to v_{t+1}. For the integrated algorithm, this can simply be done by leaving that part of the constraint in the formulation right from the start (or adding it later). This admittedly adds more constraints to the master problem, though in contrast to an eventual disaggregation it does not introduce additional upgrade variables. This is due to the fact that the imaginary arc from v_{t+1} back to v_t cannot be upgraded by any means in a time-expanded formulation.

Furthermore, note that using a path-based formulation—if reasonable in the overall context—has the advantage that it preserves possible routes inside aggregated components. For instance, in Figure 3.10, no flow can be sent from u_{t+1} to w_t in the master problem as there is no path in the original network that achieves this.

3.6.4 Multi-Scenario Problems

A multi-scenario version of Problem (3.1) has been introduced in Subsection 2.1.4. It requires to find a minimum cost upgrade decision y such that there is a feasible routing for all scenarios in some scenario set $\mathcal{U}$ (cf. Formulation (2.10)). As described in Subsection 2.1.4, there are several ways to cope with the bilevel nature of this problem.

In the particular case of single-commodity network design, a cut formulation is available that does not need explicit flow variables (see [CJL+16]). In this case, applying aggregation is directly possible; it effectively amounts to preventing the scenario separation routine from proposing cutting planes that runs through components. Only if all scenarios are feasible, the scenario separation routine is allowed to give such a cut which constitutes a disaggregation step.

Let us now assume that $\mathcal{U}$ is finite and (2.10) has been reformulated by its *deterministic equivalent* (see (2.11)). In this case, aggregation is directly applicable as well: the master problem in that case is a multi-scenario network design problem on an aggregated graph. The subproblems (global or local) decompose into many single-scenario network flow problems that all have to be feasible. Due to this decomposition, the positive effect of our aggregation can be expected to be even more pronounced as in the single-scenario case: the relative decrease in runtime between the original problem and and an aggregated problem can be expected to increase for a more complex type of problem, which a multi-scenario problem clearly is. However, before employing an aggregation scheme, we have to be confident that there exists some aggregated graph that is sufficiently detailed such that it leads to the optimal solution of the original multi-scenario problem. This likely requires relatively few scenarios compared to the graph size—similar to the situation in the multi-commodity case in Subsection 3.6.1 above.

3.7 Aggregation for Topology Planning Problems on Gas Transportation Networks

This section gives an outline about transferring the iterative aggregation scheme to a network expansion problem with nonlinear dependencies, namely from gas network optimization. This type of problem has been introduced in Subsection 2.1.5 and we will use the notation and problem formulations introduced there in this section. In particular, a formulation (together with a motivation of its details) for the network design problem on gas networks has been given for passive gas networks (see Formulation (2.13)). Readers who are not familiar with gas network optimization are advised

to consult Subsection 2.1.5 at this point. Note, however, that details of compressor modeling or similar are not important for this section as we focus on the network topology.

Let us briefly recall a problem formulation of the topology planning problem on gas networks, which is a conceptual extension of (2.13) in the sense that it contains constraints for active elements (modeled as arcs $a \in A_{\text{active}}$):

$$\text{(3.8a)} \quad \min \quad \sum_{a \in A_{\text{valve}}} k_a y_a$$

$$\text{(3.8b)} \quad \text{s.t.} \quad \sum_{a \in \delta^+(v)} q_a - \sum_{a \in \delta^-(v)} q_a = d_v \qquad (\ \forall v \in V'\ )$$

$$\text{(3.8c)} \quad p_u^2 - p_v^2 = \lambda_a q_a |q_a| \qquad (\ \forall a = (u,v) \in A'\ )$$

$$\text{(3.8d)} \quad q_a \leq \overline{q_a} y_a \qquad (\ \forall a = (u,n) \in A_{\text{valve}}\ )$$

$$\text{(3.8e)} \quad q_a \geq \underline{q_a} y_a \qquad (\ \forall a = (u,n) \in A_{\text{valve}}\ )$$

$$\text{(3.8f)} \quad p_u - p_n \leq (\overline{p_u} - \underline{p_n})(1 - y_a) \qquad (\ \forall a = (u,n) \in A_{\text{valve}}\ )$$

$$\text{(3.8g)} \quad p_u - p_n \geq (\underline{p_u} - \overline{p_n})(1 - y_a) \qquad (\ \forall a = (u,n) \in A_{\text{valve}}\ )$$

$$\text{(3.8h)} \quad g_a(p_u, p_v, q_a, z_a) \geq 0 \qquad (\ \forall a = (u,v) \in A_{\text{active}}\ )$$

$$\text{(3.8i)} \quad q_a \in [\underline{q_a}, \overline{q_a}] \qquad (\ \forall a \in A'\ )$$

$$\text{(3.8j)} \quad p_v \in [\underline{p_v}, \overline{p_v}] \qquad (\ \forall v \in V'\ )$$

$$\text{(3.8k)} \quad y_a \in \{0,1\} \qquad (\ \forall a \in A_{\text{valve}}\ )$$

$$\text{(3.8l)} \quad z \in \{0,1\}^r \qquad .$$

For each arc a, we have a continuous flow variable q_a, and for each network node v, there is a continuous pressure variable p_v. In contrast to Formulation (2.13), we assume that the gas network to be upgraded contains active elements. We do not want to go into detail here, but just take into account that the problem involves some binary decisions z associated to active elements. Those do not influence the objective function, but may occur in master problems as well as inside of components. Finally, as usual the variables y_a correspond to possible extensions. They are discrete and are the only ones that appear in the objective function. Besides bounding constraints (3.8i) and (3.8j) for the continuous variables, we have flow conservation constraints (3.8b) for each node and pressure loss equations (3.8c) for each arc that couple flow and pressure. Those are the most important types of constraints for the aggregation scheme. All constraints associated to an active element $a = (u,v) \in A_{\text{active}}$ are subsumed under a constraint (3.8h) for that element, involving a vector-valued (possibly nonlinear) constraint function g_a that is defined in terms of binary z-variables as well as p_u, p_v and q_a, i.e. the arc flow and the pressure values at adjacent network nodes. Constraints (3.8h) are treated in the master problem or a local subproblem, depending on whether the corresponding arc has both endpoints in the same component. Hence their detailed structure is not important for the aggregation scheme. This also applied to Constraints (3.8d) to (3.8g) which model the behavior of y_a operating a valve.

Remark 3.6. As a preliminary note, aggregation for gas networks has been studied in [RMWSB02]. However, the motivation in that paper is very different from ours. In [RMWSB02], aggregation is performed such that aggregated components have a unique solution under suitable assumptions. In our scheme, we aggregate parts of the network that we assume to be uncritical, i.e. which have some flexibility.

The general idea of the scheme again is to identify bottlenecks and assume that within the aggregated components, the problem is treated sufficiently well by a very coarse relaxation. This philosophy is not unproblematic here as the idea of a *bottleneck* is somewhat controversial in the context of gas network optimization (see [KHPS15, Chapter 11]). In contrast, for single-commodity network flow problems, bottlenecks are well described by limiting cuts. As before, an aggregated master problem will yield a solution that a (global) subproblem tries to extend to a solution of the original problem. Otherwise we may refine the partition guided by the results from local subproblems. There will be some interplay between global and local subproblems as the global subproblem will rely on heuristic information from the local subproblems. The algorithm terminates when extendibility of the optimal aggregated solution can be confirmed. This also solves the original problem to optimality, if the following key properties hold (cf. Theorem 3.2):

- The aggregated problem is a relaxation of the original problem.
- Extension and/or modification of the master problem's solution by subproblems does not change its objective function value.

As an overview, the general algorithm is outlined in Algorithm 3.1. In the following, details on each part of the method are discussed with a focus on Problem (3.8).

Algorithm 3.1 Iterative aggregation for topology planning on gas networks

```
1: Start with initial aggregation
2: while optimum has not been found do
3:     Solve the AGGREGATED MASTER PROBLEM
4:     for each aggregated component do
5:         Solve a LOCAL SUBPROBLEM for that component
                                          ▷ variables on master arcs fixed
6:     Solve a GLOBAL SUBPROBLEM that checks extendibility
                                          ▷ all discrete variables fixed
7:     if solution is extendible then
8:         optimum found
9:     else
10:        disaggregate according to local subproblems
```

The Master Problem The aggregated problem only considers constraints on arcs whose endpoints belong to different components. Therefore it contains e.g. flow and pressure bounds and extension decisions for master arcs. Flow conservation constraints are added up to obtain the net in- and outflow constraint for the aggregated node—as in the case of single-commodity network design. The pressure variables require more careful treatment: We cannot assign a single pressure variable to a component and impose adjacent pressure loss equations as this would destroy the relaxation property. Therefore, we still represent all original nodes in the master problem, each with its own pressure variable. Those can be coupled by known bounds on their difference, possibly depending on the current flow bounds. However, any coupling of pressure values due to pressure loss equations (3.8c) for arcs inside a component are not considered by the master problem, which of course weakens the relaxation.

In terms of complexity, the aggregated problem is an MINLP, equally hard and of the same structure as the original problem, but typically much smaller.

The Local Subproblems For each component, we have a subproblem that checks whether the throughput planned for this component by the solution of the master problem can actually be realized. Hence, all variables associated to arcs connecting different components are fixed, including the continuous quantities flow and pressure. All constraints inside the component are considered with the exception of possible extensions, i.e. the y-variables are fixed to 0. Therefore, the subproblems are MINLP feasibility problems. In order to obtain some information in case of infeasibility, we use slack variables such that a slack-0 solution corresponds to a feasible extension of the master problem's solution. This slack solution serves as guidance where to disaggregate and can be used for fixing the discrete decisions inside a component to the value that has been found to be relatively best. Some useful slack models can be found in [KHPS15, Chapter 11]. It is important to choose a model that is guaranteed to be feasible.

The Global Subproblem It might happen that an optimal extension (y-values) is found, although not all local subproblems are feasible. This happens regularly in single-commodity network design and can be expected to occur even more frequently for the case of gas networks. Especially for passive subnetworks it is very unlikely that the imposed boundary flow and pressure values fit together. Therefore, we solve a global subproblem that allows adjusting the continuous variables. As this is a problem on the whole graph, we fix all discrete variables. Expansion decision y and discrete decisions z on the master arcs are taken from the optimal master solution; discrete decisions z inside aggregated components are set to their value for the minimum slack solution of the corresponding local subproblem.

The global subproblem therefore is an NLP—a nonlinear program without integer variables. Also, it is acceptable to only solve it to local optimality, as a false-negative answer would just lead to unnecessary disaggregation but does not harm the correctness of the aggregation algorithm itself. It would make sense to first apply the global

subproblem before solving the local subproblems if we had other heuristics for fixing discrete decisions inside a component or could afford leaving them unfixed.

Disaggregation In case the optimal master can not be made feasible by local subproblems or the global subproblem, we draw the conclusion that the current aggregation is too coarse. We disaggregate all infeasible (i.e. positive-slack) components along some cut (though arguments can be made to disaggregate only some of them). Unfortunately, there is no canonical choice for the disaggregating cut. It is natural to use a cut containing the element with largest slack in the local subproblem's optimal solution. The cut could then be completed by some simple heuristic, e.g. by finding the minimum completion cut with respect to some auxiliary arc capacities (e.g. the difference of their slack value to the maximum slack value in the component), which could be done via a max-flow computation.

The Initial Aggregation Starting with the trivial aggregation that collapses the entire graph into a single vertex seems less promising than in the case of single-commodity network design—in which case the subproblems are solvable in polynomial time. Hence, the difference in difficulty to the original problem is a lot larger. In particular, the first local subproblem would be very expensive in the current setting. We could speed up the first master iterations by solving a relaxation of the respective master problem (or an otherwise simplified master as long as we obtain a diaggregating cut). Furthermore, the local subproblems do not have to be solved to optimality; theoretically, we could use any incumbent to disaggregate. Alternatively, we may want to design some heuristic that constructs the initial aggregation bottom-up.

Integration in branch-and-bound As we have seen in Section 3.4, the aggregation scheme can be embedded into a branch-and-bound framework if the solver supports *lazycuts*. This is also possible for Algorithm 3.1, though most MINLP-solvers do not support such a feature.

First Computational Tests

Based on the above considerations, a version of the aggregation schemes for network design problems on gas networks has been implemented in the course of the master's thesis [Sch15]. We discuss some results very briefly without going into details of the implementation:

The algorithms have been compared to solving (3.8) without using aggregation on two test networks. SAGG has had some promising results on a very sparse network. However, it was not competitive on a large-scale real-world gas network—which also involved many complicated compressor stations—since the network was often completely disaggregated during the algorithm. Moreover, IAGG and HAGG also suffered from way too many disaggregations such that they could not provide any advantage whatsoever.

As an immediate conclusion, we can say that the relaxation provided by the simplest model for the master problem is too weak. This is mainly due to the complete decoupling of pressure constraints inside the components. There is still plenty of room for algorithmic improvements that were outside of scope of this thesis, such as strengthening the relaxation provided by the master problem by cutting planes on pressure variables, designing specialized heuristics for an initial aggregation or designing strong primal heuristics.

However, in real-world gas networks to a large extend the difficulty can be attributed to the nonlinearities involved and the discrete configurations of compressor stations rather than the size of the network (in particular for the second test network in [Sch15]). This does not match the situations the aggregation algorithm has been designed for, showing the borders of applicability of our algorithmic scheme. The situation may well be different for large-scale transportation networks involving relatively few discrete decisions—possibly in connection with some of the enhancements mentioned above.

In the next chapter, we will investigate a setting that is specifically tailored for nonlinear network flow problems like the one from this section.

Chapter 4

Structural Investigations of Piecewise Linearized Flow Problems

In the major part of the previous chapter we dealt with linear mixed-integer optimization problems in which a main difficulty is connected with the sheer size of the network. However, in those network expansion problems we had reasons to believe that we can aggregate details and that the 'critical decisions' can be made on a coarsened version of the network reasonably well, in fact optimal. Consequently, the aggregation algorithms were designed to locally relax the problem and identify a suitable aggregation. On the other hand, in many optimization problems on transportation networks, especially those arising from power supply, the main challenge is connected to the question of how to deal with the problem's nonlinearities, arising, e.g. , from laws of physics. In contrast, simplifying the network topology seems less natural as it is already of moderate size. In Section 3.7, we have seen that aggregation techniques are still applicable for gas network optimization problems, though empirical results are not yet convincing in first empirical tests, for reasons discussed there. In this and the following chapters, we follow a different—yet classical—approach that involves locally strengthening the model (instead of relaxing it) by suitable cutting planes.

In this chapter, we consider a setting where we assume that nonlinearities are dealt with by constructing piecewise linearizations or relaxations of the involved nonlinear functions, see Section 2.2, and in particular Subsection 2.2.6 in the preliminaries chapter. This is a common approach that allows to transform the nonlinear problem into an MIP (relaxation or approximation) and thus to make it accessible to any general-purpose MIP solver. This method is especially promising for problems involving loosely coupled constraints and sparse networks such that nonlinearities can be modeled as a low-dimensional nonlinear function of the flow. This ensures that the number of binary variables introduced by piecewise linear modeling stays in a manageable order of magnitude.

For constructing a piecewise linear approximation, or respectively, for modeling a piecewise linear function, several useful formulation methods are known. The most important of them have been reviewed in Section 2.2, also see [VAN10] for a coverage of formulation methods. Most of the formulations used in practice are locally ideal,

or—in case of the Convex Combination Method—can be adapted to have this property. Hence, the formulations cannot be strengthened further for a single piecewise linear function. However, the situation is different when we consider multiple nonlinear functions that influence each other: in general, the formulation looses its desired property of being ideal—most likely already in the case of just two functions. This raises the question of how the formulation may be strengthened, a question that can not be answered in general but depends on the constraint structure. For example, [ZdF13, KdFN06] examine the case of the *separable piecewise linear optimization knapsack polytope*. They derive various classes of valid inequalities and also arrive at promising empirical results. Reference [SLL13] extends ideal formulations to the case in which an additional indicator variable is present.

We consider a setting which focuses on the network structure of the problem. As we want to avoid restricting ourselves to a specific application, we do not consider the nonlinearities explicitly, but just assume they are modeled as functions of the flow on a network arc. Therefore, we suppose that the flow variable range is subdivided into several intervals, which is a prerequisite for piecewise linearization.

This chapter is based on joint work with Frauke Liers, published in [LM16].

4.1 The Piecewise-Linearized-Flow Polytope

In the following, we consider feasible sets of the form

$$\{(q, z) \in \mathbb{R}^m \times \{0,1\}^n \mid \forall \text{ arcs } a, i \in \mathcal{I}_a : l_i \leq q_a \leq u_i \text{ if } z_i = 1\}.$$

Here, for an arbitrary network arc a, q_a denote real-valued flow variables. Here we use the notation q_a (instead of x_a) that has been introduced as our convention for gas network optimization in Section 2.1.5 to suggest that we are dealing with a nonlinear problem. The variables z_i are binary indicator variables for using an interval on that arc, and $\mathcal{I}_a$ denotes the set of indices belonging to arc a. The input parameters l_i and u_i are lower and upper bounds on the flow value, i.e. the interval boundaries.

We allow several flow intervals on the same network arc a, in order to aim for maximum generality with respect to the results in this chapter. While one can think of situations with overlapping flow intervals (e.g. if the flow itself nonlinearly depended on some quantity, and the flow intervals resulted from mapping a standard interval subdivision for this quantity to the flow-space), currently the main practical application clearly consists of piecewise linearizing the flow. Hence, all instances in the computational experiments in Section 4.3 have the typical structure resulting from piecewise linearization, in particular featuring non-overlapping intervals.

In any case, only one interval can be active—and at least one has to. Therefore, the corresponding z-variables are connected by the constraint

$$\sum_{i \in \mathcal{I}_a} z_i = 1. \tag{4.1}$$

In addition, we have flow conservation and demand satisfaction equations that can be modeled via

$$\sum_{a\in\delta^+(v)} q_a - \sum_{a\in\delta^-(v)} q_a = d_v \tag{4.2}$$

for a given network node v, where d_v denotes its demand.

The above modeling is compatible with any linearization method that uses the logic

$$z_i = 1 \quad \Rightarrow \quad q_a \in [l_i, u_i], \tag{4.3}$$

as is true, for example, for the Multiple Choice Method (MCM) as well as for the Convex Combination Method (CCM), see Sections 2.2.1 and 2.2.2, respectively. Both methods are very flexible since they allow to consider a generalized setting with possibly overlapping intervals without further modeling effort. We start from models based on (4.3) and show how to transfer our results to another popular formulation later, namely one that is based on the Incremental Method (see Section 2.2.3).

In our notation, the special case of a univariate function f of q modeled piecewise linearly by MCM on a connected domain $[l, u]$ with breakpoints $B_1 = l, B_2, \ldots, B_k$, $B_{k+1} = u$ is obtained for $l_{a_i} = B_i, u_{a_i} = B_{i+1}, i = 1, \ldots, k$. In that case we have a 'copy' q_i of the arc flow q for every interval together with

$$l_i z_i \leq q_i \leq u_i z_i \quad \forall\, i = 1, \ldots, k$$

and

$$q = \sum_{i=1}^{k} q_i, \quad f(q) = \sum_{i=1}^{k} \Big[f(B_i) z_i + (q_i - B_i z_i) \frac{f(B_{i+1}) - f(B_i)}{B_{i+1} - B_i} \Big], \tag{4.4}$$

cf. Chapter 2, Subsection 2.2.1.

To eliminate the continuous q-variables, we consider the projection of the feasible set to the z-variables. The convex hull of this projection will be the polyhedron we are examining. It will be denoted by P throughout this chapter, so

$$P = \operatorname{conv}\{z \in \{0,1\}^n \mid \exists q \in \mathbb{R}^m : (4.1), (4.2), (4.3)\}. \tag{4.5}$$

Note that few problems encountered in real-world applications are completely described by the constraints mentioned above. However, the structure we analyze here might very well be present as a substructure and understanding the structure of P—even for special cases—can be helpful. As an example, in the context of water- and gas network optimization, pressure variables are introduced for network nodes, and additional constraints describe the pressure loss along pipes, see (2.14) in Section 2.1.5 for gas network optimization. In water network optimization, essentially the same algebraic approximation of the underlying physics is commonly used [GMMS12]. Those

models fit well to our setting. However, it should be mentioned that in gas network optimization there are also models describing the pressure loss that feature more complex nonlinearities than Equation (2.14). For example, [PFG+15, Equation (7)] gives a formula for the pressure loss, which is not separable in general but only for a constant compressibility factor [PFG+15, Equation (20)]. In this case, again one separable component is a univariate function of the arc flow. Moreover, [GMMS12] also discusses a bivariate type of nonlinearity, where our modeling does not apply.

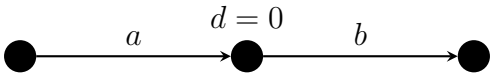
A complete description of P not necessarily leads to a complete description of the polyhedron involving the q-variables, but valid inequalities for P can still be expected to represent strong cuts. In [BCD+08] it is concluded that projected Chvátal-Gomory cuts are effective for instances of the *MIPLIB 3.0*. As a guideline, the strength of those cuts for an MILP depends on whether optimizing the integer variables is the essence of the problem, as the authors of [BCD+08] phrase it. This applies to the network transportation problems we have in mind. For example, in gas network optimization, the problem that results from fixing all integer variables can be solved relatively fast by a general-purpose NLP-solver. Indeed, it turns out in [PFG+15] that it is affordable to solve this NLP to local optimality as a subproblem many times. Also, [Gei11b] reports that after fixing all integer variables, the problem is solved very quickly by an exact solver. As an extreme case, if the objective function only depends on the z-variables, we are guaranteed that optimizing over P yields the overall optimum.

4.2 Polyhedral Studies and a New Class of Perfect Graphs

In this section we study the structure of P for specific network substructures. We start with the case of two adjacent network arcs and then advance to larger substructures.

4.2.1 Paths of Length Two

The most simple nontrivial case is that of two consecutive network arcs together with one flow conservation constraint. Each arc may have multiple, possibly overlapping, distinguished flow intervals.



In this scenario, for every integral solution in P one variable per arc is set to 1. The flow conservation constraint implies that those two nonzero variables correspond to intervals with nonempty intersection. On the other hand, this condition is also sufficient for a feasible integral solution. Hence, the problem of finding an optimal point in P is equivalent to finding an optimal edge in the graph that models interval compatibilities. We formally define it as follows:

Definition 4.1 (Compatibility Graph)**.** Given a set P as defined in (4.5), the *compatibility graph* G_{COMP} corresponding to that instance is an undirected graph that has a node for

each z-variable and an edge between two nodes if and only if requiring the arc flows to lie in the corresponding interval admits a solution to the underlying flow problem consisting of (4.2).

This graph will play an important role for deriving our results on path networks. In the simple case above, G_{COMP} has an arc between two nodes if and only if the corresponding intervals belong to different network arcs and have nonempty intersection. It can be adapted easily for the case in which the middle node has nonzero demand. Indeed, for general $d \in \mathbb{R}$, interval $I_1 = [l_1, u_1]$ on the first network arc is compatible with some interval $I_2 = [l_2, u_2]$ on the second network arc, if and only if there exist $q_1 \in I_1, q_2 \in I_2$ such that $q_1 + d = q_2$. This is equivalent to the requirement that $[l_1, u_1]$ and $[l_2 - d, u_2 - d]$ have a nonempty intersection. Therefore, we deal with nonzero demands by an appropriate interval shifting.

By the above considerations, we have reformulated the problem of optimizing a linear objective function over P as the problem of determining the maximum weight edge in a graph, namely G_{COMP}. Here, the weight of an edge $e = (a_i, b_j)$ is given by the objective value associated to choosing interval a_i on one network arc and interval b_j on the other. For the case of paths of arbitrary length as underlying networks, this generalizes to finding a maximum weight k-clique, where k is the number of arcs, as we will discuss in the next subsection. But also for a path of length 2, viewing the problem as a clique problem is helpful: we notice that the compatibility graph in the two-arc-setting is bipartite, and hence perfect. For the class of perfect graphs several combinatorial problems which are NP-hard on general graphs, such as finding the maximum clique, the maximum stable set, or the chromatic number, can be solved in polynomial time [GLS88]. We also know explicitly that adding the stable set inequalities, i.e. inequalities of the form $\sum_{e \in S} x_e \leq 1$ for some stable set S, suffices to describe the convex hull of all clique vectors, where exactly the inclusion-wise maximal stable sets constitute facets [Pad73]. We summarize some well-known properties of perfect graphs in the following theorem:

Theorem 4.2 (Characterizations of Perfect Graphs from the Literature)**.** *Let G be a graph. The following conditions are equivalent:*

a) G is perfect, i.e. for every induced subgraph of G the chromatic number is equal to the clique number.

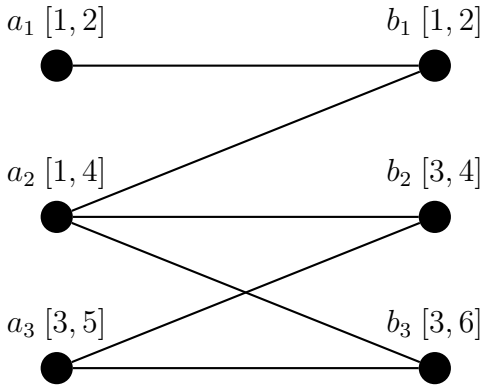
b) $P_{\text{QSTAB}} := \{x \in \mathbb{R}^{V(G)} \mid 0 \leq x_i \leq 1, \sum_{v_i \in C} x_i \leq 1 \quad \forall \text{ cliques } C\}$
$= \text{conv}\{x \in \mathbb{R}^{V(G)} \mid x \text{ is a stable set vector}\}$
i.e. the clique inequalities are sufficient to describe the stable set polytope.

c) the complement of G is perfect.

d) G has neither odd holes nor odd antiholes (complements of odd holes) of size ≥ 5 as induced subgraphs.

The first condition in Theorem 4.2 is the standard definition for perfect graphs. From this it can easily be seen that all bipartite graphs are perfect: the chromatic number and the clique number both are equal to 2 in bipartite graphs—except for the trivial case of a graph without any edges, in which case those quantities are equal to 1. Part *b)* shows why perfect graphs are interesting from an optimization point of view. Namely, if one wants to decide whether some vector is in P_{QSTAB}, it is sufficient to check the trivial inequalities and all inequalities of the form $\sum_{v_i \in C} x_i \leq 1$ where C is a clique in G. Hence, a polynomial time clique separation subroutine, i.e. an algorithm that either confirms $\sum_{v_i \in C} x_i \leq 1$ for all cliques C, or finds a clique for which this inequality is violated, directly yields a polynomial time stable set algorithm by the well-known equivalence of optimization and separation (see e.g. [GLS88]). Such a polynomial time separation is indeed possible for perfect graphs, whereas on general graphs the problem is NP-hard. Together with part *c)* of Theorem 4.2 we also know that for solving the maximum clique problem the stable set inequalities are sufficient. Graphs with property *d)* have been called *Berge graphs*. In 2006, the conjecture that all Berge graphs are perfect was finally proven [CRST06]. Since that time this characterization is known as the *Strong Perfect Graph Theorem*. It characterizes perfect graphs via forbidden induced subgraphs. Later we will use it to show that G_{COMP} is also perfect for the more complicated setting in which the network is a path of arbitrary length.

Example 4.3. Consider the following example given by its compatibility graph G_{COMP}:



One can show that in addition to equations (4.1) and the trivial inequalities $0 \leq z_i \leq 1$ for $i = a_1, a_2, a_3, b_1, b_2, b_3$, the following equations are needed to obtain P:

$$z_{a_3} + z_{b_1} \leq 1 \tag{4.6}$$

$$z_{a_1} + z_{b_2} + z_{b_3} \leq 1 \tag{4.7}$$

We see that (4.6) and (4.7) are the stable set constraints for the stable sets $\{a_3, b_1\}$, $\{a_1, b_2, b_3\}$, which in this case together with $\{a_1, a_2, a_3\}$ and $\{b_1, b_2, b_3\}$ are exactly the maximal stable sets of G_{COMP}.

We now bring together all arguments to give a complete description of P.

Theorem 4.4. *For a path of length two, the stable set constraints of G_{COMP} together with the trivial inequalities $0 \leq z_i \leq 1$ for all i and equations* (4.1) *form a complete description of P.*

Proof: Let $\tilde{P} = \{z \in [0,1]^n \mid z \text{ satisfies (4.1) and } z(S) \leq 1 \ \forall \text{ stable sets } S \subseteq V(G_{\text{COMP}})\}$ be the polytope of points satisfying the constraints mentioned in the theorem.

We have to show that $\tilde{P} = P$, i.e. every vertex of $\tilde{P}$ is integral. We show that $\tilde{P}$ is a face of the well-studied clique polytope

$$P_{\text{CLIQUE}} := \text{conv}\{x \in \mathbb{R}^{V(G)} \mid x \text{ is a clique vector}\}$$

and make use of the fact that a complete description for P_{CLIQUE} is known for perfect graphs, namely $P_{\text{CLIQUE}} = \{x \in \mathbb{R}^{V(G)} \mid 0 \leq x_i \leq 1 \ \forall i, \sum_{v_i \in S} x_i \leq 1 \ \forall \text{ stable sets } S\}$ (see Theorem 4.2).

i) By the above considerations, $\tilde{P}$ is a subset of P_{CLIQUE}, as $\tilde{P}$ satisfies all stable set inequalities. Furthermore, it follows from equations (4.1) that $z(V(G_{\text{COMP}})) = 2$ for all $z \in \tilde{P}$. Thus we conclude that $\tilde{P}$ is also a subset of the restriction of P_{CLIQUE} to inclusion-wise maximal cliques, $P_{\text{CLIQUE}}|_{z(V(G))=2}$.

ii) On the other hand, for $z \in P_{\text{CLIQUE}}$ the constraint $z(V(G_{\text{COMP}})) = 2$ implies (4.1) by construction of G_{COMP}. Hence $P_{\text{CLIQUE}}|_{z(V(G))=2}$ satisfies all constraints of $\tilde{P}$. This means we have $\tilde{P} \supseteq P_{\text{CLIQUE}}|_{z(V(G))=2}$. Finally, equality holds because of i).

Now let $\tilde{z} \in \tilde{P}$. If $\tilde{z}(V(G)) = 2$ and $\tilde{z}$ is a vertex of $\tilde{P}$, it is also a vertex of the restriction $P_{\text{CLIQUE}}|_{z(V(G))=2}$. As $z(V(G)) \leq 2$ is a valid inequality for P_{CLIQUE}, the subset $P_{\text{CLIQUE}}|_{z(V(G))=2}$ is a face of P_{CLIQUE}. We conclude that $\tilde{z}$ is a vertex of P_{CLIQUE} and hence integral. □

The following remark yields another way of viewing the stable set inequalities and it will be helpful later.

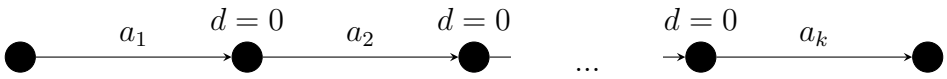
Remark 4.5. We may reformulate the stable set constraint for inclusion-wise maximal stable sets by using (4.1), yielding inequalities of the following structure. It is familiar from *Hall's Matching Theorem*:
For each subset V of vertices of G_{COMP} that belong to the same network arc, the inequality

$$z(V) \leq z(N(V)) \tag{4.8}$$

is valid for P, where $N(V) = \{u \in G_{\text{COMP}} \mid \exists v \in V : (u,v) \in E(G_{\text{COMP}})\}$ denotes the set of neighbors of V. One can show that if we allow the additional option of zero flow on both arcs by relaxing (4.1) to '$\leq$', inequalities (4.8) are stronger than the stable set constraints and yield a complete description of P. A proof can be adapted from that of the above theorem.

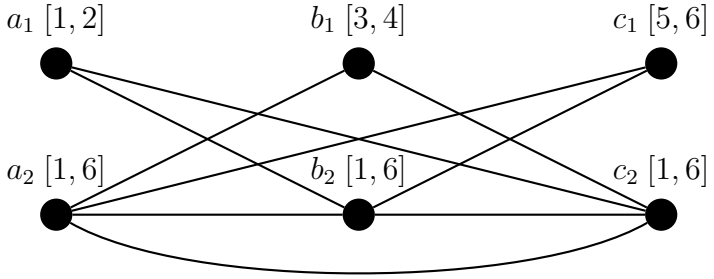
4.2.2 Paths of Arbitrary Length

We now move on to paths of arbitrary length k.



Finding an integral point in P is now equivalent to finding a maximizing set of k intervals (where 'maximum' is defined by any given linear objective) such that each interval belongs to a different network arc and their intersection is nonempty—or compatible with the demands if they are $\neq 0$. In the latter case, a similar shifting of intervals as explained in Subsection 4.2.1 is possible for constructing the compatibility graph. Again as in the previous subsection, we rephrase this task by means of the compatibility graph. Having a k-clique in this graph is obviously necessary. However, it is also sufficient for a point to lie in P, although G_{COMP} is only able to represent pairwise conflicts. The reason is that due to convexity it cannot happen that a family of intervals is incompatible although each pair is. This is basically *Helly's Theorem* [Hel23] in dimension 1. In other words, G_{COMP} still detects all possible variable conflicts, also for the case of paths of length $k > 2$.

Example 4.6. The following graph shows how G_{COMP} may look like for a path consisting of three network arcs.



We note that it contains edges for non-neighboring network arcs, e.g. $a_1 - c_2$ or $a_2 - c_2$.

A New Class of Perfect Graphs

Unfortunately, the compatibility graph for paths of arbitrary length doesn't trivially belong to any well-known class of perfect graphs. Therefore, we develop a new graph class, designed for G_{COMP}. Before moving on to our definition of *partition-chordal graphs*, we will motivate it based on two graph classes that are well known to be subclasses of perfect graphs. Recall the following definitions:

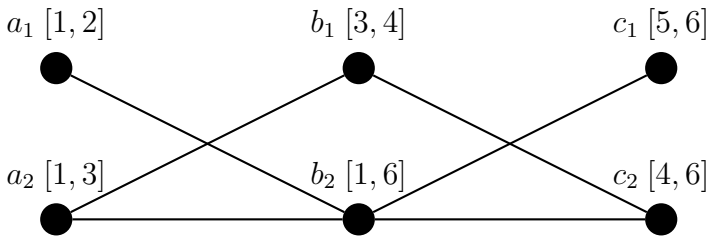
Definition 4.7 (Interval Graphs)**.** An undirected graph G is called *interval graph* if and only if each vertex v of G can be identified with an interval $I_v \subseteq \mathbb{R}$ such that G has an edge (u, v) if and only if $I_u \cap I_v \neq \emptyset$, i.e. G can be realized as an intersection graph of a family of intervals in $\mathbb{R}$.

This definition has obvious similarities with the construction of our compatibility graph G_{COMP}. In fact, G_{COMP} would be an interval graph if it wasn't for the fact that nodes of G belonging to the same partition are not allowed to have an edge between them, even if their intervals intersect. We could directly generalize this class of graphs to have G_{COMP} covered (see Remark 4.11 below), though it will turn out to be possible—and more convenient—to operate on a superclass of interval graphs, namely *chordal graphs*.

Definition 4.8 (Chordal Graphs)**.** An undirected graph G is called *chordal* (or *triangulated*) if and only if every cycle of length ≥ 4 has a *chord*, i.e. there are no induced cycles of length ≥ 4 in G.

It is known that all interval graphs are chordal [Gol80]. However, G_{COMP} is not chordal in general. The following is a counterexample.

Example 4.9. Consider a slight modification of the graph from Example 4.6.



Compared to Example 4.6, a_2 has been changed from $[1,6]$ to $[1,3]$ and c_2 from $[1,6]$ to $[4,6]$. As a consequence, the edge from a_2 to c_2 is missing (among others) and a chordless 4-cycle $a_2 - b_2 - c_2 - b_1 - a_2$ exists. This shows that G_{COMP} is not chordal in general.

The following definition is designed to capture the essential extension of chordal graphs needed to include G_{COMP}, while we will still be able to prove perfectness for the resulting graph class.

Definition 4.10 (Partition-Chordal Graphs)**.** An undirected graph G is called *partition-chordal (with partition order k)* if and only if it has a k-partition and a set

$$\tilde{E} \subseteq \{(u,v) \mid u \neq v \text{ belong to the same partition}\}$$

of edges such that adding all edges in $\tilde{E}$ to the original edge set yields a chordal graph. The elements of $\tilde{E}$ will subsequently be called *fill edges*.

Remark 4.11. In the same vein, we may also define *partition-interval graphs* as graphs that have a k-partition and a set $\tilde{E} \subseteq \{(u,v) \mid u \neq v \text{ belong to the same partition}\}$ of edges such that adding all edges in $\tilde{E}$ to the original edge set yields an interval graph. Those definitions may inspire future graph-theoretic research (e.g. regarding recognizability). Though for our purposes we focus on the more general class of partition-chordal graphs.

The following two results, Lemma 4.12 and Theorem 4.13 are of course essential for Definition 4.10 to fulfill its purpose:

Lemma 4.12. *For paths of length k, the compatibility graph is partition-chordal with partition order k.*

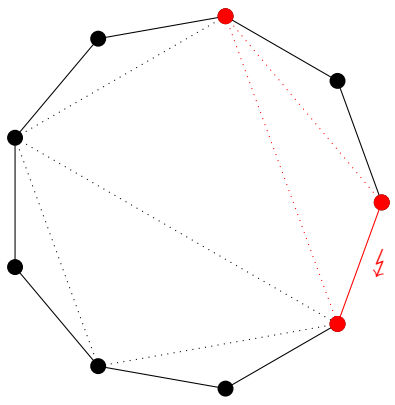
Proof: Indeed, choose the partitions to consist of all vertices that belong to the same network arc. Then the graph containing all edges in the set $\{(u, v) \mid u \neq v$ belong to the same partition$\}$ has an interval graph as a spanning subgraph by construction of G_{COMP}. Since interval graphs are chordal, the claim follows. □

In the following, *partition* will always refer to this canonical k-partition, unless specified otherwise.

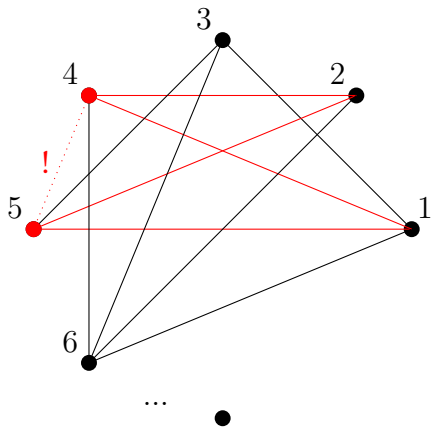
Theorem 4.13. *A graph that is partition-chordal is also perfect.*

Proof: Let G be a partition-chordal graph with vertex set V, a fixed partition and a set of fill edges $\tilde{E}$. Using the Strong Perfect Graph Theorem we have to show that G has neither odd holes nor odd antiholes of size ≥ 5 as induced subgraphs.

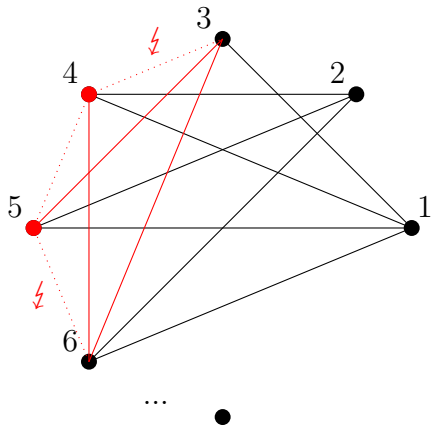
Assume G has an odd hole $C_l, l \geq 5$ as an induced subgraph. Let $\tilde{G} = (V, E(G) \cup \tilde{E})$ be the graph that results from adding all edges in $\tilde{E}$ to G. As $\tilde{G}$ is chordal by assumption, C_l must have a chord that is an element of $\tilde{E}$. This fill edge subdivides C_l into two shorter cycles in $\tilde{G}$. By iteratively applying chordality, we reach a triangulation of C_l into $l - 2$ triangles. Since l is odd there has to be a triangle using an odd number of arcs of C_l, namely 1, while the two remaining arcs are in $\tilde{E}$. But these two fill edges imply that all three vertices of the triangle lie in the same partition, contradicting the fact that there is an edge between two of them (see the figure below for an illustration).



Now assume G has an odd antihole $\bar{C}_l$ as an induced subgraph. Since an antihole of size 5 is isomorphic to a 5-cycle, we may assume $l \geq 7$. Given an anticlockwise numbering of the vertices with labels 1 to l, consider the 4-cycle consisting of the vertices $1, 4, 2$ and 5 (shown in red in the figure below).



Since $\tilde{G}$ is chordal, we conclude that there has to be a fill edge between two vertices of $\bar{C}_l$ that are neighbors with respect to the numbering, w.l.o.g. between 4 and 5. Using this arc we can find another 4-cycle which uses a fill edge (see the figure below, in which this cycle consists of the vertices 3, 5, 4 and 6).



Again, by chordality one of the chords must be a fill edge. However, this implies that there exist three out of the four vertices of the 4-cycle (3, 5, 4 and 6) that belong to the same partition, which is a contradiction to the fact that edges in $\bar{C}_l$ can not connect vertices belonging to the same partition. This contradiction implies that G also cannot have an odd antihole as an induced subgraph.

We therefore conclude that G is perfect. □

Remark 4.14. We may rephrase Definition 4.10 in a way that puts emphasis on the construction of partition-chordal graphs: Start with a chordal graph and color it in an arbitrary way. If we then remove all edges from the graph whose endpoints have the same color, the result will be a partition-chordal—and thus a perfect—graph. In fact, by definition all partition-chordal graphs can be constructed that way.

To the best of the author's knowledge, the class of partition-chordal graphs has been first introduced in [LM16] and has not been studied before. A detailed graph-theoretic classification lies outside the main focus of this work. Still, based on the

above deliberations we may contribute the following to the classification of partition-chordal graphs as a subclass of perfect graphs:

Proposition 4.15 (Some Classification Statements on Partition-Chordal Graphs)**.** *The class of partition-chordal graphs is*

a) antihole-free (also for even antiholes), and therefore a strict subclass of perfect graphs.

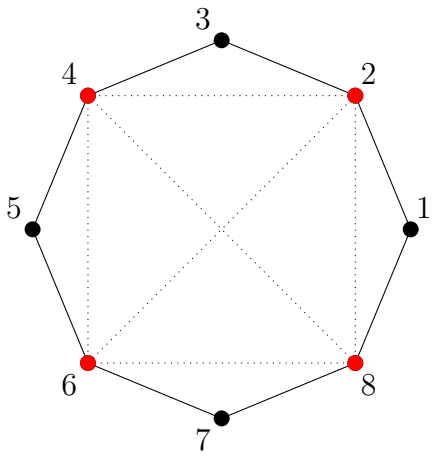
b) odd-hole-free, but not hole-free.

c) a strict superclass of chordal graphs.

Proof:

a) This claim follows from the fact that for the part of the proof of Theorem 4.13 on odd antiholes, we did not need to utilize that the antihole $\bar{C}_l$ has an odd number of nodes. Hence, we may give an even antihole of size ≥ 6 as an example of a perfect graph that is not partition-chordal.

b) That partition-chordal graphs are odd-hole-free directly follows from Theorem 4.13, and has been shown explicitly in the proof thereof. We show that this proof cannot be extended to even holes by proving that cycles C_l of even length l are indeed partition-chordal: Let $\tilde{V}$ denote the set of vertices at even positions in C_l. Now let $\tilde{V}$ be a partition and partition all remaining vertices (that form a stable set in C_l) in an arbitrary way. Let the set of fill edges $\tilde{E}$ consist of all connections between vertices in $\tilde{V}$. For an illustration see the figure below, in which the partition $\tilde{V}$ (shown in red) consists of the vertices 2, 4, 6 and 8; edges in $\tilde{E}$ are displayed as dotted lines.



Now C_l together with $\tilde{E}$ forms a chordal graph. This can be seen from the fact that every cycle of length ≥ 4 in this graph has to use at least three vertices from $\tilde{V}$, and therefore must have a chord as the three vertices from $\tilde{V}$ form a clique. Thus, C_l is a partition-chordal graph.

c) This is already implied by the existence of a compatibility graph that is not chordal, see Example 4.9.

□

Due to Theorem 4.13, we know that we can solve the maximum clique problem in polynomial time on the compatibility graph, and therefore also the separation of stable set constraints. We might ask how fast the stable set separation can actually be performed. In their book [GLS88], Grötschel, Lovász and Schrijver derive the polynomial time result via the so called *theta body* of a graph. It is motivated by Lovász's theta-bound on the Shannon-capacity of a graph. In this theta body, separation is possible in polynomial time. Furthermore, it turns out to be a polytope if the graph is perfect, and its facets can be shown to be equivalent to clique inequalities. The runtime of the separation lies in $\mathcal{O}(n^4)$, which the algorithms spends mainly with calculating determinants in order to check positive semidefiniteness of a matrix.

However, in practice one might prefer either heuristics (if missing a small number of possible cutting planes seems affordable) or exact method that have exponential worst-case time complexity, like an auxiliary MIP (if separation subproblems stay reasonably small). In the following part of this section, we will see that for more structured problems with non-overlapping intervals, we do not need to run a separation routine based on finding minimum stable sets. Instead, we can restrict ourselves to polynomially many stable set inequalities that can be explicitly listed and precomputed.

Corollary 4.16 (Generalization of Theorem 4.4 to Paths of Arbitrary Length)**.** *For paths of arbitrary length, the stable set constraints of G_{COMP} together with the trivial inequalities and equations* (4.1) *form a complete description of P.*

Proof: Since the compatibility graph is a perfect graph for all paths, the claim can be proven analogously to that of Theorem 4.4. □

The question comes up, on how many partitions the stable set inequalities have to be defined. In particular, it is interesting to study whether it is sufficient to include only the stable set constraints defined on nodes from just two partitions per constraint. Although it turns out that this is not true in general, it is possible to identify situations in which those inequalities suffice. We first rewrite those inequalities as in Remark 4.5, in the form

$$z(V) \leq z(N_U(V)), \tag{4.9}$$

where U is some partition of G_{COMP}, V a set of vertices of the same partition different from U, and $N_U(V)$ denotes the set of neighbors of V that belong to partition U. This notation will be helpful for the proof of the next theorem. It gives a criterion for the inequalities involving only two partitions being already sufficient for a complete description of P.

Theorem 4.17. *If for each ordered pair (I_1, I_2) of intervals that belong to the same partition the set $I_1 \backslash I_2$ is connected, then the inequalities of type* (4.9) *together with the trivial inequalities and equations* (4.1) *form a complete description of P.*

Remark 4.18. The criterion in the above theorem is satisfied if any two intervals associated to the same network arc must not 'strictly' contain each other in the sense that the larger one has larger upper bound as well as a smaller lower bound. If instances result from piecewise linearization, this assumption is usually fulfilled because the model is redundant otherwise.

Proof (of Theorem 4.17)**:** The proof follows from Corollary 4.16, if we can show that all stable set inequalities for G_{COMP} are implied by inequalities (4.9). The following is illustrated in Figure 4.1. Let $S = \{s_1, ..., s_l\} \subseteq V(G_{\text{COMP}})$ be a stable set in G_{COMP} and $\phi : V(G_{\text{COMP}}) \to \{1, ..., k\}$ a partition map onto the k partitions. By abuse of notation, we will later in this proof identify the partitions with their corresponding number in $\{1, ..., k\}$. Since each element of S is associated to an interval, the assumption of the theorem together with S being a stable set allows us to define a total ordering of S, where $s_i = [l_i, u_i] \leq s_j = [l_j, u_j]$ if and only if $l_i \leq l_j$ and $u_i \leq u_j$. We may assume that S is ordered increasingly, i.e. for $s_i, s_j \in S$ with $i < j$ we have $s_i \leq s_j$. If $\phi(s_i) \neq \phi(s_{i+1})$ for some i, we say that S has a *partition change*. We show the claim by induction on the number of partition changes. If this number is 0, there is nothing to show since $z(S) \leq 1$ is already implied by equation (4.1) for the partition that S belongs to.

Now let $S = \{s_1, ..., s_l\}$ have d partition changes. Let $t = \max_{i=1,...,l}\{\phi(s_{i-1}) \neq \phi(s_i)\}$, and let $T = \{s_t, ..., s_l\}$. So all elements of T belong to the same partition and s_{t-1} belongs to a different one. Consider $N_{\phi(s_{t-1})}(T)$. This set does not contain s_{t-1}, since S is a stable set. But in addition, we can show that $(S \backslash T) \cup N_{\phi(s_{t-1})}(T)$ is also a stable set of G_{COMP}: Assume, on the contrary, there exists $c \in N_{\phi(s_{t-1})}(s_m) \cap N_{\phi(s_{t-1})}(T)$ for some $s_m \in S$. We deduce that c and s_{t-1} both belong to the same partition. However, they contradict our assumption, because the difference of intervals $c \backslash s_{t-1}$ has to contain elements below as well as above s_{t-1}. Therefore, $(S \backslash T) \cup N_{\phi(s_{t-1})}(T)$ is a stable set with $d - 1$ partition changes. Finally, we have

$$z(S) = z(S \backslash T) + z(T) \leq z(S \backslash T) + z(N_{\phi(s_{t-1})}(T)) = z((S \backslash T) \cup N_{\phi(s_{t-1})}(T)) \leq 1,$$

using the induction hypothesis for the last step. □

From this proof, we can also deduce a slightly more general theorem for the case in which the assumption of Theorem 4.17 does not hold.

Theorem 4.19. *Let $\tilde{\mathcal{I}}$ be the set of intervals I for which there exists another interval I' that belongs to the same partition and strictly contains I, i.e. $I' \backslash I$ is not connected. Then, all facets of P besides the trivial inequalities and equations* (4.1) *are either of type* (4.9) *or contain at least one element from $\tilde{\mathcal{I}}$ with nonzero coefficient.*

Proof: The statement can be obtained from the proof of Theorem 4.17 above: We see that the induction step on the number of color changes works as long as S does not

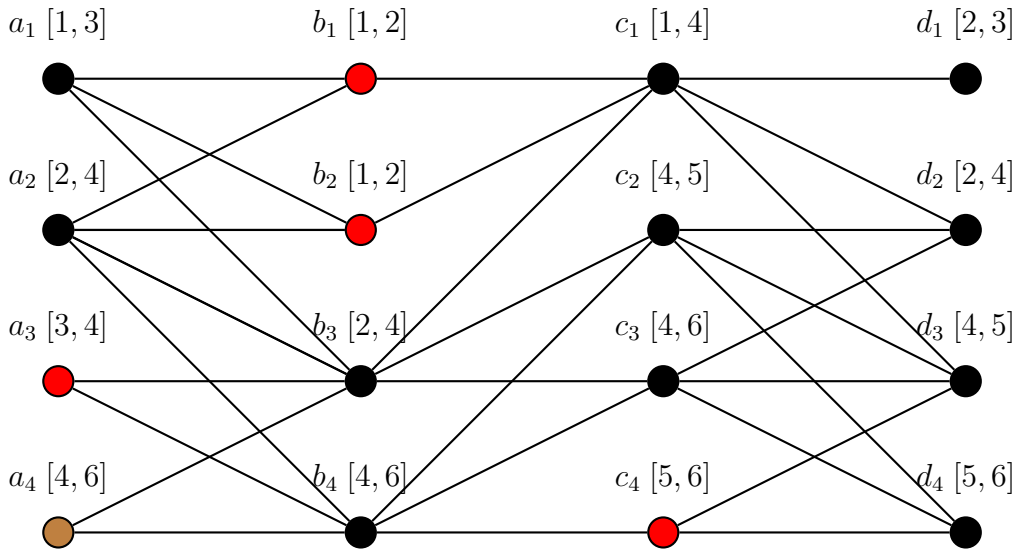
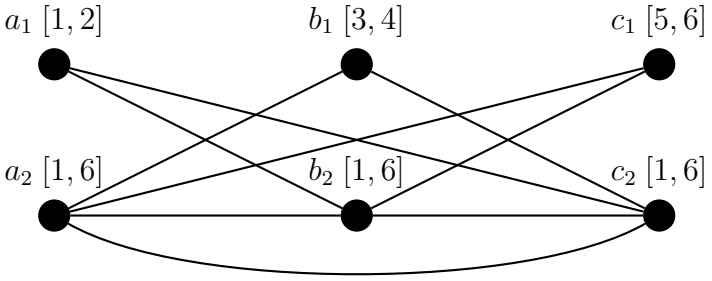


Figure 4.1: Illustration of the proof of Theorem 4.17. Consider the example instance shown in this figure with the partitions $A = \{a_1, a_2, a_3, a_4\}, B = \{b_1, b_2, b_3, b_4\}, \ldots$ (edges of G_{COMP} between non-adjacent network arcs are not shown for the sake of clarity). Let the stable set $S = \{b_1, b_2, a_3, c_4\}$ consist of the red vertices. Being ordered, S has two partition changes. In the notation of the proof we have $T = \{c_4\}$ and $N_{\phi(s_{t-1})}(T) = N_A(\{c_4\}) = \{a_4\}$. Replacing $\{c_4\}$ by $\{a_4\}$ yields a set S' with only one partition change. S' also has to be a stable set, since by the theorem's assumption a_4 is not allowed to intersect with b_2 or b_1 as it would then surround a_3. In addition, the stable set inequality of S' implies that of S.

contain an element of $\tilde{\mathcal{I}}$. As this step finds another stable set whose stable set condition, together with (4.9), implies that of S, we know that S does not define a facet of P. This means that stable sets involving more than two partitions can only define facets if they contain an element from $\tilde{\mathcal{I}}$. □

The next example shows that in general, inequalities (4.9) are not sufficient for a complete description of P.

Example 4.20. Consider again the graph from Example 4.6 with three network arcs, given by the following compatibility graph.



We can see that the point $(z_{a_1}, z_{a_2}, z_{b_1}, z_{b_2}, z_{c_1}, z_{c_2}) = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ satisfies all inequalities of type (4.9). However, it does not lie in $P = \text{conv}\{(0,1,0,1,0,1)\ , (1,0,0,1,0,1), (0,1,1,0,0,1), (0,1,0,1,1,0)\}$, which in contrast satisfies the stable set inequality $z_{a_1} + z_{b_1} + z_{c_1} \leq 1$. This is well in accordance with Theorem 4.17, since the intervals b_2 and b_1 violate its assumption. Moreover, as predicted by Theorem 4.19, the inequality $z_{a_1} + z_{b_1} + z_{c_1} \leq 1$ does involve a variable that is associated to a 'strictly contained' interval, namely z_{b_1}.

Under the assumption of Theorem 4.17, we can further classify the inequalities that are necessary for determining a complete description. This then yields complete description of P that is of polynomial size.

Definition 4.21. Let U be a partition of G_{COMP} and assume the condition of Theorem 4.17 is fulfilled. This means that for each ordered pair (I_1, I_2) of intervals that belong to the same partition, the set $I_1 \backslash I_2$ is connected. Similarly as in the proof of Theorem 4.17, we define a total ordering of intervals within each partition, where $I_1 = [l_1, u_1] \leq I_2 = [l_2, u_2]$ if and only if $l_1 \leq l_2$ and $u_1 \leq u_2$. We note that two incomparable intervals would violate the above assumption. We call a set $V \subseteq U$ *upwards connected* if

$$I_1 \in V, I_1 \leq I_2 \Rightarrow I_2 \in V \quad \forall\, I_1, I_2 \in U.$$

This means that V is of the form $V = \{I \in U \mid I \geq I_{\text{min}}\}$ for some particular I_{min}. Analogously, we call $V \subseteq U$ *connected* if

$$I_1, I_3 \in V, I_1 \leq I_2 \leq I_3 \Rightarrow I_2 \in V \quad \forall\, I_1, I_2, I_3 \in U.$$

This means that V is of the form $V = \{I \in U \mid I_{\text{min}} \leq I \leq I_{\text{max}}\}$ for some particular $I_{\text{min}}, I_{\text{max}} \in U$.

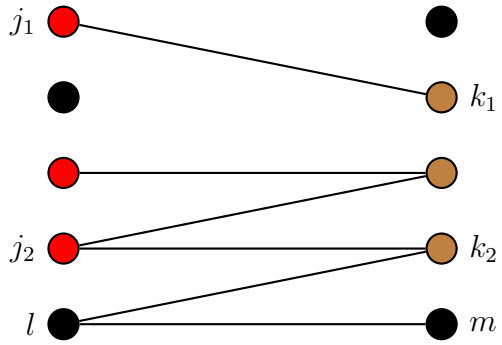
Theorem 4.22. *If for each ordered pair (I_1, I_2) of intervals that belong to the same partition, the set $I_1 \backslash I_2$ is connected, then a complete description of P is given by the trivial inequalities, equations* (4.1)*, together with the inequalities* (4.9)*, where V is upwards connected.*

Proof: We have to show that all inequalities (4.9) are implied by the ones where V is upwards connected. So let V be a subset of intervals of some partition $A = \{a_1, \dots a_l\}$ and $B \neq A, B = \{b_1, \dots b_m\}$ be another partition (indexed according to the ordering introduced in Definition 4.21). Depending on the structure of $N_B(V)$ we distinguish the following cases, which build upon each other:

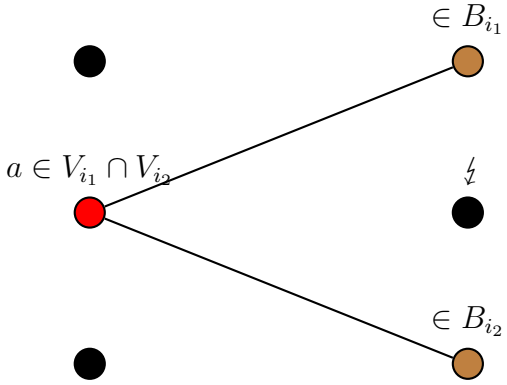
a) $N_B(V)$ is connected with $N_B(V) = \{b_i \mid k_1 \leq i \leq k_2\}$. Let j_1, j_2 be the indices of the minimal and the maximal interval of V, respectively. Hence $V \subseteq \{a_i \mid j_1 \leq i \leq j_2\}$ (see Figure 4.2a) and we have

$$z(V) \leq z(\{a_{j_1}, ..., a_{j_2}\}) = z(\{a_{j_1}, ..., a_l\}) - z(\{a_{j_2+1}, ..., a_l\})$$

Now $\{a_{j_1}, ..., a_l\}$ is upwards connected, so we may use (4.9) to obtain



(a) Illustration of the notation used for the proof of Theorem 4.22. A is on the left with V shown in red, whereas B is on the right hand side and $N_B(V)$ in shown in brown.



(b) This situation cannot happen, as node a would have to be connected to the middle node in B as well, which contradicts the definition of the B_i.

Figure 4.2: Example illustrations for the proof of Theorem 4.22.

$$\cdots \leq z(\{b_{k_1}, ..., b_m\} - z(\{a_{j_2+1}, ..., a_l\})$$

Also $\{b_{k_2+1}, ..., b_m\}$, i.e. the set of intervals 'above' $N_B(V)$, is also upwards connected and its neighborhood $N_A(\{b_{k_2+1}, ..., b_m\})$ is a subset of $\{a_{j_2+1}, ..., a_l\}$, hence $z(\{a_{j_2+1}, ..., a_l\}) \geq z(N_A(\{b_{k_2+1}, ..., b_m\}))$. Using this and applying (4.9) yields

$$\cdots \leq z(\{b_{k_1}, ..., b_m\} - z(\{b_{k_2+1}, ..., b_m\}) = z(\{b_{k_1}, ..., b_{k_2}\}) = z(N_B(V))$$

.

b) $N_B(V)$ is not connected and splits into the connected sets $B_1, ..., B_r$. Then we find sets $V_1, \ldots, V_r$ such that $V = \bigcup_{i=1}^r V_i$ and $N_B(V_i) = B_i \quad \forall\, i = 1, \ldots, r$. We claim that V necessarily is the *disjoint* union $V = \sum_{i=1}^r V_r$: Assume on the contrary that there is $a \in V_{i_1} \cap V_{i_2}$. Then a has neighbors in B_{i_1} as well as B_{i_2}. But that means that by construction every interval between those neighbors also has to be a neighbor of a, which contradicts the fact that $B_{i_1} \cup B_{i_2}$ is not connected (see Figure 4.2b).

So $V = \sum_{i=1}^r V_r$ and it follows from a) that

$$z(V) = \sum_{i=1}^{r} z(V_i) \leq \sum_{i=1}^{r} N_B(z(V_i)) = z(N_B(V))$$

This shows that inequalities (4.9) are implied and hence the proof is completed. □

Remark 4.23. In particular, this yields a polynomial size complete description of P, as there are only linearly many sets V per network arc that have to be considered for constructing inequalities (4.9). The same reasoning also applies for 'downwards connected' sets, when defined analogously.

4.2.3 Transferability to a Formulation According to the Incremental Method

In case of no overlaps between intervals belonging to the same partition, we are in the standard situation of piecewise linear modeling. In particular, the costs associated to sending a certain amount of flow is a piecewise linear function of the flow. As noted in Section 4.1, the model for P used so far uses (4.3) and therefore relies on a modeling according to the Multiple Choice Method or the Convex Combination Method (see Subsections 2.2.1 and 2.2.2). We will now see that our results can also be applied if the Incremental Method is used, and how the facets look like in that case. MCM and CCM are more flexible when we think of overlapping intervals. However, as already mentioned in Section 2.2, the δ-Method has been superior for certain applications, e.g. for optimization problems on gas networks [Gei11b, CPSM14], and is very widely used in practice.

Let an interval $[l, u]$ for the flow value and breakpoints $B_1 = l, B_2, \ldots, B_n, B_{n+1} = u$ that divide the interval in n subintervals be given. For describing the polytope, in addition to the binary z-variables the Incremental Method uses continuous $[0, 1]$-variables δ_i, and the constraint

$$q = B_1 z_1 + \sum_{i=1}^{n} (B_{i+1} - B_i)\delta_i$$

together with the *filling condition* constraints $z_i \geq \delta_i, i = 1, \ldots, n$ and $\delta_i \geq z_{i+1}, i = 1, \ldots, n-1, \delta_n \geq 0$. A piecewise linear function f of q can then be written as

$$f(q) = f(B_1) z_1 + \sum_{i=1}^{n} (f(B_{i+1}) - f(B_i))\delta_i$$

(cf. Subsection 2.2.3). If we don't want to allow the extra option of q being 0, we set $z_1 = 1$. As before with P, we consider the polytope after projection to the z-variables. Let P_δ denote the convex hull of this projected set of feasible points, in formulas

$$P_\delta = \operatorname{conv}\{z_\delta \in \{0,1\}^n \mid \exists q \in \mathbb{R}^m, \delta \in \mathbb{R}^n : z_{a_1} = 1 \; \forall \text{ arcs } a, \text{ (4.2) , filling condition}\}.$$

By construction, the z-variables are decreasing, where a 'jump' $z_i = 1, z_{i+1} = 0$ means that the flow q lies in the i-th interval $[B_i, B_{i+1}]$. Given intervals $I_a = [l_a, u_a]$ and $I_b = [l_b, u_b]$ with $l_a < l_b$ associated to different arcs A and B of the path, we see that

$$z_{I_a} \geq z_{I_b} \tag{4.10}$$

is a valid inequality, since in the case of violation, i.e. $z_{I_a} = 0, \; z_{I_b} = 1$, we can conclude $q_a \leq l_a, q_b \geq l_b$ and hence $q_a < q_b$, which contradicts flow conservation. This reasoning can be extended easily for the case of nonzero demand of the middle nodes of the path.

We show next that these simple inequalities are the equivalent of inequalities (4.9) for the modeling according to the δ-method. Furthermore, we show that they are sufficient for a complete description of P_δ. Note that for this subsection, we assume that we are in the standard setting for using the δ-Method—in particular, for each ordered pair of intervals belonging to the same partition, their intersection is at most a single point.

Theorem 4.24. *For paths of arbitrary length, the inequalities of type* (4.10) *together with the trivial inequalities and the filling inequalities $z_1 \geq z_2 \geq \cdots \geq z_n$ for each partition form a complete description of P_δ.*

Proof: The proof is based on the well-known linear bijection from P_δ to the corresponding P (associated with the familiar modeling used so far), which has been introduced in Chapter 2, Subsection 2.2.3. We denote the transformation by T. It maps z to $z_i - z_{i+1}, i = 1, \ldots, n-1; z_n := z_n$ and has the inverse $T^{-1} : z_i \mapsto \sum_{j=i}^{n} z_i$. The existence of such a transformation implies a one-to-one correspondence between the extreme points of P and P_δ. Following [Vie15], we obtain a complete description of P_δ by taking a complete description of P and for each inequality, replacing every occurrence of a z-variable by $T(z)$.

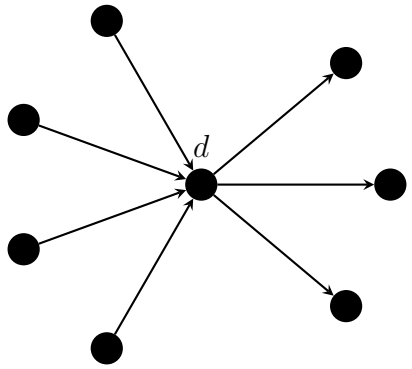
So it only remains to confirm that the inequalities from Theorem 4.22 map to inequalities (4.10). Let $A = \{a_1, \ldots a_l\}$ be some partition and $B \neq A, B = \{b_1, \ldots b_m\}$ be another partition, again indexed according to the ordering introduced in Definition 4.21. Also, let $V = \{a_i \mid i \geq j_1\}$ for some j_1 be upwards connected with k_1 being the minimal interval index of $N_B(V)$. We may assume without loss of generality that $N_B(V)$ is also upwards connected (as elements in $\{b_{k_1}, \ldots, b_m\} \backslash N_B(V)$ have no neighbors in A. They are therefore forced to 0 anyway and may be added to the right hand side of (4.9) without weakening the inequality). Then $z(V) \leq N_B(V)$ mapped by T forms telescope sums on both sides and yields inequalities (4.10). □

We have seen that our complete description derived for linearization methods using (4.3) can be transferred to the polytope of the δ-Method. It turned out that the two polytopes are essentially the same up to a linear bijection, where inequalities (4.10) are sparser than those of type (4.9).

4.2.4 Junctions

In this subsection we consider star graphs with $k > 2$ arcs, where k_1 of them are inflow arcs and $k_2 = k - k_1$ are directed outwards. Let the central node v_c have demand d. We again use the setting that follows (4.3).

Graph theoretic considerations about the compatibility graph are not applicable here in the same way as for paths, as it is not enough to display all binary conflicts. In general, G_{COMP} will be extremely dense if k is not too small as fixing a flow interval on two network arcs is likely to still allow for several feasible solutions. If we project the linear relaxation of the original feasible set (including the q-variables) directly onto the



z-variables, besides the trivial inequalities and inequalities (4.1), we get the following two constraints:

$$\sum_{a\in\delta^-(v_c)}\sum_{i\in\mathcal{I}_a} l_i z_i \quad + \quad d \qquad \leq \quad \sum_{a\in\delta^+(v_c)}\sum_{i\in\mathcal{I}_a} u_i z_i \tag{4.11}$$

$$\sum_{a\in\delta^-(v_c)}\sum_{i\in\mathcal{I}_a} u_i z_i \quad + \quad d \qquad \geq \quad \sum_{a\in\delta^+(v_c)}\sum_{i\in\mathcal{I}_a} l_i z_i \tag{4.12}$$

They are easy to interpret: A point is feasible only if the minimal inflow, given by $\sum_{a\in\delta^-(v_c)}\sum_{i\in\mathcal{I}_a} l_i z_i$, plus the demand d of the central node is at most as large as the maximal outflow $\sum_{a\in\delta^+(v_c)}\sum_{i\in\mathcal{I}_a} u_i z_i$. The second inequality can be interpreted analogously.

In this setting a complete description seems out of reach from our experience. In contrast to paths of degree-2-nodes, the problem is NP-hard.

Theorem 4.25. *On star graphs, deciding whether P is empty is NP-hard, even if there are at most 2 intervals per arcs.*

Proof: We show the theorem by a polynomial-time reduction from the (weakly) NP-hard problem PARTITION: Given integers $c_1, ..., c_n$, determine whether there is a subset $I \subseteq \{1, ..., n\}$ of indices such that $\sum_{i\in I} c_i = \sum_{i\notin I} c_i$, or in other words $\sum_{i\in I} c_i = \frac{1}{2}C$ where $C := \sum_{i=1}^n c_i$.

Let I_1 be an instance of PARTITION, w.l.o.g. $c_i \neq 0 \ \forall i = 1, ..., n$. We construct an instance of P on a star graph as follows. We create n arcs leaving the central vertex of the star. Each arc has two intervals consisting of one point each, namely $\{0\}$ and $\{c_i\}$. The demand d at the central node is chosen as $\frac{1}{2}C$ (see Figure 4.3).

We easily see that a feasible configuration of intervals corresponds to a subset $I \subseteq \{1, ..., n\}$ such that $\sum_{i\in I} c_i = \frac{1}{2}C$, and vice versa. □

Remark 4.26. The technique used above only works if intervals are not required to be the result of subdividing a larger interval. Still, even in that case a similar construction shows that optimizing over P is NP-hard. However, if we restrict ourselves to star graphs with a constant number k of arcs, optimizing over P can be done in polynomial time: we know that every integral solution has exactly k of the z-variables set

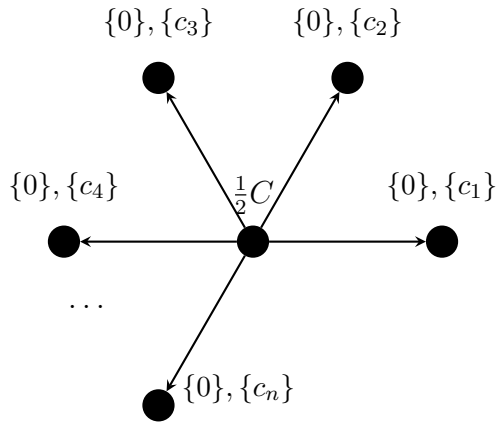


Figure 4.3: Illustration of the reduction from PARTITION in the proof of Theorem 4.25

to 1. Therefore, we can check all combinations in polynomial time if k is considered constant. Fixing all z-variables of course leaves a very easy flow problem.

In the following, we will present a simple class of valid inequalities that we will separate heuristically in Section 4.3.

Consider an integer vector z satisfying equations (4.1) such that for each arc a, the index j_a specifies which component of z is set to 1 on this arc. In particular, for all arcs a, we have $z_{a_{j_a}} = 1$. Furthermore, let z violate (4.11), i.e.

$$\sum_{a \in \delta^-(v_c)} l_{a_{j_a}} + d > \sum_{a \in \delta^+(v_c)} u_{a_{j_a}}.$$

Infeasibility of z allows us to obtain the valid inequality

$$\sum_{a \in \delta^-(v_c)} z_{a_{j_a}} + \sum_{a \in \delta^+(v_c)} z_{a_{j_a}} \leq k - 1, \tag{4.13}$$

cutting off the infeasible point. Note that they can also be obtained as knapsack cover inequalities. To see this, we substitute $\bar{z}_i$ for $1 - z_i$ in (4.11) and obtain the knapsack inequality

$$\sum_{a \in \delta^-(v_c)} \sum_{i \in \mathcal{I}_a} l_i z_i + \sum_{a \in \delta^+(v_c)} \sum_{i \in \mathcal{I}_a} u_i \bar{z}_i + d \leq \sum_{a \in \delta^+(v_c)} \sum_{i \in \mathcal{I}_a} u_i.$$

Now any knapsack cover consisting of one z-variable and $n_a - 1$ of the $\tilde{z}$-variables per arc translates to (4.13), where n_a denotes the number of intervals on arc a.

Using the fact that at most one z-variable per arc can be set to 1, we can strengthen (4.13) by summing over all intervals with larger lower bound than the interval corresponding to $z_{a_{j_a}}$ in the first sum as well as over all intervals with smaller upper bound in the second. This leads to

$$\sum_{a \in \delta^-(v_c)} \sum_{i=j_a}^{n_a} z_{a_i} + \sum_{a \in \delta^+(v_c)} \sum_{i=1}^{j_a} z_{a_i} \leq k - 1, \tag{4.14}$$

where again n_a denotes the index of the topmost interval of arc a, assuming intervals are numbered appropriately.

We separate these inequalities later in Section 4.3. Note that they are valid for any structure of intervals, including overlapping intervals on the same network arc. For the special case of instances resulting from piecewise linear modeling (as for Subsection 4.2.3), see [ZdF13] for further classes of valid inequalities derived from knapsack inequalities.

Remark 4.27. This class of inequalities is also transferable to a formulation according to the δ-method. The equivalent cutting planes to inequality (4.14) then read

$$\sum_{a \in \delta^+(v_c)} z_{a_{j_a}} + \sum_{a \in \delta^-(v_c)} (1 - z_{a_{j_a+1}}) \leq k - 1.$$

This is derived by using the same transformation as for Theorem 4.24.

4.3 Computational Results

4.3.1 Separation Algorithms

Three versions of IP solution methods have been implemented based on using callbacks in Gurobi [Gur17]. The algorithms differ in terms of the class of inequalities they separate. The new methods presented here are compared to using Gurobi without separating callbacks. As in the previous chapter, this reference is denoted by MIP in the following.

- The first method separates the inequalities from the complete description of P when defined on paths of network arcs (see Corollary 4.16). It identifies all suitable sub-paths of degree-two-nodes in the network and constructs the corresponding compatibility graphs—which only has to be done once. Theorem 4.22 allows to pre-compute a complete description for each of the detected sub-paths. This description is quadratic in the path's length and linear in the number of intervals per arc. We don't add all those constraints right from the start as in practice many of them are redundant. Instead, we use a separation callback at every 50th branch-and-bound node that finds all violated inequalities and adds them to the model. The callback is called at most 100 times.

  We call this method PATHCUT.

- We may also separate inequalities (4.14) from Subsection 4.2.4, which is applicable at every network node. The separation is done heuristically. Inequalities (4.14) are derived from an infeasible combination of intervals. Also, we want any generated cutting plane to cut off the current fractional solution. First, we construct a candidate for this combination of intervals from the current (branch-and-bound) node relaxation. For every network arc incident to the network node

considered, we simply take the interval whose z-variable has maximal (fractional) value. We then use a local search procedure in order to improve this candidate in terms of the two criteria, infeasibility of the combination and violation of the resulting constraint by the current node relaxation. This cutting plane method is called FORKCUT and is used also at every 50th branch-and-bound node.

- Finally, CUT calls the separation routines of both PATHCUT and FORKCUT.

4.3.2 Benchmark Instances and Test Environment

The different methods are evaluated on benchmark instances for P based on two different sets of underlying network topologies. These are random scale-free networks according to an underlying preferential attachment model [AB02], and the original network topology of a real-world gas network. For all test sets additional input data is generated at random. This includes the vector d of demands as well as the initial arc capacities c. Capacities were scaled in such a way that feasibility of all instances is guaranteed. We then chose a random partition of the interval $[-c_a, c_a]$ into a given constant number of intervals for each network arc. This configuration resembles a piecewise linearization. Note that the intervals have the special structure needed for Theorem 4.17, which is exploited by PATHCUT. To implement (4.3), the Multiple Choice Method is used. The objective function is constructed by drawing integer coefficients for the z-variables. This is done uniformly at random from the interval from 0 to twice the number of intervals per arc, with the restriction that there is an upper bound on the resulting 'slope' of the objective function. Section 4.3.5 uses different objective functions as will be explained there. The instances do not contain any additional constraints apart from those defining P.

As the generation of instances includes randomness, we always generated sets of five instances of the same type in terms of number of nodes and number of intervals per arc. The solution times given in the following are always (geometric) averages over five instances each. If only a subset of the five instances was solvable within the time and memory limitations, the average is taken over this subset only. In any case we also state the number of instances that could be solved.

The computational experiments have been performed on a queuing cluster of Intel Xeon E5-2690 3.00 GHz computers with 25 MB cache and 128 GB RAM, running Version 7 of Debian GNU/Linux. We have implemented the methods introduced above using the C++-API of Gurobi 6.00. We use Gurobi's standard parameter settings, except for turning on *PreCrush* for our cutting plane methods, which is mandatory if we want to add user cuts. Each job was run on 4 cores and with a time limit of 40 hours CPU-time.

4.3.3 Computational Results on Random Networks

The topology of the instances in this benchmark set is generated according to a preferential attachment model. It generates so-called scale-free graphs [AB02], which are known to represent the evolutionary behavior of complex real networks well. Starting with a small clique of initial nodes, the model iteratively adds new nodes. Each new node is connected to m of the already existing nodes. This parameter m, the so-called *neighborhood parameter*, influences the average node degree. We set $m = 2$ in order to generate sparse graphs that resemble infrastructure networks. Unfortunately, nodes of degree two are never adjacent in this setting, although longer paths of degree-two-nodes are very common in practice. Therefore, we modified the graph construction as follows: The second edge of each new node is present in the graph only with probability $\frac{2}{3}$. This remedies the mentioned shortcoming and guarantees connectedness.

First, we analyze the performance of the methods for instances of different size (50-150 nodes) with a fixed number of 10 intervals per arc. Subsequently, we will study the impact of the number of intervals per arc on a test set with fixed networks.

# network	MIP		PATHCUT		FORKCUT		CUT	
nodes	sol	CPU[s]	sol	CPU[s]	sol	CPU[s]	sol	CPU[s]
50	5	2.67	**5**	**2.34**	5	2.57	5	2.59
60	5	8.32	**5**	**7.02**	5	7.53	5	8.59
70	5	14.48	5	13.65	5	13.59	**5**	**12.43**
80	5	29.31	5	33.40	**5**	**25.77**	5	27.59
90	5	53.34	5	43.30	5	42.37	**5**	**36.74**
100	5	99.36	5	90.16	5	79.96	**5**	**77.27**
110	5	149.14	5	120.15	5	123.77	**5**	**109.50**
120	5	387.85	5	421.00	5	385.91	**5**	**305.87**
130	5	325.46	**5**	**186.55**	5	219.04	5	187.44
140	5	1 361.14	5	913.09	**5**	**813.26**	5	966.19
150	5	604.95	**5**	**378.96**	5	589.11	5	407.58

Table 4.1: Number of instances solved ('sol') and average solution times (CPU-time [s]) for instances on scale-free networks of varying size, 10 intervals per arc.

Table 4.1 reports the CPU-times required to solve the instance to optimality in seconds, each averaged over five instances of the same size. The first number in each column states the number of solved instances, whereas the second gives the average solution time. Note that we apply the geometric mean for the average values in order to account for outliers. The fastest method in each row is emphasized with bold letters. We rank the methods first by the number of solved instances and second by the average solution time. If a method did not solve any of the 5 instances of a given configuration, we denote this by an average solution time of '∞'.

We see that the solver benefits from using our cutting plane separators for most of the instance sets. For all instance sets, one of the new methods introduced here is fastest. Indeed, in most cases, both PATHCUT as well as FORKCUT lead to a considerable benefit. Therefore, it is not surprising that overall CUT performs best on the current test set. CUT achieves faster solution times than MIP for all instance sets except for the one with 60 nodes. Only for few instance sets one of the methods is less efficient than MIP. In those cases, the difference is not significant. The success of CUT correlates with a reduction of the number of branch-and-bound nodes required by the solver that is reduced by about the same factor as the runtime is. The gain in solution time is moderate for medium size instances, but grows with the size of the instances. For the set of largest instances, CUT needs about $\frac{2}{3}$ of the time required by MIP on average.

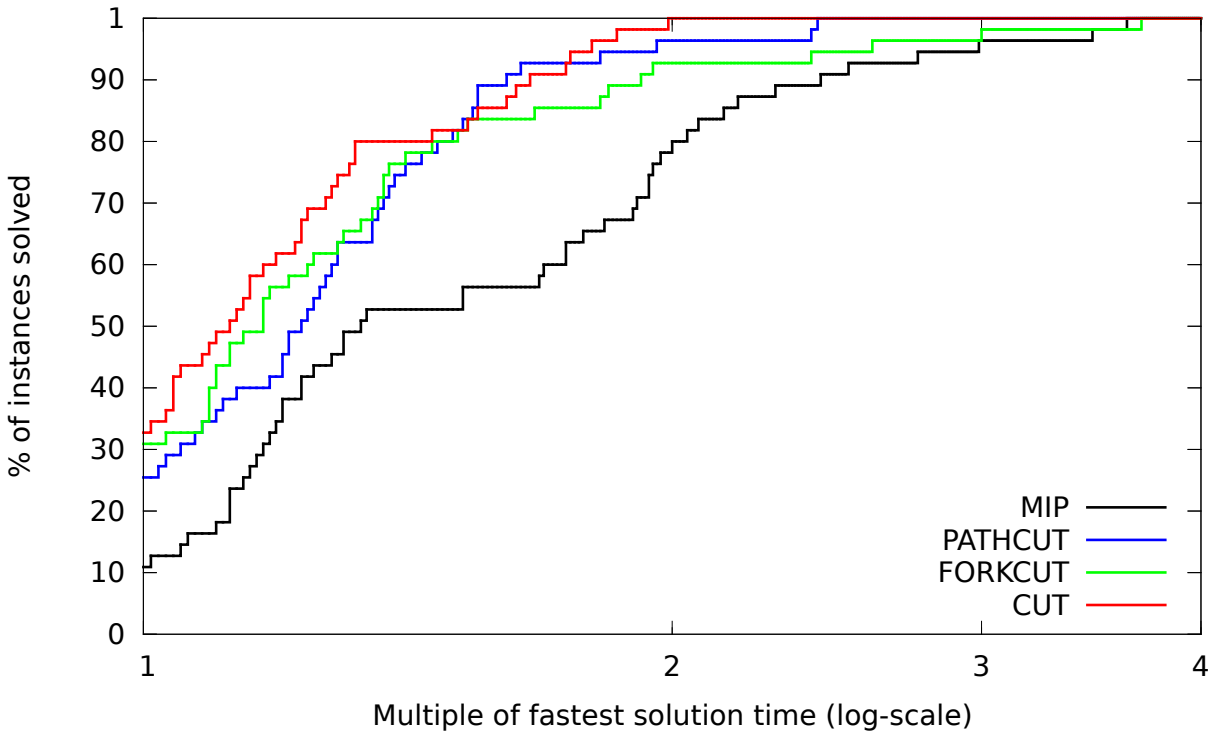


Figure 4.4: Performance profile for instances on scale-free networks of varying size (55 in total), 10 intervals per arc, compared by solution time (CPU-time [s]).

This is also emphasized in the performance profile in Figure 4.4. Though performance profiles have been used already in Chapter 3, we quickly recall their meaning for convenience: For each method, the percentage of all instances solved is shown as a function of the available time. This time is given in multiples of the solution time of the fastest method. In particular, the intercept of each curve with the vertical axis shows the percentage of instances for which the corresponding method achieves the shortest solution time. This kind of plot also provides additional information on how good a method is in catching up on instances for which it is not fastest. We see that

CUT is the fastest method for more instances than any other method, whereas for MIP this quantity is just about 10%, including mostly relatively easy instances. Also, given twice the time of the fastest method, MIP still solves only about $\frac{2}{3}$ of the instances. That means it is far behind for quite a number of instances. The performance of PATHCUT and FORKCUT also suggests that both types of cutting planes lead to improvement, independently from each other.

# intervals	MIP		PATHCUT		FORKCUT		CUT	
per arc	sol	CPU[s]	sol	CPU[s]	sol	CPU[s]	sol	CPU[s]
5	5	3.85	5	3.78	**5**	**2.86**	5	3.91
7	5	25.74	5	26.05	5	28.05	**5**	**22.85**
10	5	102.07	5	92.25	5	81.89	**5**	**78.78**
15	5	576.22	5	387.40	**5**	**269.23**	5	277.01
20	4	2 088.74	4	1 583.41	5	2 967.00	**5**	**2 047.47**
25	5	4 644.31	5	2 332.10	5	2 338.88	**5**	**1 695.06**
30	3	7 946.24	4	4 846.78	5	7 897.16	**5**	**3 406.70**
50	0	∞	1	79 424.01	1	19 897.23	**2**	**33 771.02**
70	0	∞	1	55 651.25	2	73 108.05	**2**	**33 779.41**
100	0	∞	0	∞	0	∞	0	∞

Table 4.2: Number of instances solved ('sol') and average solution times (CPU-time [s]) for instances on scale-free networks with 100 nodes, varying number of intervals per arc.

We expect instances to become more difficult when the number of intervals per arc increases. However, there is the natural question whether this has any impact on the relative performance of the methods. Therefore, we now consider instances of fixed networks of 100 nodes, varying the number of flow intervals for each network arc. The results are shown in Table 4.2. CUT outperforms all other implementations with the exception of two instance sets. And it is clearly more efficient than MIP except for the set of very easy instances that could be solved in less than 10 seconds of CPU-time. Here, also programming overhead in our separating routines might be an issue. We observe similar results when considering the number of branch-and-bound nodes instead of the solution time. As before, the benefit from using CUT increases with problem size. However, this effect is much more evident than in Table 4.1. Using CUT, the solver is able to optimize significantly more instances. This is most notable for the set of instances with 30 intervals per node. Here, the average solution time is much smaller for CUT, although it includes the two instances MIP was not able to solve. This is a drastic improvement. Our conclusion is that our methods are better at dealing with the difficulties posed by lager number of intervals per arc. We might observe this from a purely empirical point of view, but a possible explanation might be the following. For the clique problem with classical edge-formulation, it is known

that the *Gomory-rank* of stable set inequalities increases with the size of the stable set [Chv73]. This may lead to strong cutting planes that the solver is less likely to find by itself. We experimented with the Gurobi- parameter *MIPfocus*, which allows to increase the solver's aggressiveness in generating cuts, but we did not observe significantly different results.

4.3.4 Performance on a Real-World Network Topology

This test set consists of instances based on a realistic topology of a gas network by the German gas network operator *Open Grid Europe* (OGE). It consists of 592 nodes and 623 arcs. 224 nodes have degree two and 128 paths of degree-two-nodes were detected, which amounts to an average length of 2.75. The longest of those paths has length 8.

For large numbers of intervals per arc we encountered numerical difficulties on this test set. These numerical issues are already observed in the standard solution method MIP. To overcome this we used Gurobi's parameter *NumericFocus* to tell the solver to be more careful regarding numerical issues. To be safe, we set it to the maximum value of 3 for all tested solvers for the following computations. As a result, we did not observe numerical difficulties for any of the instances any more. However, this choice results in longer running times for all solvers. Due to this and the large number of nodes, instances could only be solved up to 10 intervals per arc.

# intervals	MIP		PATHCUT		FORKCUT		CUT	
per arc	sol	CPU[s]	sol	CPU[s]	sol	CPU[s]	sol	CPU[s]
3	5	66.63	5	73.13	5	59.77	**5**	**47.96**
4	5	4 943.87	5	468.66	5	546.96	**5**	**265.98**
5	5	9 001.10	5	1 627.76	5	1 449.77	**5**	**723.65**
6	2	31 384.31	3	10 089.66	**5**	**7 924.77**	5	8 220.91
7	1	103 191.92	2	122 299.54	5	21902.84	**5**	**16 490.54**
8	0	∞	0	∞	3	16 541.76	**4**	**27 133.11**
9	0	∞	0	∞	1	13 253.19	**3**	**40 070.03**
10	0	∞	0	∞	0	∞	**2**	**133 889.72**

Table 4.3: Number of instances solved ('sol') and average solution times (CPU-time [s]) for instances on a gas network topology with 592 nodes, varying number of intervals per arc.

Table 4.3 shows the results on this test set. We see similar behavior as in Table 4.2 for the scale-free networks: MIP is competitive only for easy instances and our algorithms are clearly ahead from a certain level of difficulty/number of intervals onwards. CUT and FORKCUT both solve all instances of the test sets up to 7 intervals per arc, which MIP is not able to do. CUT still manages to solve more than half of the 15 most difficult

instances, whereas MIP doesn't solve any of them, which is mainly due to memory limitations. From the log files it seems unlikely that MIP would have succeeded within the time limit given enough memory. Nevertheless, reduced memory consumption is an important advantage of our cutting plane methods. The benefit of using our cutting planes is drastic from this test set with realistic topology. Apparently, both the cuts of PATHCUT and FORKCUT contribute to the success of CUT. This confirms the results obtained on the topology of thinned scale-free graphs from Subsection 4.3.3.

4.3.5 Continuous Piecewise Linear Objectives and the Incremental Formulation

Until now, the results for the instances reported here had an objective function that only considered the binary z-variables and hence corresponds to a piecewise constant approximation. In many applications, a continuous piecewise linear function is present, which means that the objective function is also bound to the q-variables. So we may ask whether instances with such an objective function behave differently than what has been reported so far. This is investigated in the following.

In order to obtain instances featuring a continuous piecewise linear objective function according to (4.4), we take the instances from Table 4.1, but with a different objective that is created as follows: the drawn values for the z-variables are instead interpreted as function values at the breakpoints (and one more value has to be drawn of course). Objective coefficients for the z- and q-variables then follow from (4.4). Note that all instances in the computational section feature the interval overlap structure that is typical for piecewise linearization (i.e. intervals result from subdivision of a larger interval). Both variants for the objective function are illustrated in Figure 4.5.

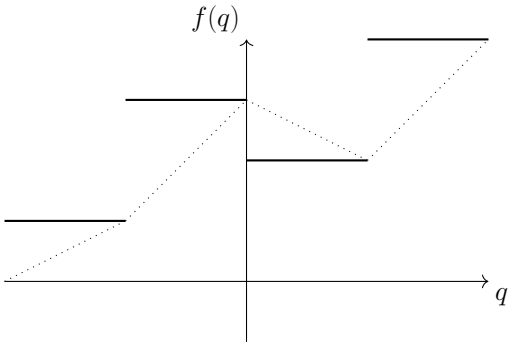


Figure 4.5: Illustration of the objective as a function of the flow q - reduced to a particular arc. Drawing objective coefficients for z-variables corresponds to an objective f that is piecewise constant in q. It is depicted by thick lines. Dotted lines show the corresponding continuous piecewise linear interpolation as in (4.4).

In particular, the continuous piecewise linear objective is constructed in such a way that it interpolates the coefficients for the piecewise constant objective. Also, corre-

sponding instances use the same random graph, so the instances are the same as those from Table 4.1 up to the objective function. The instances with continuous piecewise linear objective function turned out to be much more difficult than their piecewise constant counterparts. Also for this test set, numerical difficulties for MIP were encountered. As before, we set Gurobi's parameter *NumericFocus* to the value of 3 which remedied any inconsistencies.

# network	MIP		PATHCUT		FORKCUT		CUT	
nodes	sol	CPU[s]	sol	CPU[s]	sol	CPU[s]	sol	CPU[s]
20	5	10.53	**5**	**5.03**	5	10.30	5	6.25
30	5	63.34	5	55.12	5	66.15	**5**	**53.32**
40	5	489.93	5	639.15	**5**	**473.83**	5	537.92
50	5	9361.08	5	7689.44	**5**	**6598.01**	5	7887.72
60	3	26299.43	4	29773.15	5	40159.14	**5**	**38246.99**
70	0	∞	0	∞	0	∞	0	∞
80	0	∞	0	∞	0	∞	0	∞

Table 4.4: Number of instances solved ('sol') and average solution times (CPU-time [s]) for instances with continuous piecewise linear objective on scale-free networks of varying size, 10 intervals per arc.

The results are shown in Table 4.4. The increased computational difficulty of this instance set is obvious. From 70 nodes on, no instance was solved within the time limit by any method. Therefore, we cut the table beyond instances with 80 nodes and added sets with $20, 30$ and 40 nodes, respectively. Despite the changes in difficulty, no qualitatively different behavior of the methods can be observed. CUT is more efficient than MIP on 4 out of 5 meaningful instance sets - and reduces the runtime by about the same factor as measured from Table 4.1.

Finally, we consider a formulation according to the Incremental Method as described in Subsection 4.2.3, as it is very popular and often the method of choice for piecewise linearization in gas network optimization. Given that the presented cutting planes simplify significantly for the δ-Method, one might expect an even larger benefit from using them. On the other hand, this means that they are also more likely to be found by the solver's generic cut generation methods anyway. For the following experiment we use the same instances and setting as for Table 4.4, apart from using the incremental formulation.

As can be readily seen from Table 4.5, using the Incremental Method reduces the runtime of all solution methods significantly. For example, instances from the set with 50 nodes are solved ~ 25 times faster for MIP and ~ 45 times faster for CUT on average. Also, much more instances are now solvable within the time limit, allowing for test sets with more network nodes. This is not surprising; in [CPSM14], a recent in-depth computational study for piecewise linear functions in the context of gas network opti-

# network nodes	MIP sol	MIP CPU[s]	PATHCUT sol	PATHCUT CPU[s]	FORKCUT sol	FORKCUT CPU[s]	CUT sol	CUT CPU[s]
20	5	1.21	**5**	**1.13**	5	1.22	5	1.15
30	5	3.83	**5**	**3.16**	5	3.91	5	3.29
40	5	29.92	**5**	**19.00**	5	27.51	5	19.08
50	5	352.05	**5**	**168.48**	5	320.29	5	171.03
60	5	1838.76	**5**	**707.42**	5	1081.13	5	759.05
70	5	27767.41	**5**	**8216.85**	5	14981.19	5	12331.28
80	3	71907.28	**5**	**22771.34**	4	31749.03	4	17519.88
90	0	∞	**3**	**72546.29**	1	141738.59	2	52504.78
100	0	∞	**2**	**47902.72**	1	89969.98	2	87192.53
110	0	∞	**1**	**76720.97**	0	∞	1	95796.02
120	0	∞	0	∞	0	∞	0	∞

Table 4.5: Number of instances solved ('sol') and average solution times (CPU-time [s]) for instances with continuous piecewise linear objective on scale-free networks of varying size, 10 intervals per arc, using the Incremental Method.

mization sees the Incremental Method coming out on top, outperforming the Multiple Choice Method even by several orders of magnitude for some test sets. A clear benefit of using our cutting planes persists also for this formulation, where the relative reduction in runtime overall has moderately increased compared to Table 4.4. Most notably, our cutting planes allow solving a number of instances that would have hit the time limit otherwise. In contrast to Table 4.1 and Table 4.2, now PATHCUT is the most successful method for all instance sets, though also FORKCUT and CUT clearly outperform MIP. A possible explanation would be that the cutting planes at paths of length 2 are quite valuable here, as this is the common feature of PATHCUT and FORKCUT, where the former is suited better to this task due to its separation being exact.

4.4 Further Remarks on Extending Applicability

For the computations in the previous section, the results from Subsection 4.2.2 and Subsection 4.2.4 were applied to induced paths in the network and single network nodes, respectively. Though it turned out that this resulted in significant strengthening of the model, I would like to remark that there are more situations in which the cutting planes are applicable.

First of all, the results on paths of arbitrary length from Subsection 4.2.2 may also be applied to substructures that form induced paths of degree-2-nodes if we only consider edges for which the flow is not fixed. This includes structures in which middle nodes might have junctions but the flow contribution from outside the substructure is

fixed (or can be fixed during preprocessing). The reason is that for forming P, we can effectively convert those fixed arc flows into demands for middle nodes—exactly as we did in Chapter 3 for the construction of the maximum flow subproblems. Though we might have to keep the arc in the network model due to other problem constraints like pressure compatibility in gas networks. Note that we assumed nonzero demand at the middle nodes of the path only for the sake of simplicity, and that dealing with the more general case is described in Section 4.2. We can apply the results e.g. to isolated circles in the network, i.e. circles that do not intersect with another circle. This opens up more situations for application, especially if the network is relatively sparse and does not contain a lot of intersecting cycles as it is not uncommon for gas networks.

Moreover, note that the results for stars in Subsection 4.2.4 apply to any cut in the network, not only to cuts around a single node. This implies that we might use our separation routine for inequalities from Subsection 4.2.4 at aggregate nodes. It is not clear beforehand which aggregate nodes are suitable for generating strong cutting planes, and it might not be practical to apply separation at every subset of the nodes. However, this observation means that the techniques from this and the previous chapter can in principle be combined: Cuts can be obtained from applying FORKCUT at nodes of an aggregated master problem and stay valid after disaggregation. Moreover, we can hope that aggregation techniques can help to identify interesting aggregate nodes for applying FORKCUT even if the network eventually is completely disaggregated during the algorithm.

Of course it would be useful to be able to also transfer our results to a logarithmic formulation (see Subsection 2.2.4) like we did with the Incremental Formulation in Subsection 4.2.3. Unfortunately, there does not seem to be a direct way do this. At this point I want to highlight that a key concept of our reasoning is the compatibility graph. In order to define it, we need the fact that it is sufficient to consider pairwise conflicts for the binary variables. Also, the compatible choices for a given binary variable (interval) on one arc have to be 'connected' in some sense, to allow us to make the connection to chordality in graphs. Both properties are not present for a logarithmic formulation. As mentioned in Chapter 2, the logarithmic model is often outperformed by other formulations despite having fewer variables. So the reduction in variables apparently comes at the cost of a somehow less 'solver-friendly' structure. Our investigations might give some additional hint at the reason for the strength of incremental modeling for nonlinear network flow problems. Namely, the complete description of P can be expressed by very sparse inequalities.

In the following chapter, we will create an abstract generalization based on the two key ingredients mentioned above.

Chapter 5

Staircase Compatibility

Compatibility structures are prevalent in many combinatorial optimization problems. In fact, they arise whenever the choice of one solution element immediately narrows down the choice of other elements. Such compatibilities are the core of many combinatorial optimization problems on graphs. Typical examples include the clique problem, which consists of finding a maximum clique in a given undirected graph, or the k-colorability problem, which asks for finding a k-partition of a given graph such that vertices connected by an edge have to be in different partitions.

In this chapter, we consider a special type of compatibility problem, inspired by problems from the previous chapter as well as an application from energy efficient timetabling that is closely related to the classical project scheduling problem. Again, our aim will be to derive strong integer programming formulations for the structures considered. Major parts of this chapter are based on joint work with Andreas Bärmann, Thorsten Gellermann and Oskar Schneider, published in [BGMS16].

5.1 The Clique Problem with Multiple-Choice Constraints

We consider the combination of compatibility constraints together with another frequently-occurring structure, namely so-called multiple-choice constraints

$$\sum_{i \in \mathcal{I}} x_i = 1,$$

where x_i are binary variables for i in some finite index set $\mathcal{I}$. These constraints are present whenever there is a partition of the set of eligible elements into subsets such that it is required to choose exactly one element from each subset. In the literature—especially in the context of set packing and set covering—they are also known as *partitioning constraints*, e.g. in [CCZ14]. Altogether, this leads to a problem that can be classified as a clique problem with multiple-choice constraints (CPMC). It is formally defined as follows:

Definition 5.1 (The Clique Problem under Multiple-Choice Constraints)**.** Let S be a finite basic set together with a partitioning of S into m disjoint subsets $\mathcal{S} = \{S_1, \dots, S_m\}$, i.e. $S = \bigcup_{i=1}^{m} S_i$ and $S_i \cap S_j = \emptyset$ for $i \neq j$. Consider a symmetric relation

$$R \subseteq (S \times S) \setminus \bigcup_{i=1}^{m} (S_i \times S_i).$$

Two elements $s \in S_i$, $t \in S_j$ are said to be *compatible* if and only if $(s,t) \in R$ holds. The *clique problem under multiple-choice constraints* (CPMC) is then given by the task to

(SC1) choose exactly one element from each subset S_i

such that the selected elements are pairwise compatible.

Hence, the clique problem under multiple-choice constraints amounts to finding a clique in the undirected graph $G = (S, R)$ whose nodes are the elements of S and whose arcs connect exactly the pairwise compatible elements in S such that exactly one element from each subset in the partition of S is chosen. We call G the *compatibility graph* associated with relation R. Note that this definition of the compatibility graph is consistent with Definition 4.1 in Chapter 4.

Remark 5.2. In this generality, (CPMC) is NP-hard as it e.g. covers the problem of graph colorability for a given number of colors. This problem can be modeled as (CPMC) in the following way: Let H be the graph to be colored and let k be the number of available colors. Create a partition S_v for each vertex $v \in V(H)$ with choices $S_v = \{s_{c,v},\ c = 1, \dots, k\}$, meaning that v is colored with color c. The compatibility relation R is then defined by

$$(s_{c,v}, s_{c',u}) \in R \quad \Leftrightarrow \quad c \neq c' \text{ or } (u,v) \text{ is not an edge in } H.$$

This fully describes the colorability problem in terms of (CPMC). Furthermore, it is NP-complete to decide if a given graph admits a k-coloring for a given $k \geq 3$ [GJS74].

While (CPMC) is NP-hard in general we want to investigate a special case where it is solvable efficiently. This is possible for a restriction of the compatibility graphs to graphs with a special compatibility structure. We have seen in Subsections 4.2.1 and 4.2.2 of the previous chapter that the combinatorial optimization problem of choosing compatible flow intervals along a path of degree-2-nodes can be reformulated as such a clique problem. We know already from Chapter 4 that (CPMC) is solvable in polynomial time if the compatibility graph is partition-chordal (see Definition 4.10).

In this chapter, we go a different route and integrate this problem into another general class of clique problems that admit a perfect description of the convex hull of feasible points that even is of linear size, essentially the one from in Theorem 4.22. This of course means that we have to restrict ourselves to a special interval structure that is required for this theorem. However, this is the case for the interval structure originating from piecewise linearization, hence we will assume this case for the examination

of interval compatibility problems within this chapter. We will see that this problem has significant similarities to the classical project scheduling problem, for which such model reformulations are already known (see [MSSU01]). Our definition of *staircase compatibility* will generalize the key properties of both special cases that allow us to state efficient integer programming formulations for which we can show that the corresponding constraint matrix is totally unimodular. However, the notion of staircase compatibility provides a common, more general framework to study the underlying clique problem with multiple-choice constraints. In particular, it will be shown that the derived integrality results hold for a wider class of compatibility graphs.

Note that Definition 5.1 requires that the whole problem can be defined in terms of pairwise compatibilities. This is a severe restriction. Though compatibility structures are extremely common in all kinds of optimization problems, often times a combination of choices might rule out an option for the solution, while every single choice does not. This was e.g. the case for interval compatibilities in star graphs in Subsection 4.2.4, where we could not take advantage of the compatibility graph. However, as remarked for our setting in Chapter 4, the structure of (CPMC) investigated here might well be present as a substructure and strengthening the formulation locally can have a huge impact on solving times.

In order to demonstrate that there is great benefit from studying this structure, we present computations on real-world applications which are special cases of (CPMC) under staircase compatibility in Subsection 5.4. First, we consider a problem in railway timetabling which is a special case of the project scheduling problem. After that we revisit piecewise linearized network flow problems in the light of this chapter.

5.2 Staircase Compatibility

Let us now focus on a special case of (CPMC) with a certain 'connectedness' structure in the underlying compatibility relation:

Definition 5.3 (Staircase Relations)**.** Let each subset S_i in the partition of S be an ordered set according to a total order $<_i$, which allows us to denote the elements of S_i by $s_{i,1}, \dots, s_{i,n_i}$ with $n_i = |S_i|$. In the following, we omit the index i and simply write $<$ whenever no confusion is possible. We then call a symmetric relation R on S a *staircase relation* if two conditions hold. The first condition states the connectedness of the compatible choices for a given element:

(SC2a) $$(a, b_{k_1}) \in R \wedge (a, b_{k_3}) \in R \;\Rightarrow\; (a, b_{k_2}) \in R,$$

whenever $a \in S_i, b_{k_1}, b_{k_2}, b_{k_3} \in S_j, b_{k_1} < b_{k_2} < b_{k_3}$. The second condition forces some kind of monotonic behavior of R:

(SC2b) $$(a_{l_1}, b_{k_2}) \in R \wedge (a_{l_2}, b_{k_1}) \in R \;\Rightarrow\; (a_{l_1}, b_{k_1}) \in R \wedge (a_{l_2}, b_{k_2}) \in R$$

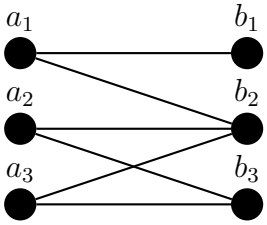
for $a_{l_1}, a_{l_2} \in S_i,\ b_{k_1}, b_{k_2} \in S_j,\ a_{l_1} < a_{l_2},\ b_{k_1} < b_{k_2}$.

If the relation R in the setting of Definition 5.1 is a staircase relation, we call the arising special case of (CPMC) the *clique problem with multiple-choice constraints under staircase compatibility* (CPMCS).

The choice of the term 'staircase relation' becomes clear when considering the adjacency matrix of the compatibility graph corresponding to such a relation: each submatrix that describes the compatibility between the elements of two subsets of the partition is a staircase matrix if its rows and columns are ordered according to the $<_i$ (see [Fou84] for an extensive compilation of the properties of staircase matrices).

Before moving on to a further discussion of this problem, we consider an example for illustration.

Example 5.4. Consider the following example:



$$\begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \end{array} \begin{array}{c} \begin{array}{ccc} b_1 & b_2 & b_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \end{array}$$

It shows the compatibility graph for (CPMCS) as well as the corresponding adjacency matrix for a certain staircase relation R on the set $S = S_1 \cup S_2$ with $S_1 = \{a_1, a_2, a_3\}$ and $S_2 = \{b_1, b_2, b_3\}$. We see that removing edge $\{a_2, b_2\}$ would violate (SC2a) (and also (SC2b), see Lemma 5.5 above), while removing $\{a_3, b_3\}$ would violate (SC2b). Any selection $\{a, b\}$ with $(a, b) \in R$ would be feasible for (CPMCS). The axiom (SC2b)—which might seem complicated at first—has the following illustrative meaning: whenever there is a crossing of edges (a_{l_1}, b_{k_2}) and (a_{l_2}, b_{k_1}) in the compatibility graph (if it is drawn as in this example) then also the 'uncrossed' edges (a_{l_1}, b_{k_1}) and (a_{l_2}, b_{k_2}) must belong to the compatibility graph.

Lemma 5.5. *Under the assumption that each element s of a subset S_i in the partition of S has at least one element in each of the remaining subsets with which it is compatible, i.e.*

$$\{s' \in S_j | (s, s') \in R\} \neq \emptyset \; \forall s \in S_i, \; \forall j = 1, \ldots, m, \; j \neq i, \tag{5.1}$$

(SC2a) *is implied by* (SC2b).

Proof: Let $s \in S$ be an element of subset S_i for some $i \in \{1, \ldots, m\}$ and let $S_j, j \neq i$ be another partition. Now let $a \in S_i, b_{k_1}, b_{k_2}, b_{k_3} \in S_j, b_{k_1} < b_{k_2} < b_{k_3}$ and $(a, b_{k_1}) \in R$ as well as $(a, b_{k_3}) \in R$ (see Figure 5.1 for an illustration). In order to prove the claim, we have to show that $(a, b_{k_2}) \in R$, assuming (5.1) and (SC2b). Applying (5.1) to b_{k_2}, there is an element $s' \in S_i$ such that $(s, b_{k_2}) \in R$. If $s' = s$, the claim immediately follows; otherwise, without loss of generality let $s' > s$.

Due to $(s', b_{k_2}) \in R$ and $(s, b_{k_3}) \in R$, we can apply (SC2b): it follows $(s, b_{k_2}) \in R$ (and $(s', b_{k_3}) \in R$) which completes the proof. □

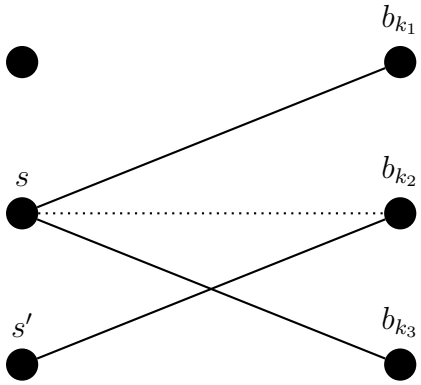


Figure 5.1: Illustration of the proof of Lemma 5.5.

The assumption in Lemma 5.5 is fairly mild since an element that does not have at least one compatible choice in every component cannot belong to any feasible selection and may be eliminated in a preprocessing step. Therefore, we will assume this case for the remainder of the chapter. This will come in especially handy in Section 5.5, where we will discuss issues regarding recognizability of staircase graphs, and only have to check (SC2b).

5.2.1 Two Applications of (CPMCS)

In the following, we give two example applications (CPMCS) may arise from. In both examples, the project scheduling problem and interval compatibilities in path flows, it is a possible way to characterize the set of feasible solutions. The latter application is already familiar from Chapter 4.

Project Scheduling Let m tasks $j = j_1, \dots, j_m$ be given. Each task has to be carried out at exactly one time slot, where we assume a discrete set $T_j = \{t_{j,1}, \dots, t_{j,n_j}\}$ of possible execution times to be given that may differ for different jobs. Additionally, pairs of tasks may have precedence restrictions requiring one of them to start in a predefined time window relative to the other (if no relation is given, they may be done in any order, or possibly in parallel). This problem is called the *project scheduling problem with precedence constraints*. For further information and examples, see [SZ15] and the references therein.

The following is a possible formulation for the above scheduling problem:

(5.2)
$$\begin{aligned} \text{find} \quad & x \\ \text{s.t.} \quad & x_k - x_l \leq \overline{d}_{k,l} && (\ 1 \leq k < l \leq m\ ) \\ & x_k - x_l \geq \underline{d}_{k,l} && (\ 1 \leq k < l \leq m\ ) \\ & x_j \in T_j && (\ j = 1, \dots, m\ ) \end{aligned}$$

for some $\overline{d}_{k,l}, \underline{d}_{k,l}$ with $k = 1, \dots, m$ and $l = k+1, \dots, m$.

We can model this problem as (CPMCS) as follows: each subset S_i represents a job j_i, where the elements in each subset are identified with the possible execution times $\{t_{j,1}, \ldots, t_{j,n_j}\}$. Consequently, the subsets come with an obvious chronological ordering. For different jobs j_k, j_l with $k \neq l$, we have

$$(t_{k,i_k}, t_{l,i_l}) \in R \Leftrightarrow \underline{d}_{k,l} \leq t_{k,i_k} - t_{l,i_l} \leq \overline{d}_{k,l}$$

It is easily seen that (SC2a) is satisfied due to the convexity of the relative time window defined by $\overline{d}_{k,l}$ and $\underline{d}_{k,l}$. Furthermore, violating (SC2b) would contradict the temporal ordering. Therefore, R as defined here is a staircase relation.

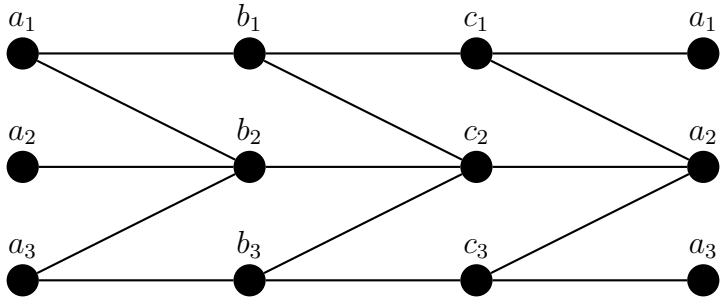
Interval Compatibilities in Path Flows Let a path network consisting of m edges $e_1, \ldots, e_m$ be given. Each edge has an interval for the feasible flow on the edge which furthermore is subdivided into n_i subintervals, $i = 1, \ldots, m$. This scenario appears as a substructure in network flow problems where the flow has been piecewise linearized as explained in detail in Chapter 4. Though note that in the current chapter we only consider the special case in which intervals on an arc arise from subdivision of a larger interval; in particular, the assumption of Theorem 4.17 holds and the case of Example 4.6 is excluded. We will see in Section 5.5 that the relation represented in this example does not have the staircase property.

The task is to describe the set of feasible combinations of flow intervals. It represents a special case of (CPMCS) as can be seen as follows: define S as the set of all intervals, where subset S_i includes all intervals belonging to edge i of the path. As those intervals are obtained from subdividing a larger interval, a canonical ordering is available. Intervals belonging to different not necessarily adjacent edges are *compatible* if and only if it is possible for the path flow to satisfy the bounds of both intervals. If the demand of all intermediate nodes of the path is zero, this is true if and only if they have nonempty intersection. Nonzero demands on path nodes can be reduced to this case by simple interval arithmetic which amounts to shifting intervals appropriately. An important observation is that the resulting relation R completely describes the problem, as a set of intervals is guaranteed to be compatible altogether if each pair of intervals is compatible. Finally, R is a staircase relation, where (SC2a) follows from the fact that intervals are convex, and (SC2b) can be seen to hold from the way intervals can be sorted for each network edge. In particular, the precondition of (SC2b), namely $(a_{l_1}, b_{k_2}) \in R \wedge (a_{l_2}, b_{k_1}) \in R$ for some $a_{l_1}, a_{l_2} \in S_i,\ b_{k_1}, b_{k_2} \in S_j$ with $a_{l_1} < a_{l_2}$ and $b_{k_1} < b_{k_2}$, is only fulfilled if the subdivisions on the network arcs corresponding to S_i and S_j use a common breakpoint. Consequently, this breakpoint is contained in all intervals corresponding to $a_{l_1}, a_{l_2}, b_{l_1}, b_{l_2}$, and it follows $(a_{l_1}, b_{k_1}) \in R$ and $(a_{l_2}, b_{k_2}) \in R$.

Relation to General (CPMCS) The set of staircase relations that may originate from one of the two special cases of (CPMCS) forms a strict subclass of general staircase relations as defined in Definition 5.3. Intuitively, this is explained by the fact that most applications—including the two above—allow for some 'transitivity reasoning',

i.e. the compatibilities between subsets S_1 and S_2 together with those between S_2 and S_3 restrict the possible compatibilities between S_1 and S_3. However, according to the definition, both (SC2a) and (SC2b) only consider two subsets at a time. The following gives an example for a compatibility graph that does not originate from either of the two special cases mentioned above.

Example 5.6. Consider the following compatibility graph G belonging to an instance of Problem (CPMCS) with three subsets $A = \{a_1, a_2, a_3\}$, $B = \{b_1, b_2, b_3\}$ and $C = \{c_1, c_2, c_3\}$, each of which has three elements. Note that there is another copy of partition A in the figure below to represent the compatibilities with partition C in order to highlight the symmetric structure of the example.



Suppose G was obtained from an instance of Model (5.2). Then we could identify each partition with a job and each element of the partition with a possible execution time. We denote by $d_{A,B} \coloneqq \overline{d}_{A,B} - \underline{d}_{A,B}$ the length of the time window between jobs A and B and similarly for the other relations. As $(a_2, b_2) \in R$, but $(a_2, b_1) \notin R$, $(a_2, b_3) \notin R$, we can conclude that $d_{A,B}$ is less than the time difference between b_1 and b_3, by slight abuse of notation denoted by $b_3 - b_1 > d_{A,B}$. Due to c_2 being connected to all nodes in B, the time window of length $d_{B,C}$ has to include b_1 as well as b_3 and hence $d_{B,C} \geq b_3 - b_1$, implying $d_{B,C} > d_{A,B}$. As the instance is symmetric, we can repeat this argument to obtain $d_{A,C} > d_{B,C}$ and $d_{A,B} > d_{A,C}$, which leads to the contradiction $d_{A,B} < d_{B,C} < d_{A,C} < d_{A,B}$.

Similar reasoning shows that G also cannot be obtained from an instance of interval compatibilities on a path flow network (as it is described above): the argument is completely analogous, but uses the diameter of the intervals belonging to a_2, b_2, c_2 instead of $d_{A,B}, d_{B,C}, d_{A,C}$.

Moreover, the situation is no different if we do not assume the ordering of the elements within each partition to be given. This is because there is no other ordering that makes R a staircase relation apart from reversing all partition orderings.

Remark 5.7. The k-coloring problem has been utilized in Remark 5.2 for showing that the general version of (CPMC) is NP-hard. Modeling the k-coloring problem as (CPMC) as described there in general does not lead to a staircase relation. The reason is that the k colors do not offer a natural ordering that ensures (SC2b) or even (SC2a). In fact, for $k \geq 3$ it can be checked quickly that the k-coloring problem cannot lead to staircase relations even for a graph consisting of a single edge, and irrespective of the

ordering on both partitions. This is consistent with what we will show in the following section, namely that (CPMCS) represents a special case of (CPMC) that is solvable in polynomial time.

In this context I would like to emphasize that staircase compatibility is not suited to model 'all-different-constraints'. For modeling the project scheduling problem (5.2) as (CPMC) it is therefore key that jobs can be run in parallel if no relative time window constraint is given.

5.3 Efficient MIP-Formulations for (CPMCS)

The problem (CPMCS) introduced in the previous section can be modeled as a mixed-integer program (MIP) in a straightforward fashion: for each element $s \in S$ we introduce a variable $x_s \in \{0,1\}$ that takes a value of 1 if this element is chosen and 0 otherwise. A vector x is then a feasible selection if and only if it is a solution to the following feasibility problem:

$$
\begin{aligned}
& \text{find} && x && && \\
\text{(5.3a)} \quad & \text{s.t.} && \sum_{s \in S_i} x_s && = 1 \qquad && (\ \forall S_i \in \mathcal{S}\ ) \\
\text{(5.3b)} \quad & && x_s && \leq \sum_{\substack{t \in S_j: \\ (s,t) \in R}} x_t \qquad && (\ \forall S_i \in \mathcal{S},\ \forall s \in S_i,\ \forall S_j \in \mathcal{S}, j > i\ ) \\
\text{(5.3c)} \quad & && x && \in \{0,1\}^{|S|}, &&
\end{aligned}
$$

where $\mathcal{S}$ denotes the given partition consisting of subsets $S_1, \ldots, S_m$, $m \in \mathbb{N}$. The multiple-choice constraints (5.3a) ensure that exactly one element of each subset in $\mathcal{S}$ is chosen, while compatibility constraints (5.3b) enforce the pairwise compatibility of the chosen elements according to the relation R: choosing an element s from one subset S_i implies that we have to choose one of the elements compatible to s in each of the remaining subsets S_j. Integrality constraints (5.3c) finally restrict variables x to take binary values. Note that constraints (5.3b) for two subsets S_i, S_j are redundant if $(s,t) \in R$ for all $s \in S_i$ and $t \in S_j$.

Remark 5.8. It is easy to find examples where Constraints (5.3c) are actually needed as Constraints (5.3a) and (5.3b) are not sufficient to ensure integrality of the solution. For the instance presented in Example 5.4, Model (5.3) reads:

$$
\begin{aligned}
& \text{find} && x \\
& \text{s.t.} && x_1 + x_2 + x_3 && = 1 \\
& && x_4 + x_5 + x_6 && = 1 \\
& && x_1 && \leq x_4 + x_5 \\
& && x_2 && \leq x_5 + x_6 \\
& && x_3 && \leq x_5 + x_6
\end{aligned}
$$

$$x \quad \in \quad \{0,1\}^6.$$

It allows for the fractional solution $(0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2})$ if $x \in \{0,1\}^6$ is relaxed to $x \geq 0$ (observe that $x \leq 1$ is redundant). This solution is easily checked to be an extreme point of the corresponding polytope.

As we have seen, the polytope underlying Model (5.3) is not integral in general. However, we will see now that a small adaption leads to a totally unimodular description of the feasible set. Consider the following formulation:

$$\begin{aligned}
&\text{find} \quad x \\
\text{(5.4a)} \quad &\text{s.t.} \quad \sum_{s \in S_i} x_s \quad = \quad 1 && (\ \forall S_i \in \mathcal{S}\ ) \\
\text{(5.4b)} \quad &\sum_{\substack{s' \in S_i: \\ s' \geq s}} x_{s'} \quad \leq \quad \sum_{\substack{t \in S_j: \\ \exists t' \leq t: (s,t') \in R}} x_t && (\ \forall S_i \in \mathcal{S},\ \forall s \in S_i,\ \forall S_j \in \mathcal{S}, j \neq i\ ) \\
\text{(5.4c)} \quad &x \quad \in \quad \{0,1\}^{|S|}.
\end{aligned}$$

It uses the same set of variables as Model (5.3) as well as the same multiple-choice constraints to enforce (SC1). However, it features new compatibility constraints whose left-hand side arises by summing all $x_{s'}$ for $s' \in S_i$ with $s' > s$ onto the left-hand side of the old compatibility constraint (5.3b) corresponding to element $s \in S_i$ and some S_j with $j \neq i$. Its right-hand side arises by taking the old right-hand side and adding all variables x_t for $t \in S_j$ that are greater than some $t' \in S_j$ that is compatible to s. Readers of Chapter 4 may have noticed that those constraints match Constraints (4.9) for upwards connected subsets of S_i (see Definition 4.21). Furthermore, note that this new model also incorporates compatibility constraints for subsets S_i and S_j with $i < j$. In the following, we show that the two models are in fact equivalent.

Proposition 5.9. *The respective feasible sets of Models* (5.3) *and* (5.4) *coincide.*

Proof: We begin by showing that each feasible solution to Model (5.3) is also feasible for Model (5.4). To see this, consider an element s of a subset S_i and its corresponding compatibility constraint (5.3b) with the elements of another subset S_j, which reads

$$x_s \leq \sum_{\substack{t \in S_j: \\ (s,t) \in R}} x_t.$$

By summing up these constraints for all elements $s' \geq s$, we obtain

$$\sum_{\substack{s' \in S_i: \\ s' \geq s}} x_{s'} \overset{\text{(5.3b)}}{\leq} \sum_{\substack{s' \in S_i: \\ s' \geq s}} \sum_{\substack{t \in S_j: \\ (s',t) \in R}} x_t = \sum_{t \in S_j} |\{s' \in S_i \mid s' \geq s, (s',t) \in R\}| \cdot x_t.$$

Due to (SC2a), all coefficients for variable x_t with $t > t'$ for some t' with $(t', s) \in R$ on the right-hand side of this inequality are exactly those which are non-zero, i.e. 1

or greater. As its left-hand side can at most take a value of 1 due to the multiple-choice constraint for subset S_i, all coefficients on the right-hand side greater than 1 can be reduced to 1 without changing the set of integer solutions fulfilling the inequality. Therefore,

$$\sum_{t \in S_j} |\{s' \in S_i \mid s' \geq s, (s', t) \in R\}| \cdot x_t = \sum_{\substack{t \in S_j: \\ \exists s' \in S_i : s' \geq s, (s', t) \in R}} x_t$$

Using (SC2b), we see that

$$\{t \in S_j \mid \exists s' \in S_i : s' \geq s, (s', t) \in R\} = \{t \in S_j \mid \exists t' \leq t : (s, t') \in R\}$$

This exactly yields Compatibility Constraints (5.4b), which proves that the feasible set of Model (5.3) is included in that of Model (5.4).

To prove the opposite inclusion, we show that every integral solution to (5.4) is indeed feasible for (CPMCS), i.e. whenever $x_s = 1 = x_t$ for $s \in S_i, t \in S_j,\ i \neq j$ we can show that $(s, t) \in R$. Using (5.4b), $x_s = 1$ implies that one element from the set $\{\tilde{t} \in S_j \mid \exists t' \leq \tilde{t} : (s, t') \in R\}$ also must have a value of 1. As only one element from S_j can have value 1 due to (5.4a), the element t has to be contained in that set. Hence, there exists $t' \leq t$ with $(s, t') \in R$. On the other hand, we may swap the roles of s and t and use (5.4b) in order to deduce the existence of $s' \leq s$ with $(s', t) \in R$ with the same arguments. This puts us in a situation to apply (SC2b) to s', s, t' and t, concluding $(s, t) \in R$. □

Note that similar as in the above proof, it can be shown that Constraints (5.4b) for subsets S_i, S_j with $j < i$ are redundant to the corresponding constraint for $j > i$ if $\{t \in S_j \mid \exists t' \leq t : (s, t') \in R\} = \{t \in S_j \mid (s, t) \in R\}$, i.e. elements $s \in S_i$ are compatible to all $t \in S_j$ from a certain element of S_j onwards.

Remark 5.10. Continuing the discussion of Example 5.4, we consider Model (5.4) for the associated problem instance:

$$\begin{array}{lrcl}
\text{find} & x & & \\
\text{s.t.} & x_1 + x_2 + x_3 & = & 1 \\
& x_4 + x_5 + x_6 & = & 1 \\
& x_1 + x_2 + x_3 & \leq & x_4 + x_5 + x_6 \\
& x_2 + x_3 & \leq & x_5 + x_6 \\
& x_3 & \leq & x_5 + x_6 \\
& x_4 + x_5 + x_6 & \leq & x_1 + x_2 + x_3 \\
& x_5 + x_6 & \leq & x_1 + x_2 + x_3 \\
& x_6 & \leq & x_2 + x_3 \\
& x & \in & \{0, 1\}^6.
\end{array}$$

This feasibility problem no longer allows for the fractional solution $(0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2})$ if relaxed to an LP. In fact, it can be checked that the corresponding polyhedron is integral.

Generalizing the observation of Remark 5.10, we now show that the underlying polyhedron of Model (5.4) is always integral.

Theorem 5.11. *The constraint matrix of Model* (5.4) *is totally unimodular.*

Proof: In our proof, we use the following equivalent characterization of total unimodularity:

> A matrix A is totally unimodular, i.e. each square submatrix of A has determinant 0, $+1$ or -1, if and only if each collection of columns of A can be split into two parts such that the sum of the columns in one part minus the sum of the columns in the other part is a vector with entries in $\{0, +1, -1\}$ only (see [GH62] and [Sch86, Theorem 19.3 (iv), p. 269]).

We begin by showing the total unimodularity of the constraint matrix of Model (5.4) for the case of Example 5.4. We will then see that the idea behind the proof directly extends to the general case. Observe that the constraint matrix has a very special structure:

$$\left(\begin{array}{ccc|ccc||rcr|r} \multicolumn{3}{c|}{S_1} & \multicolumn{3}{c||}{S_2} & \multicolumn{3}{c|}{\sum_{\text{alt},S_1} \; + \; \sum_{\text{alt},S_2}} & \sum \\ \hline 1 & 1 & 1 & 0 & 0 & 0 & 1 & + & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & + & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & + & -1 & 0 \\ 0 & 1 & 1 & 0 & -1 & -1 & 0 & + & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & -1 & 1 & + & 0 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 & -1 & + & 1 & 0 \\ -1 & -1 & -1 & 0 & 1 & 1 & -1 & + & 0 & -1 \\ 0 & -1 & -1 & 0 & 0 & 1 & 0 & + & 1 & 1 \end{array}\right),$$

where we have left out the submatrices I and $-I$ for the variable bounds, as they have no effect on total unimodularity. When computing the alternating sum of the columns corresponding to the elements of subset S_1, going backwards and starting with a positive sign in the last column, we observe that this yields a column vector that only consists of entries in $\{0, +1, -1\}$. The same holds for the columns corresponding to the elements of subset S_2. For the rows corresponding to the multiple-choice constraints (5.4a), exactly one of the two column vectors contains an entry $+1$ and the other one an entry 0. For the rows corresponding to compatibility constraint (5.4b) for the elements of S_1, the S_1-column vector contains either a $+1$ or a 0 and the S_2-column vector either a -1 or a 0, and vice versa for the elements of S_2. Thus, when adding the two column vectors, the result is a new column vector whose entries are in $\{0, -1, +1\}$ only. This property still holds when forming a submatrix by deleting individual columns of the constraint matrix due to the staircase structures in the compatibility constraint. Therefore, we have shown the total unimodularity of the matrix.

Now, when considering an arbitrary instance of Model (5.4), we can use the same strategy as above. Given an arbitrary subset of the columns of the constraint matrix,

we partition it according to the partition of S and compute the m column vectors arising when summing the columns in such a partition in a backwards fashion (exploiting the ordering of the subsets S_i), starting with a positive sign for the last element. For the rows belonging to the multiple-choice constraints, exactly one resulting column vector will have an entry of 1, the other an entry of 0. As each row belonging to the compatibility constraint corresponds to the elements of exactly two subsets, at most one column vector will have an entry $+1$, and at most one column vector will have an entry -1. The other entries will be 0. As a result, when summing all the column vectors, the result will be a column vector with entries in $\{0, -1, +1\}$ only. This concludes the proof. □

In many cases, totally unimodular constraint matrices correspond to problems defined on a network. More precisely, the matroid formed by a totally unimodular constraint matrix can be decomposed into matroids that are graphic, cographic, or isomorphic to the special matroid R_{10} (on the decomposition of regular matroids, see [Sey80]) – which is neither graphic nor cographic and rarely occurs in practical applications. Thus, it is natural to ask the question whether the constraint matrix of Model (5.4) is graphic or cographic (i.e., the linear matroid obtained from the matrix is a graphic or cographic matroid), in which case (CPMCS) is equivalent to a network flow problem or a dual network flow problem ('potential problem') respectively. The reader not familiar with those notions of matroid theory may consult [Oxl06].

Theorem 5.12. *The constraint matrix of Model* (5.4) *is cographic.*

Proof: We show this by transforming Model (5.4) into a dual network flow problem. This type of problem has been introduced in Subsection 2.1.2 of Chapter 2. Given a graph $G = (V, A)$, such a problem has the general form

$$\begin{aligned} &\min && c^T \pi && && \\ &\text{s.t.} && \pi_j - \pi_i && \leq && d_{ij} \qquad (\ \forall\, a = (i,j) \in A\ ) \\ &&& \pi && \in && \mathbb{R}^{|V|} \end{aligned} \tag{5.5a}$$

To obtain this form, we use the following variable transformation: let

$$y_{i,j} \coloneqq \sum_{k=j}^{n_i} x_{i,k} \quad (\forall i = 1, \ldots, m)(\forall j = 1, \ldots, n_i).$$

We have seen this transformation already in a different context: it connects the binary variables of different modeling methods for piecewise linear functions (see Subsection 2.2.3); it was a key ingredient to transfer polyhedral results to a formulation according to the Incremental Methods in Subsection 4.2.3. Recall that the transformation is bijective with $x_{i,j} = y_{i,j} - y_{i,j+1}$ if $j < n_i$, and $x_{i,n_i} = y_{i,n_i}$. Stating Model (5.4) in terms of the y-variables, we see that both sides form telescope sums, leaving only one variable on each side. Thus, Compatibility Constraint (5.4b) for two subsets S_i and S_l and some $j \in S_i$ now reads

$$y_{i,j} - y_{l,\min(j,S_l,R)} \leq 0,$$

which has the form of (5.5a). Constraints (SC1) translate to

$$y_{i,1} = 1 \quad (\forall\, i = 1, \ldots, m). \tag{5.6}$$

This also implies upper bounds on the x-variables. Their lower bounds can be expressed via

$$y_{i,j+1} - y_{i,j} \leq 0 \quad \text{if } j < n_i, \quad \text{and } -y_{i,n_i} \leq 0 \quad (\forall i = 1, \ldots, m). \tag{5.7}$$

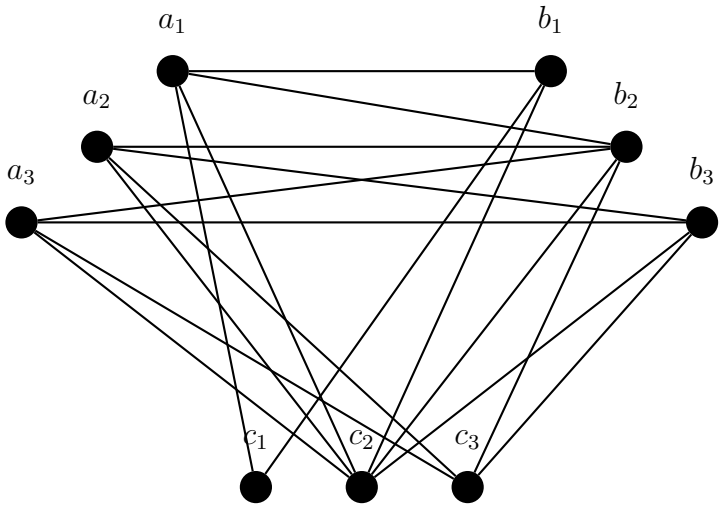
G is simply defined to have a vertex for every y-variable and an arc (i, j) if and only if there is a constraint $y_j - y_i \leq 0$. □

Remark 5.13. The y-variables have the following interpretation: $y_{i,j} = 1$ means: 'from S_i, pick an element with index j or greater'. This is very similar to the *Incremental Method* for linearizing a univariate function (see Subsection 2.2.3). Furthermore, the above transformation is well-known from this context, where it is used to connect the Incremental Method to, for example, the *Convex Combination Method*, and vice versa. We can also recognize (5.7) as the *filling condition*.

In this sense, our dual flow formulation corresponds to an incremental formulation, whereas Model (5.4) is related to the Multiple Choice Method or the Convex Combination Method.

The following example illustrates the transformation of (CPMCS) to a dual network flow problem. It will also show that the constraint matrix is not graphic in general.

Example 5.14. Let S be partitioned into three subsets $A = \{a_1, a_2, a_3\}$, $B = \{b_1, b_2, b_3\}$ and $C = \{c_1, c_2, c_3\}$. Let R be given by the following compatibility graph. Each pair of subsets behaves as in Example 5.4.



As described in the proof of Theorem 5.12, Compatibility constraints (5.4b) transform into Inequalities (5.3), e.g. considering node a_2 together with subset B, the corresponding inequality

$$x_{a_2} + x_{a_3} \leq x_{b_2} + x_{b_3}$$

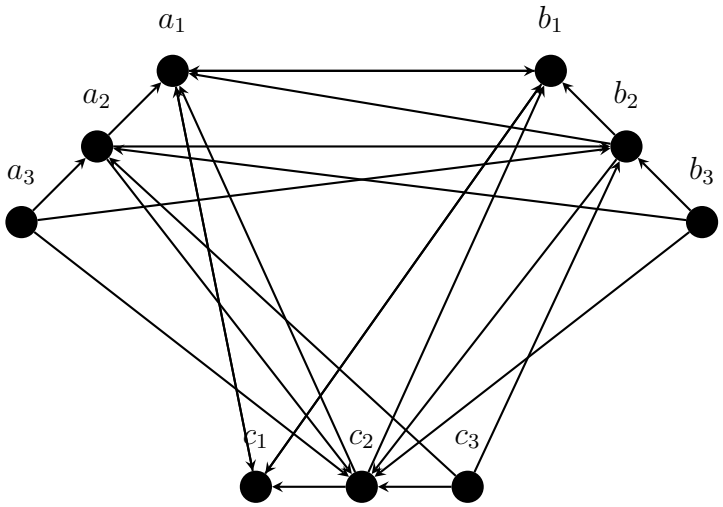
in terms of the y-variables now reads

$$y_{a_2} \leq y_{b_2}.$$

More generally, for every element $s \in S_i$ and every subset $S_j, j \neq i$ we have

$$y_s \leq y_{t(s)},$$

where $t(s) = \min\{t \in S_j \mid (s,t) \in R\}$. Due to (5.7), there are additional constraints ordering the y-variables within each subset. Therefore, by the proof of Theorem 5.12 we can formulate the given instance of (CPMCS) as a dual network flow problem on the following directed graph.



In this graph, arcs (u, v) may be read as implications of the form $(u = 1) \Rightarrow (v = 1)$. The example shows that the constraint matrix of Model (5.4) is not graphic in general, as this would require the above graph to be planar. However, this is not the case, as, for example, it has $K_{3,3}$ as a subgraph using nodes $\{a_2, b_2, c_2\}$ and $\{a_3, b_3, c_3\}$. This implies that the graph has no planar embedding due to *Kuratowski's Theorem*.

As the constraint matrix is totally unimodular, we are guaranteed that each fractional point is the convex combination of integral solutions. Next, we will show how to find such a convex combination. In the case where (CPMCS) forms a substructure of a more complex problem, this may be useful for constructing a heuristic, as the integer points spanning a fractional solution are candidates for good feasible solutions.

The following is an easy way to obtain integral solutions from a fractional one. It generalizes the well-known fact that for dual network flow problems, rounding all components up or rounding all components down preserves feasibility.

Definition 5.15. Let $\hat{y}$ be a solution to (CPMCS), and let $\lambda \in (0, 1]$ be some threshold value. We define $\hat{y}_\lambda$ to denote the integer point obtained from rounding all components of $\hat{y}$ according to the following rule:

$$\hat{y}_{\lambda,i} = \begin{cases} 1 & \text{if} \quad \hat{y}_i \geq \lambda \\ 0 & \text{if} \quad \hat{y}_i < \lambda \end{cases},$$

and say that $\hat{y}_\lambda$ is obtained from *λ-rounding* $\hat{y}$.

It is easy to see that $\hat{y}_\lambda$ is also a solution to (CPMCS), for all $\lambda \in (0, 1]$. The key observation is that every operation that does not change the relative ordering of the y_{ij} (and also does not violate the $0-1$-bounds), preserves feasibility, as $d_{ij} = 0$ in (5.5a) whenever there are two variables present in the constraint.

Theorem 5.16. *Let $\hat{y}$ be a solution to (CPMCS). Then $\hat{y}$ is a convex combination of the integral solutions*

$$\{\hat{y}_\lambda \mid \lambda \in \{\hat{y}_i,\ i = 1, \ldots, |S|\}\}.$$

Proof: Let $\Lambda \coloneqq \{\hat{y}_i,\ i = 1, \ldots, |S|\}$ denote the set of values occurring in $\hat{y}$. We denote them by $\lambda_1, \ldots, \lambda_{|S|}$ and assume they are ordered increasingly, i.e. $\lambda_i \leq \lambda_j$ whenever $i \leq j$. We claim that

$$\hat{y} = \sum_{k=1}^{|S|} (\lambda_k - \lambda_{k-1}) \hat{y}_{\lambda_k}, \tag{5.8}$$

where λ_0 is interpreted as 0. Indeed, the i-th component of $\sum_{k=1}^{|S|} (\lambda_k - \lambda_{k-1}) \hat{y}_{\lambda_k}$ is equal to

$$\begin{aligned} \sum_{k=1}^{|S|} (\lambda_k - \lambda_{k-1}) \hat{y}_{\lambda_k,i} &\overset{\text{Def. 5.15}}{=} \sum_{k:\lambda_k \leq \hat{y}_i} (\lambda_k - \lambda_{k-1}) 1 \\ &\overset{\text{telescope sum}}{=} \max_{k:\lambda_k \leq \hat{y}_i} \lambda_k - \underbrace{\lambda_0}_{=0} \\ &= \hat{y}_i. \end{aligned}$$

Furthermore, we have

$$\lambda_k - \lambda_{k-1} \geq 0 \text{ for all } k = 1, \ldots, |S|,$$

and also

$$\sum_{k=1}^{|S|} (\lambda_k - \lambda_{k-1}) = \lambda_{|S|} - \lambda_0 = 1,$$

since (5.6) implies $1 \in \Lambda$, and therefore $\lambda_{|S|} = 1$. Thus, Equation (5.8) describes $\hat{y}$ as a convex combination of $\{\hat{y}_\lambda \mid \lambda \in \Lambda\}$. □

5.4 Computational Results

In this section, we compare the efficiency of the three MIP formulations for (CPMCS) we have discussed previously: the first, naive compatibility formulation (5.3), the totally unimodular compatibility formulation (5.4) and the formulation as a dual network flow problem. We do this by evaluating them on real-world benchmark instances. First, we consider an application on energy-efficiency of a railway timetabling. After that we revisit instances from the context of piecewise linearization of the physical flow constraints on gas networks—though this latter application will be discussed very briefly due to the detailed coverage in Chapter 4. We will see that passing from the original to the unimodular formulation already brings a significant computational advantage, but that the sparsity of the dual-flow formulation allows for the best results by far. Our computational study thus immediately shows two more things: staircase structures are present in real-world application problems and their exploitation is very beneficial in terms of computation time.

5.4.1 Computational Results for Energy-Efficient Timetabling

The first example for a successful exploitation of staircase compatibility we present here is a problem in railway timetabling. The aim is to take a preliminary timetable which is currently in the planning phase (typically towards the end) and to use the remaining degrees of freedom to allow for a reduction of the energy costs of the involved train operating companies (TOCs). This is possible by taking into account that a big consumer of electricity—as a TOC undoubtedly is—typically has an electricity contract consisting of two price components: the overall energy consumption and the maximum average power consumption over all 15-minute intervals in the billing period. In the special case of a German TOC, the electricity provider charges the collective consumption of all the trains operated by this TOC. This is done by summing up their individual power consumption profiles as measured by the electricity meters in the locomotives and computing both the area under the resulting curve, i.e. the total energy consumption, as well as the maximum 15-minute average. Both values are multiplied with some cost factor and summed to obtain the final electricity bill. One possibility for optimization via timetabling now lies in adjusting the departure times of the trains in the stations. A train generally draws most power while accelerating. Thus, high peaks in consumption can be avoided if too many simultaneous departures are desynchronized, which can be used to decrease the price component based on peak consumption. In many cases, this effect can already be achieved via small shifts in the departure times and is thus an interesting trade-off to be considered. The power-based price component typically makes up for 20–25 % of the energy bill.

We illustrate the effect of this optimization in Figure 5.2. It shows the power consumption profile before and after optimization for one of the benchmark instances introduced later (*Würzburg*) on a sample day. The curves in red show the power consumption in each second, while the blue curves show their consecutive 15-minute averages. As stated, the TOC is charged proportionally to the highest such average

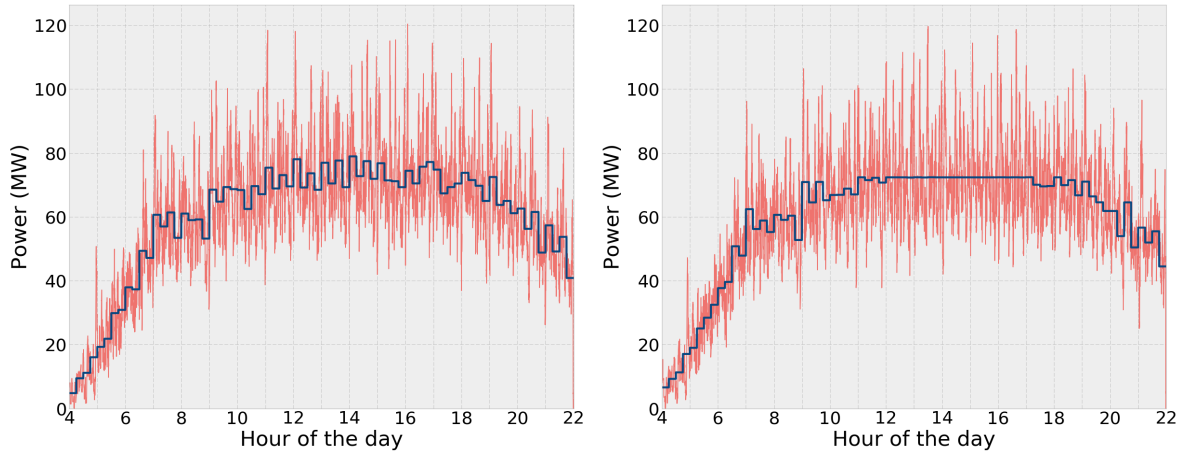


Figure 5.2: Power consumption profile of timetabling instance *Würzburg* before (left) and after (right) optimization. The total power consumption per minute of all trains is shown in red. 15-minute averages (relevant for billing) are displayed in blue.

over the billing period (typically one year). According to the official price sheet by *DB Energie GmbH* for 2016, the cost factor is 120.83 € per kW and year such that the demonstrated reduction from 87 to 80 MW in peak consumption equals an annual cost saving of around $850,000$ € (and this is a rather small instance). Note that the energy recuperated from braking trains is refunded separately and not offset against the power drawn from the power supply. Thus, we can assume that the consumption profile is always non-negative.

In the following, we give a statement of the problem in terms of staircase compatibility and present a computational study on problem instances of different sizes.

The Problem as a Special Case of (CPMCS) We consider a given initial timetable in which each departure time of a train from a station may be shifted within some interval around the current departure time. We assume that the travel times of the trains on the tracks as well as the corresponding power consumptions are fixed. Furthermore, we assume that the temporal order of the trains passing a certain track may not be changed by the optimization and that all connections between different trains in a station must be preserved in order to maintain the structure of the original timetable as far as possible. Assuming a fixed order of the trains on each track, we also know the safety distances to respect between each consecutive pair of trains. The problem is now to find an adjusted timetable that minimizes the maximum average power consumption.

In order to state this problem in terms of (CPMCS), we need to define the basic set S and the compatibility relation R. Let D be the set of all trains, V^d the set of all stations from which train $d \in D$ departs and A^d the tracks it uses. Let furthermore $J^{dv} \subseteq T$

denote the set of all feasible departure times for train $d \in D$ from station $v \in V^d$ within a given planning horizon T. We choose S to be the set of all triples (r, v, j) of a train $d \in D$ and its feasible departure times $j \in J^{dv}$ from some of its stations $v \in V^d$. It is then natural to choose the partition $S = \bigcup_{(d,v):d\in D, v\in V^r} S_{dv}$, where S^{dv} are all feasible triples (d, v, j) for some fixed d and v. A feasible timetable is then made up of a selection of exactly one element from each subset S^{dv}:

$$\sum_{j\in S^{dv}} x_j^{dv} = 1 \quad (\forall d \in D)(\forall v \in V^d).$$

To be feasible, this selection has to respect several further constraints which are stated in the following. The travel time for a train $d \in D$ to pass a track $a = (v, w) \in A^d$ on a journey between two stations $v, w \in V^d$ is Γ^{da}, and after arriving at station w it has to stop for a minimum time of c^{dv}. For each pair of consecutive trains (d_1, d_2) on a track between two stations v and w, as given by a set L^{vw}, we have to keep a minimum headway time of $s^{d_1d_2vw}$. Finally, for each station $v \in \bigcup_{d\in D} V^d$ where a pair of trains (d_1, d_2) meets such that the time that passes between the arrival of d_1 and the departure of d_2 is at least $\rho^{d_1d_2v}$ and at most $\theta^{d_1d_2v}$, as given by a set U_v, this property has to be preserved in the new timetable to maintain the possibility to change between the two trains.

The relation R stating the compatibility between two elements $r_1 = (d_1, v_1, j_1), r_2 = (d_2, v_2, j_2) \in S$ is now given by

$$R = R_1 \cap R_2 \cap R_3.$$

Here, relation R_1 models the compatibility according to the minimum stopping times:

$$R_1 = \big\{(r_1, r_2) \in S \times S \mid d_1 = d_2 =: d, v_1 = v_2 =: v \in V^d, (v, w) =: a \in A^d, \\ j_2 \geq j_1 + \Gamma^{da} + c^{dw}\big\},$$

relation R_2 models the compatibility according to the minimum headway times:

$$R_2 = \big\{(r_1, r_2) \in S \times S \mid v_1 = v_2 =: v \in V^d, (v, w) \coloneqq a \in A^{d_1} \cap A^{d_2}, (d_1, d_2) \in L^{d_1d_2a}, \\ j_2 \geq j_1 + s^{d_1d_2a} + \min(\Gamma^{d_1a} - \Gamma^{d_2a}, 0)\big\}$$

and relation R_3 models the compatibility according to the connection times:

$$R_3 = \big\{(r_1, r_2) \in S \times S \mid (v_1, v_2) \in A^{d_1}, (d_1, d_2) \in U^{v_2}, \\ j_2 \geq j_1 + \Gamma^{d_1a} + \rho^{d_1d_2v} \wedge j_2 \leq j_1 + \Gamma^{d_1a} + \theta^{d_1d_2v}\big\}.$$

It is easy to check that each of the three relations R_1, R_2 and R_3 is a staircase relation on S. Likewise, it is easy to check that the intersection of any number of staircase relations is again staircase. Consequently, R is a staircase relation on S, which allows us to formulate the set of feasible selections according to each of the three models derived in Section 5.3.

What is left to define is the objective function. Let $p^{dat} \geq 0$ be the consumption of train d when passing track $a = (v, w) \in A^r$ at point $0 \leq t \leq \Gamma^{da}$ after departure. Consequently, if train d departs from station v at time j, the consumption at point $t \in T$ is given by:

$$\bar{p}_j^{dat} = \begin{cases} \max(p^{dat}, 0), & 0 \leq t - j \leq \Gamma^{da} \\ 0, & \text{otherwise.} \end{cases}$$

Let $I = \{1, 2, \ldots, m\}$ be the set of the m consecutive 15-minute (= 900-second) intervals in T (where the last interval may actually be somewhat shorter). The total energy consumption of a train $d \in D$ on track $a \in A^r$ within an averaging interval $i \in I$ when choosing departure time $j \in J^{dv}$ is then given by $e_j^{dai} = \frac{1}{2}(\bar{p}_j^{da,900i} + \bar{p}_j^{da,900(i+1)}) + \sum_{900i+1 \leq t \leq 900(i+1)-1} \bar{p}_j^{dat}$ (we consider the consumption p as a piecewise-linear function over time). The average power consumption over an interval $i \in I$ by all trains $d \in D$ depending on the chosen departure times is then given by

$$z_i(x) = \frac{1}{900} \sum_{d \in D} \sum_{a=(v,w) \in A^d} \sum_{j \in J_v^d} e_j^{dai} x_j^{dv}.$$

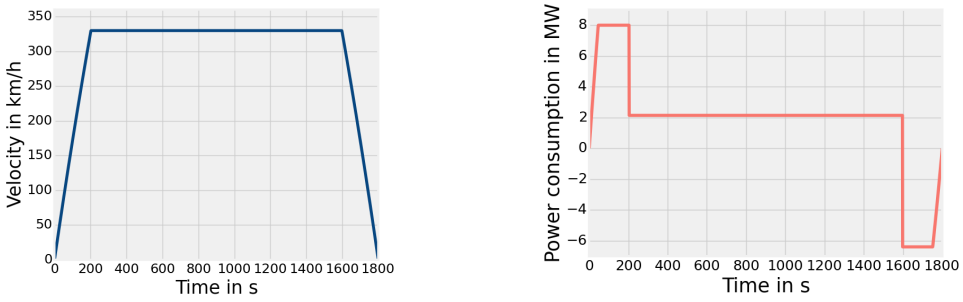
This leads to the following optimization problem to minimize the highest of these averages:

$$\min_{x \in X} \max_{i \in I} z_i(x),$$

where X is the set of all feasible timetables. For this set X, we can now choose between one of the three models for staircase compatibility derived in Section 5.3. Note that this timetabling problem is NP-hard even if all trains only have one track and $m = 2$ as can easily be shown by a reduction from the partition problem (see [GJ79, Problem *SP12*]).

Computational Comparison of the Models for (CPMCS) We now present a computational study that compares the different formulations for staircase compatibility considered before as a part of the timetabling problem introduced above. We do this on real-world instances derived from the 2015 timetable for the German passenger traffic operated by the industry partner *Deutsche Bahn AG (DB)*. We complemented this data by power consumption profiles based on height data of the stations as well as simplified speed profiles taking into account train characteristics. An example is depicted in Figure 5.3 which shows an assumed speed profile for an ICE-3 on a journey of 30 minutes in Figure 5.3a and the corresponding power profile on a track with an upwards inclination in Figure 5.3b. The minimum headway times we chose are based on [Pac16, Table 5.4] by rounding up the given values to full minutes.

Altogether, we have created 31 instances of different sizes, each for a planning horizon of 18 hours (4am to 10pm). These contain 18 *local instances* which contain all trains passing a certain station in Germany, 1 *Fernverkehr* instance covering the German long-distance traffic, 10 regional instances which contain all short-distance trains circulating in a given region of Germany, 1 *Regionalverkehr* instance covering all of the German short-distance traffic, as well 1 *Germany* instance covering all German DB passenger trains. Each instance contains those parts of the journeys of the involved trains



(a) Speed profile

(b) Power consumption profile

Figure 5.3: Example profiles for an ICE-3 on a 30 minutes journey climbing an inclination

which fall within the planning horizon. The allowable shift in departure time was uniformly chosen to be ± 3 minutes around the current departure time. The sizes of the created instances, the computation times of the three different models as well as the achieved savings in peak power consumption are shown in Table 5.1. Here, NA denotes the naive formulation (5.3), TU stands for the totally unimodular formulation (5.4), and DF represents the formulation as a dual network flow problem.

The computational experiments have been performed on a queuing cluster of AMD Opteron 6134 $2,3$-GHz computers with a total of 16 CPU cores on each cluster node, and 128 GB of main memory. The implementation uses the python interface of Gurobi 7.0 [Gur17], where the solver is assigned 5 of the cores, and each of the instances was run in exclusive mode on a cluster node (hence, all 16 nodes may be used for building up the large model, which does not count towards the solution time).

The result is very clear: Formulation DF is by far the best way to formulate the compatibilities as it leads to the fastest solution times on all but a few instances. In many cases, the benefit is very significant. Most notably, the Germany-wide instance can be solved within $\sim$ 23 minutes via the dual flow formulation, whereas Formulation NA cannot solve this instance to optimality within the time limit of 10 hours. The table also shows that the computation time of Formulation TU is usually between the solution times required for Formulations NA and DF. This shows the general benefit of passing to a totally unimodular description of the set of feasible timetables. However, the sparsity of Formulation DF leads to much lower node solution times in the branch-and-bound tree and is therefore vastly superior. We remark here that the stated reduction of about 5 % in peak power consumption for the Germany-wide instance would allow for cost savings of several million euros per year. More detailed information on the problem can be found in [BMS17].

Instance	#Trains	#Trips	Computation time [s] NA	TU	DF	Sav. [%]
Zeil	42	762	24.65	6.18	**1.42**	14.89
Bayreuth Hbf	68	327	1.49	0.79	**0.34**	22.18
Passau	75	1 040	1 835.38	355.94	**28.49**	14.48
Jena Paradies	78	1 102	301.73	76.97	**13.82**	12.46
Lichtenfels	113	1650	1359.2	472.94	**43.79**	15.25
Erlangen	142	2 969	9 511.88	1 127.05	**108.92**	15.28
Bamberg	209	3 644	-	273.71	**46.86**	13.07
Aschaffenburg	245	3 463	356.94	81.74	**7.06**	12.95
Kiel Hbf	297	2 130	251.70	58.80	**8.35**	11.19
Leipzig Hbf (tief)	369	6 810	27.24	29.45	**4.39**	6.40
Würzburg Hbf	371	4 456	-	29 817.58	**2 047.97**	8.32
Dresden	422	6 936	-	1 639.76	**401.61**	9.29
Ulm Hbf	468	5 729	13 790.59	156.63	**17.08**	11.23
Stuttgart Hbf (tief)	628	11 594	3 962.31	2 187.78	**41.14**	0.93
Berlin Hbf (S-Bahn)	639	16 114	**57.53**	203.62	316.23	2.97
Hamburg-Altona(S)	722	12 373	1 756.98	460.26	**52.50**	1.29
Frankfurt(Main) Hbf	728	8 626	-	2 515.70	**111.80**	10.28
Nürnberg	951	12 189	1 519.30	72.26	**16.64**	7.10
S-Bahn Hamburg	1 208	17 533	3 788.40	795.17	**247.73**	2.36
Regio Nord	1 476	13 379	309.98	99.33	**23.01**	12.79
Regio Nordost	1 494	16 496	8 659.40	265.17	**30.26**	15.50
Regio Hessen	1 547	25 092	**93.57**	198.95	181.63	5.64
Regio Suedwest	1 863	24 191	1 382.92	252.55	**33.11**	13.00
Regio Suedost	2 357	31 917	3 685.06	311.43	**73.26**	8.96
Regio BW	2 382	30 172	9 649.42	904.94	**181.57**	13.36
S-Bahn Berlin	2 578	53 353	**234.73**	1 653.37	4 676.73	1.73
Regio NRW	2 826	47 026	9 256.70	2 104.53	**337.56**	5.13
Regio Bayern	3 554	49 262	1 415.59	578.32	**567.86**	10.72
Fernverkehr	667	7 053	819.13	216.45	**26.09**	5.38
Regionalverkehr	21 288	308 472	-	22 728.63	**19 997.96**	9.52
Germany	21 955	315 525	-	22 204.67	**1 369.89**	5.05

Table 5.1: Computational results for the energy-efficient timetabling problem showing the solving time[s] for the three problem formulations as well as the number of trains and trips and total amount of energy saved ('Sav. [%]') for each instance.

5.4.2 Computational Results for Piecewise Linearized Path Flows

Another example for a staircase compatibility structure originates from the setting that has been investigated in Chapter 4 (please see Section 4.1 for an introduction to this setting and a definition of the polytope considered in Chapter 4). Among others, we were able to derive a complete description for the case of a network that is a path of arbitrary length (see Subsection 4.2.2). In the current chapter, we assume that the intervals on each network arc result from subdivision of a larger interval. This has also been covered as a special case in Chapter 4, though using different reasoning. We have already seen in Subsection 5.2.1 that this represents a special case of (CPMCS).

Computational Comparison of the Models for (CPMCS) We have already seen in Chapter 4 that there is a significant impact of using complete descriptions for substructures for instances arising from piecewise linearized network flow problems. We will reconsider some of the computations with an emphasis on the results from this chapter.

We use the same setting for our test instances as in Chapter 4 (see Subsections 4.3.1 on the separation routine PATHCUT and 4.3.2 for the generation of instances). In particular, we first identify all suitable subpaths of degree-two nodes in the network, construct the corresponding compatibility graphs and precompute the unimodular formulation of Model (5.4) for each of the detected subpaths. This description is quadratic in the length of the path and linear in the number of intervals per arc.

The underlying network is given by the topology of a real-world gas network by the German gas network operator *Open Grid Europe* (OGE) consisting of 592 nodes and 623 arcs. It has also been used for computational experiments in Subsection 4.3.4. As the network is not a path, there is no complete description available. However, (CPMCS) is present as a substructure, e.g. at each induced path of degree-two nodes in the network. 224 nodes have degree two and there are 128 paths of degree-two-nodes, which amounts to an average length of 2.75. The longest of those paths has length 8. In the following, we want to test the effect of using our improved formulations of (CPMCS) in those places.

Using results from this chapter we may reformulate subpaths of degree-two nodes in the network using either the totally unimodular formulation (5.4) or the formulation as a dual network flow problem. As in the previous subsection, those will be denoted by TU and DF, respectively. Formulation TU naturally uses variables from the Multiple Choice Method whereas the choice of variables in DF can be associated with the Incremental Method as a linearization method. Note that the transformation used to obtain the formulation DF and to prove Theorem 5.12 is exactly the same that connects the Incremental Method to the Multiple Choice Method (see Remark 5.13). Therefore, the Incremental Method will be used for computations on DF.

Table 5.2 (which is an excerpt from Table 4.3 in Chapter 4) shows the effect of separating constraints from formulation TU compared to a standard formulation obtained from applying the Multiple Choice Method. Adding constraints from the TU formula-

# intervals per arc	MCM solved	MCM CPU[s]	MCM + TU-paths solved	MCM + TU-paths CPU[s]
3	**5**	**66.63**	5	73.13
4	5	4 943.87	**5**	**468.66**
5	5	9 001.10	**5**	**1 627.76**
6	2	31 384.31	**3**	**10 089.66**
7	1	103 191.92	**2**	**122 299.54**
8	0	∞	0	∞

Table 5.2: Number of instances solved and average solution times for instances on a gas network topology with 592 nodes and a varying number of intervals per arc.

tion of the (CPMCS)-substructures (i.e. paths of degree-two nodes) improves the runtime of the solver considerably for most test sets. This effect increases with a growing number of intervals per arc, resulting in a total of 2 more instances that can be solved within the time limit.

In the following, we compare the standard formulation with and without adding constraints from the totally unimodular dual-flow formulation (which naturally uses the binary y-variables of the Incremental Method).

# intervals per arc	INC solved	INC CPU[s]	INC + TU-paths solved	INC + TU-paths CPU[s]
4	**5**	**5.61**	5	6.10
5	5	13.73	**5**	**10.50**
6	5	141.02	**5**	**41.96**
7	5	197.94	**5**	**68.49**
8	5	1424.02	**5**	**195.95**
9	5	1144.44	**5**	**857.59**
10	5	25506.75	**5**	**837.45**
12	3	85712.83	**5**	**3048.45**
15	0	∞	**5**	**44275.51**
20	0	∞	**1**	**824.18**
25	0	∞	0	∞

Table 5.3: Number of instances solved and average solution times for instances on a gas network topology with 592 nodes and a varying number of intervals per arc, using the Incremental Method.

The results can be found in Table 5.3 (which has not been shown in Section 4.3). Using the Incremental Method reduces the overall runtime by a large factor such that instances up to 12 intervals per arc (20 with the TU formulation on paths) can now

be solved. This agrees with the results in Subsection 4.3.5 obtained on random scale-free networks. It also gives an additional argument for the dual-flow formulation, as its variables seem to suit solvers well in this context. Remember that even without using our totally unimodular description, switching to the Incremental Method leads to faster solution times. Providing the solver with the TU-formulation on paths again increases the performance of the solver significantly. For more computational experiments on piecewise linearized flow problems the reader may reconsider Section 4.3.

The results on the application of piecewise linearized flow problems show that the TU-formulations can have a large benefit, not only if the feasible set—as in the last subsection—can be described as (CPMCS) as a whole, but also if (CPMCS) is present as a substructure.

5.5 Recognizability of Staircase Relations

The previous section has shown that for two example applications, where (CPMCS) is present as a substructure, using totally unimodular formulations—in particular the dual flow formulation—for (CPMCS) represents a significant improvement over a naive formulation and can vastly reduce solution time. With these insights, one may aim to identify (CPMCS) within more applications or even detect it automatically in general MIPs in order to do a reformulation. In this section, we want to address questions related to recognizability of staircase compatibility.

5.5.1 Complexity of Recognition Problems

Staircase compatibility according to Definition 5.3 is not a property of a graph, but rather a joint property of a graph together with a partition and together with a total ordering on each partition. Therefore, the question of recognizability can be stated on different levels, depending on which of those features are already fixed.

At the most restrictive level, we may assume that for a graph $G = (V, E)$ the partitioning $\mathcal{S} = \{S_1, \dots, S_k\}$ for some $k \in \mathbb{N}$ as well as total ordering on each partition are already fixed.

For those questions of recognizability Lemma 5.5 is very useful. It tells us that in case of (5.1), i.e. if each element s of a subset S_i in the partition of S has as least one element in each of the remaining subsets with which it is compatible, (SC2b) implies (SC2a). With the partitioning $\mathcal{S}$ given, it is easy to check (5.1). Moreover, confirming that elements belonging to the same partition are incompatible according to R can also be done efficiently. Therefore, we will consider it sufficient to check for (SC2b) to decide whether a given relation is a staircase relation; moreover, we will denote (SC2b) by (SC2).

The first observation is that the recognition problem on the lowest level is easy:

Proposition 5.17. *Let a partitioning of S into $\mathcal{S} = \{S_1, \dots, S_k\}$ for some $k \in \mathbb{N}$ total orderings on each partition $S_i, i = 1, \dots, k$ be given. The problem to decide whether a given relation R is a staircase relation, can be solved in polynomial time.*

Proof: As discussed above, we can reduce the problem to checking (SC2b). However, by the structure of this condition, we can simply select any subset consisting of four elements $a_1, a_2 \in S_i,\ b_1, b_2 \in S_j$ with $a_1 < a_2, b_1 < b_2$ from two partitions S_i and S_j and check the implication from (SC2b), i.e. if either $(a_1, b_2) \notin R$, $(a_2, b_1) \notin R$ or $(a_1, b_1) \in R \wedge (a_2, b_2) \in R$. This clearly can be done in polynomial time, e.g. in $\mathcal{O}(|S|^4)$, though this bound is quite rough due to the special membership structure that is required for a_1, a_2, b_1, b_2 in order to satisfy the assumption of (SC2b). We can improve it by only considering 4-tuples of nodes that form 2 'crossing' edges (see our colloquial explanation of (SC2b) in Example 5.4). This leads to a (still rather pessimistic) bound of $\mathcal{O}(|R|^2)$. □

Remark 5.18. In case a relation is not a staircase relation, we could want to form a *staircase relaxation* by adding compatibilities to R. The Problem (CPMCS) for the resulting staircase relation R' then is a relaxation of (CPMC) for the original relation R. In particular, the cuts from the totally unimodular formulation (5.4) are still valid for the original problem. They are merely insufficient for obtaining a complete description.

There is a unique optimal *staircase relaxation* in the sense that any other staircase relaxation strictly contains the minimal one: It can be obtained by repeatedly applying (SC2b), i.e. adding the edges that would be implied by (SC2b) but are not yet present, to the compatibility graph. Every edge added that way has to be contained in any staircase relaxation of R, hence the minimal *staircase relaxation* is found after (SC2b) is finally satisfied. As the number of edges that can be added during this process is finite, this relaxation is found after polynomially many steps.

We may also state the recognizability question for the case where the partitioning $\mathcal{S}$ is given but the orderings on each partition are yet to be determined. This problem is a lot more challenging. The following hardness result can be given:

Theorem 5.19. *Let a partitioning of S into $\mathcal{S} = \{S_1, \dots, S_k\}$ for some $k \in \mathbb{N}$ and a relation R on S be given. The problem to decide whether there are total orderings on each partition $S_i, i = 1, \dots, k$ such that R satisfies* (SC2b)*, is NP-complete.*

Proof: We show the theorem by a polynomial-time reduction from the *Betweenness Problem*, referred to as BETWEENNESS in this proof. In this problem, a ground set U is given together with a set $\mathcal{U}$ of ordered triples with elements from U. One has to determine whether there is a linear ordering of U such that the middle item of each given triple is placed somewhere between the other two items. This problem is NP-hard [Opa79].

From an instance of BETWEENNESS, we construct an equivalent instance of the recognition problem from Theorem 5.19 as follows: We have a special partition $S_0 = U$, where each node is identified with an item from BETWEENNESS. For each triple $t = (u, v, w) \in \mathcal{U}$, we construct a partition consisting of 3 nodes each that are identified

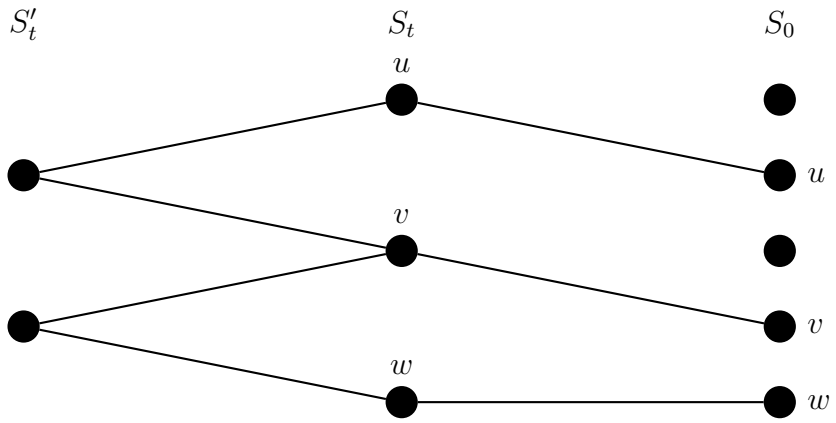


Figure 5.4: Construction of the compatibility graph for the proof of Theorem 5.19, showing partitions S_0 as well as S_t and S'_t for a single triple $t = (u, v, w)$.

with the items from the respective triple (but of course different from the nodes in S_1). Each node in such a partition is solely compatible with their respective copy in S_0. Moreover, each partition S_t representing a triple t from $\mathcal{U}$ comes together with an auxiliary partition S'_t consisting of two nodes. This partition S'_t has nontrivial compatibilities only with S_t and those are designed to force v_t, the designated middle element of the triple, to be sorted in between the other two elements in S_t (see Figure 5.4). Note that there are exactly two possibilities for choosing orderings on S_t and S'_t. Those correspond to the two possible orderings of t that put v in the middle. The compatibility relation between partitions corresponding to different triples may be chosen as complete bipartite graphs such that (SC2b) is trivially satisfied for those pairs of partitions.

If there is a solution to BETWEENNESS, this gives us a linear ordering on U that we can use for S_0. This ordering will ensure that the middle element of each S_t for $t \in \mathcal{S}$ is between the other elements of t. Therefore, there will be no crossing of edges between S_t and S_0 if the ordering on S_t and S'_t is oriented correctly. On the other hand, every choice of orderings on the partitions that lead to R being a staircase relation immediately gives us a solution to BETWEENNESS. Moreover, the transformation of instances is clearly polynomial in the encoding size of the input. □

Please note that the instances of (CPMCS) in the proof of Theorem 5.19 do not satisfy (5.1) such that it does not prove that deciding whether R is a staircase relation is hard. Still, it gives quite a strong hint that this question is also difficult.

Even if there are no orderings such that a given relation R is a staircase relation we might still want to determine an ordering that gets R as close to being a staircase relation as possible. On the one hand, this either provides a good basis for constructing a staircase relaxation as described in Remark 5.18. On the other hand, a reformulation using the variables from the dual flow formulation might already help a general-purpose MIP solver—similar as using the Incremental Method might speed up the solution process in case a sensible ordering is available as it has been observed in Sub-

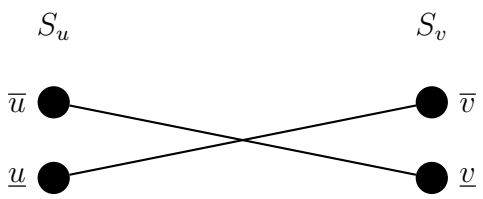


Figure 5.5: Construction of the compatibility graph for the proof of Theorem 5.20, showing partitions S_u and S_v for $u, v \in E(H)$.

section 5.4.2. A natural way to measure the quality of a choice for the total orderings consists of minimizing the number of violations of (SC2). However, this turns out to be NP-hard, even in a severely restricted case:

Theorem 5.20. *Let a partitioning of S into $\mathcal{S} = \{S_1, \ldots, S_k\}$ for some $k \in \mathbb{N}$ and a relation R on S be given. The optimization problem to determine total orderings on each partition $S_i, i = 1, \ldots, k$ such that the number of violations of (SC2) is minimized is NP-hard, even when the problem is restricted to at most two elements per partition.*

Proof: We show the theorem by a polynomial-time reduction from the famous problem MAXCUT. For a given graph H, it asks for partitioning the nodes into two sets such that the number of arcs between both partitions is maximized. The weighted version of the corresponding decision problem was one of Karp's 21 NP-complete problems [Kar72], and also the unweighted case is well known to be NP-hard [GJ79].

From an instance of MAXCUT on a graph H, we construct an equivalent instance of the optimization problem from Theorem 5.20 with only two elements per partition: For each $v \in V(H)$, construct a partition $S_v = \{\underline{v}, \overline{v}\}$ consisting of the two elements $\underline{v}$, $\overline{v}$. Define the compatibility relation R as follows: for $u, v \in V(H)$, but $(u, v) \notin E(H)$, compatibilities are chosen to be trivial, say S_u, S_v form a complete bipartite graph. In case $(u, v) \in E(H)$, let $(\underline{u}, \overline{v}) \in R, (\overline{u}, \underline{v}) \in R$ and $(\underline{u}, \underline{v}) \notin R, (\overline{u}, \overline{v}) \notin R$ (see Figure 5.5).

Therefore, sorting $\underline{u} < \overline{u}$ and $\underline{v} < \overline{v}$ would contribute exactly one violation of (SC2) to the objective, while flipping one of the orderings (but not both) satisfies (SC2) for the two partitions S_u and S_v. As a consequence, any cut in H containing l edges with node partitions V_1, V_2 corresponds to a choice for total orderings on the partitions $S_v,\ v \in V(H)$ with $|E(H)| - l$ violations, namely $\underline{v} < \overline{v}\ \forall v \in V_1$ and $\underline{v} > \overline{v}\ \forall v \in V_2$, and vice versa. □

5.5.2 An MIP formulation for the Recognition Problem with Fixed Partitioning

Despite Theorems 5.19 and 5.20 we may still want to solve the problem of determining suitable total orderings one each partition reasonably well in practice. In this subsection, we will develop an MIP-model to do so. We deploy a modeling that is well known from the *Linear ordering Problem* (LOP) [MR11]. For each pair of elements

s, s' within the same partition S_i, we have a binary variable $z_{s,s'}$ that encodes the relative ordering of s and s', i.e.

$$z_{s,s'} = \begin{cases} 1 & \text{if} \quad s < s' \\ 0 & \text{if} \quad s' < s \end{cases}$$

according to the total order on S_i. We have to make sure that z variables are consistent within each partition, in particular

$$z_{s,s'} = 1 \Leftrightarrow z_{s',s} = 0$$

and exclude cycles in the relative ordering, i.e.

$$z_{u,v} = 1 \wedge z_{v,w} = 1 \Rightarrow z_{u,w} = 1$$

for elements u, v, w from the same partition. From the theory of directed complete graphs (so-called *tournaments*), this is known to be sufficient to characterize a valid linear ordering on each partition.

In addition to the linear ordering constraints we have to model (SC2). However, this can be done by constraints with a surprisingly simple structure. Given two edges $(\underline{a}, \overline{b}), (\overline{a}, \underline{b})$ in the compatibility graph with $\underline{a}, \overline{a} \in S_i,\ \underline{b}, \overline{b} \in S_j$ for $i \neq j$, we can check easily whether a given ordering, say $\underline{a} < \overline{a}$ and $\underline{b} < \overline{b}$ would violate (SC2) as mentioned in Proposition 5.17. In that case, we can derive the implications $(\underline{a} < \overline{a}) \Rightarrow (\underline{b} > \overline{b})$ and $(\underline{b} < \overline{b}) \Rightarrow (\underline{a} > \overline{a})$. This translates to $z_{\underline{a},\overline{a}} = z_{\overline{b},\underline{b}}$. One can check quickly that at most one such constraint is needed for each pair of edges in the compatibility graph. This leads to the following MIP formulation for the recognition problem from Theorem 5.19:

$$\begin{aligned}
&\text{(5.9a)} \quad \text{find} \quad z \\
&\text{(5.9b)} \quad \text{s.t.} \quad z_{u,v} + z_{v,u} = 1 && (\ \forall\, u \neq v \quad \text{in the same partition}\ ) \\
&\text{(5.9c)} \quad z_{u,v} + z_{v,w} + z_{w,u} \leq 2 && (\ \forall\, u \neq v \neq w \text{ in the same partition}\ ) \\
&\text{(5.9d)} \quad z_{s,s'} - z_{t',t} = 0 && (\ \forall s < s', t < t' :\ \text{(SC2) is violated}\ ) \\
&\text{(5.9e)} \quad z_{s,s'} \in \{0,1\}. && (\ \forall\, (s,s') \in \textstyle\bigcup_{i=1}^{m} (S_i \times S_i)\ )
\end{aligned}$$

Constraints (5.9b) and (5.9c) model linear ordering problems on each partition. Condition (SC2) is covered by Equations (5.9d). Note that (5.9d) and (5.9b) are ideal to be used for substitution (cf. [FM05, paragraph on *Aggregation*]), which effectively reduces the number of variables. The remainder is a restricted LOP. This type of problem is well studied and there are a number of exact and heuristic methods to solve LOPs effectively in practice, see e.g. [CML15, SS05] and the references therein.

Remark 5.21. Formulation (5.9) can be adapted to solve the optimization version of the recognition problem from Theorem 5.19. A direct way would consist of replacing the right-hand side of (5.9d) with slack variables $y_{(s,s',t,t')}$ and minimizing the 1-norm of y. For this modified problem, even heuristic or approximate solutions might be very helpful to obtain a good ordering in case an instance turns out to be too hard to solve to optimality.

5.5.3 On Defining Staircase Graphs

Moving to the highest level of recognition problems related to staircase compatibility, we may ask whether a general graph can be the compatibility graph of any staircase relation. However, a definition of the form

'An undirected graph $G = (V, E)$ is a *staircase graph* if there exist a partition $\mathcal{S}$ of V and total orderings on each partition such that E defines a staircase relation.'

is not useful. The reason is that it would result in any simple graph being a staircase graph: Let $\mathcal{S}$ partition V into singletons, i.e. each partition has exactly one vertex. This way, each pair of partitions would trivially satisfy the conditions from Definition 5.3.

We can come up with a more sensible definition by remembering that staircase relations were defined on graphs on which we want to solve a special type of clique problem. Namely, we ask for the optimal (with respect to some objective) clique of size k, where k is the number of partitions. However, the above construction leads to $k = |V|$ and therefore leaves (CPMCS) without a feasible solution unless G is a complete graph. Thus, we may want to restrict the size of the partitioning $\mathcal{S}$ in our definition:

Definition 5.22 (Staircase Graphs)**.** Let $G = (V, E)$ be an undirected graph and let $\omega(G)$ denote the *clique number* of G, i.e. the maximum size of a clique in G. The graph G is called *staircase graph* if there exist a partition $\mathcal{S}$ of V with $|\mathcal{S}| \leq \omega(G)$ and total orderings on each partition such that E defines a staircase relation.

Note that there can be no partition with $|\mathcal{S}| < \omega(G)$, hence Definition 5.22 effectively asks for $|\mathcal{S}| = \omega(G)$.

While technically Theorem 5.19 does not imply that recognizing staircase graphs is NP-hard if the partitioning is not fixed, the alternative would be very surprising. Even if the clique number $\omega(G)$ is known, the task of finding a partitioning into $\omega(G)$ partitions (not considering (SC2)) is already NP-complete.

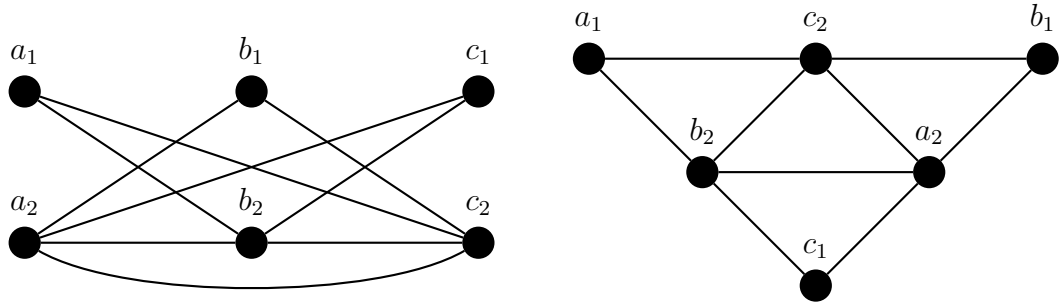
Although staircase graphs and partition-chordal graphs (see Definition 4.10) share the type of application that gave rise to their definition and can be used for proving similar results, there is no inclusion-relation in any direction. Whereas partition-chordal graphs can be sorted into an established hierarchy of well-known subclasses of perfect graphs (see Theorem 5.23), staircase graphs do not fit into this hierarchy: they are neither a subclass of perfect graphs nor a superclass of chordal graphs.

Theorem 5.23.

a) Not every staircase graph is perfect.

b) Not every chordal graph is a staircase graph.

Proof:



(a) Representation from Example 4.6, based on grouping by partitions.

(b) Alternative representation that clearly shows that the graph is chordal.

Figure 5.6: The graph from Example 4.6 is chordal but not a staircase graph.

a) Though we have shown that staircase graphs are a class of graphs that allow solving special clique problems in polynomial time, they are not perfect in general. The abstract reason is that staircase graphs focus on cliques of size k, whereas the definition of perfect graphs requires the clique number to be equal to the chromatic number for every induced subgraph as well. This shows us how to construct a staircase graph that is not perfect: Take the graph that consists of two isolated components, where one of them is a circle of length 5, and the other one a complete graph on 5 vertices. This graph is not perfect as it has an odd hole as an induced subgraph (recall Theorem 4.2 regarding different characterization of perfect graphs). On this graph, we chose 5 partitions, each of which having one vertex from the C_5 component and the K_5 component. All components are ordered such that the vertex from C_5 is larger than the vertex from K_5. Then the resulting relation is a staircase relation.

b) To show that there are chordal graphs that are not staircase graph, we use the graph G from Example 4.6. First of all, this graph is chordal as can be clearly seen from the representation shown in Figure 5.6b. We now show that it is not a staircase-graph. The clique number of the graph is 3 so we look for a partitioning into 3 partitions. Nodes a_2, b_2, c_2 form a clique, and hence have to be in different partitions. Node a_1 is connected to both b_2 and c_2. Thus, it can only be assigned to the partition of Node a_2. In the same way, b_1 and b_2 are in the same partition as well as c_1 and c_2. Hence, the only possible partitioning is that from Example 4.6 (see Figure 5.6a). The reader—by now well trained in spotting violations of (SC2)—will notice that the total orders suggested by Figure 5.6a do not give a staircase relation (e.g. consider (a_1, b_2) and (a_2, b_1)). Moreover, it can also be checked that flipping some of the orderings does not fix this. Hence, G is not a staircase graph.

□

Remark 5.24. From a practical point of view it would be very helpful to identify staircase relations in general MIPs. However, recognizing (near) staircase graphs from

general (unpartitioned) compatibility graphs seems less promising than searching for an ordering if a particular partitioning is meaningful from the application context (see Subsection 5.5.2). In any case, many formulations contain equations of the form

$$\sum_{i \in \mathcal{I}} x_i = 1$$

with binary variables $x_i,\ i \in \mathcal{I}$ for some index set $\mathcal{I}$. Those could be the basis for finding partitions heuristically. Experiments on the quality of staircase relaxations that are obtained heuristically might be an interesting directions for further research.

Chapter 6

Simultaneous Convexification

In Chapter 4, we faced a mixed-integer programming formulation that consists of many components (corresponding to a network arc each) that are already locally ideal, and hence no further improvement is possible when considering only a single network arc. Our strategy was to strengthen the formulation by considering specific substructures consisting of several network arcs, which led to considerable improvements from a theoretical as well as a computational point of view—as we have seen in Sections 4.2 and 4.3, respectively.

In this chapter, we give an outlook on a similar approach that follows the same general idea, and can directly be applied to nonlinear programming formulations. A classic approach of general MINLP solvers consists of decomposing a nonlinear formulation into 'atomic' nonlinear functions and for each such function, constructing a convex hull of its function graph—or a polyhedral approximation thereof. For a recent survey on global optimization for MINLPs, consider [BMF16].

Although computing such a convex hull is very hard in general, the problem is well-studied for typical functions that are considered atomic by solvers such as a product of several variables or commonly-encountered univariate functions. For some classes of functions, even closed-form expressions are known (see e.g. [KS12], [LS13, Chapter 4]) such that the convex relaxation of a single function cannot be improved any further. This is similar to the situation in Chapter 4 with respect to the formulation's linear relaxation.

However, we may strengthen the formulation by considering (partial) convex hulls that involve several nonlinear functions simultaneously, which we call the *simultaneous convex hull* of multiple functions. This set is clearly more restrictive than putting together the separately computed convex hulls of single functions. Hence, we may obtain strong cutting planes. Simultaneous convexification, i.e. computing the convex hull of the function graph of a vector-valued function (in contrast to a real-valued function), has been well-studied for special cases, e.g. the set of quadratic monomials [AB10]. Treatment of examples can also be found in [Bal13, Chapter 5], and more results are referenced in the introduction of [Taw10]. However, results for more general functions are rare according to [Bal13], giving [Taw10] as an example.

In this chapter, we will discuss the potential of aiming for simultaneous convexi-

fication of several functions for optimization problems on passive gas networks. Similar to our approach in Chapter 4, we will consider multiple functions that model a particular substructure of the network, namely a junction (cf. Subsection 4.2.4 for our coverage on junctions in the context of piecewise linearization). This chapter is part of a joint project in progress together with Frauke Liers, Alexander Martin, Nick Mertens, Dennis Michaels and Robert Weismantel.

6.1 The Simultaneous Convex Hull of Functions

We consider nonlinear programs of the following type:

$$\begin{aligned} \min \quad & c(x) \\ \text{s.t.} \quad & f_i(x) \leq 0 \qquad (\ i = 1, \dots, k\ ) \\ & x \in D = [\underline{x}, \overline{x}] \subseteq \mathbb{R}^n_+. \end{aligned} \tag{6.1}$$

In the above formulation, f_i denote continuous nonlinear functions

$$f_i \colon D \to \mathbb{R},$$

over some box domain $D = [\underline{x}, \overline{x}]$ for $i = 1, \dots, k$. Without loss of generality, the objective function c can be assumed to be linear.

In global optimization, a standard approach for solving (mixed-integer) nonlinear optimization problems is based on constructing convex relaxations in conjunction with branch-and-cut. A convex relaxation of (6.1) can be constructed with the help of convex underestimators and concave overestimators of the functions f_i.

Definition 6.1 (Convex Underestimator, Concave Overestimator)**.** A *convex underestimator* of a function f is a convex function $\underline{f} \colon D \to \mathbb{R}$ that bounds the function from below, i.e. $\underline{f}(x) \leq f(x)\ \forall x \in D$. Similarly, a *concave overestimator* of f is a concave function $\overline{f} \colon D \to \mathbb{R}$ with $\overline{f}(x) \geq f(x)\ \forall x \in D$.

The task of computing the best possible (or at least good approximate) estimators is difficult in general. However, for certain function classes they can be computed efficiently, or are known from the literature. Given such estimator functions, we obtain the relaxed formulation

$$\begin{aligned} \min \quad & c(x) \\ \text{s.t.} \quad & \underline{f_i}(x) \leq y_i \leq \overline{f_i}(x) \qquad (\ i = 1, \dots, k\ ) \\ & y_i \leq 0 \qquad (\ i = 1, \dots, k\ ) \\ & x \in D \\ & y \in \mathbb{R}^k, \end{aligned} \tag{6.2}$$

where $\underline{f_i}$ and $\overline{f_i}$ denote convex underestimators and concave overestimators for the constraint functions f_i, respectively, for all $f_i,\ i = 1, \dots, k$. Formulation (6.2) is a convex program and can therefore be solved efficiently.

If $\underline{f_i}$ and $\overline{f_i}$ are best possible (so-called *lower convex envelopes* and *upper concave envelopes*), i.e.

$$\{(x, y_i) \mid \underline{f_i}(x) \leq y_i \leq \overline{f_i}(x),\ x \in D\} = \operatorname{conv}\{(x, f_i(x)) \mid x \in D\}$$

for all $f_i,\ i = 1, \ldots, k$, we call (6.2) the convex relaxation of (6.1) obtained by *separate convexification*.

We can improve this relaxation by considering multiple f_i simultaneously in the shape of a vector-valued function

$$f \colon D \to \mathbb{R}^k,\ x \mapsto (f_1(x), \ldots, f_k(x)).$$

Definition 6.2 (Simultaneous Convex Hull, cf. [Bal13, Definition 5.1])**.**
Let $f_i \colon D \to \mathbb{R},\ i = 1, \ldots, k$ be real-valued functions. Then the set

$$\operatorname{conv}_D[f_1, \ldots, f_k] \coloneqq \operatorname{conv}\{(x, f_1(x), \ldots, f_k(x)) \mid x \in D\} \tag{6.3}$$

is referred to as the *simultaneous convex hull* of the functions f_i over D.

Instead of convexifying the constraint functions f_i separately, we may apply simultaneous convexification, leading to the convex relaxation

$$\begin{aligned} \min \quad & c(x) & & \\ & (x, y) & \in & \operatorname{conv}_D[f_1, \ldots, f_k] \\ & x & \in & D \\ & y & \in & \mathbb{R}^k. \end{aligned} \tag{6.4}$$

Note that simultaneous convexification of all constraint functions gives an exact reformulation of (6.1). Alternatively, it may be applied to any subset $f_i,\ i \in \mathcal{I}$ for some $\mathcal{I} \subseteq \{1, \ldots, k\}$ of the constraint functions, depending on the prospects to compute or approximate $\operatorname{conv}_D[\{f_i \mid i \in \mathcal{I}\}]$. This leads to local reformulations of (6.1) that strengthen the corresponding convex relaxation according to (6.2).

Example 6.3. Formulation (6.4) can be significantly stronger than Formulation (6.2). To illustrate this, consider the functions

$$f_1, f_2 \colon [0, 1] \to \mathbb{R},\ f_1(x) = x^2,\ f_2(x) = (x + 1)^2.$$

Since f_1 and f_2 are both convex, the separate convexification is lower bounded by the functions themselves, whereas the upper concave envelopes are given by secants. Hence, separate convexification gives

$$\{(x, y_1, y_2) \mid x^2 \leq y_1 \leq x,\ (x + 1)^2 \leq y_2 \leq 3x + 1\} \tag{6.5}$$

as the feasible set. However, all points on the function graph of $f \colon x \mapsto (x, f_1(x), f_2(x))$ satisfy

$$2x + f_1(x) - f_2(x) + 1 = 0,$$

and therefore $2x + y_1 - y_2 + 1 = 0$ is a strong valid inequality than can be derived from $\operatorname{conv}_{[0,1]}[f_1, f_2]$, but does not hold for (6.5).

Clever solvers may recognize this linear dependency and only introduce y_1 in the first place. As another (more practical) example, [Bal13, Example 5.5] considers $x \mapsto (x, x^2, x^3)$, the *moment curve* in dimension 3, on the interval $[1, 2]$, and gives a factor of 27 for the volume reduction of the feasible set when moving from separate to simultaneous convexification.

The simultaneous convex hull of functions $f_i,\ i = 1, \dots, k$ for some k can be represented by lower-dimensional objects, namely the separate convex hulls for all linear combinations of those functions.

Theorem 6.4 (cf. [Bal13, Corollary 5.25])**.** *Let $f\colon D \to \mathbb{R}^k,\ x \mapsto (f_1(x), \dots, f_k(x))$ be a continuous vector-valued function, and $D \subseteq R^n$ be a compact convex set. Then*

$$\operatorname{conv}_D[f] = \bigcap_{\alpha \in \mathbb{R}^k} \{(x, y) \in \mathbb{R}^{n+k} \mid (x, \alpha^T y) \in \operatorname{conv}_D[\alpha^T f]\}. \tag{6.6}$$

A proof can be found in [Bal13, Section 5.2.2].

The reformulation in Theorem 6.4 is in between the so-called *inner representation* in terms of extreme points as in (6.3) and a so-called *outer representation* in terms of supporting hyperplanes.

Theorem 6.4 can be very helpful from an algorithmic point of view as it reduces computations for simultaneous convex hulls to convex hulls for scalar-valued functions. Still, a priori, the latter has to be done for any linear combination. This raises the question which weight vectors α are actually needed in (6.6). Some results on this can be found in [Bal13].

Another option consists in using the outer representation

$$\operatorname{conv}_D[f] = \bigcap_{\alpha \in \mathbb{R}^k} \{z \in \mathbb{R}^{n+k} \mid \alpha_{\min} \leq \alpha^T z \leq \alpha_{\max}\}, \tag{6.7}$$

where

$$\begin{aligned} \alpha_{\min} &= \min\{\alpha^T z \mid z \in (x, f(x)), x \in D\} \\ \alpha_{\max} &= \max\{\alpha^T z \mid z \in (x, f(x)), x \in D\}. \end{aligned} \tag{6.8}$$

For any given $\alpha \in \mathbb{R}^k$, we can obtain valid inequalities for (6.4) in terms of the variables x and y as stated in (6.7). However, this approach assumes that we can afford to solve the corresponding optimization problem (6.8) or at least have a way to obtain a good dual bound for it. Choosing suitable values for α is important for an approach based on (6.7) or (6.6). For example, in Example 6.3 the choice $\alpha = (2, 1, -1)$ gives $\alpha_{\min} = \alpha_{\max} = -1$ and therefore reveals the affine linear dependency.

Within the scope of proof-of-concept computations in Section 6.2, we will sample values for α in order to test the potential of those cutting planes in terms of the improvement of the optimal value of relaxation (6.2).

6.2 Application to Gas Network Optimization

Optimization problems on gas networks have already served as an example for a nonlinear network flow problems in several chapters. Subsection 2.1.5 can be consulted for an introductory overview. The nonlinearities given by the pressure loss equation (2.14) are sufficiently nontrivial to provide room for improving solvers, but are still controllable due to their smooth quadratic nature. This makes gas network optimization a promising field of application for simultaneous convexification. Moreover, as already remarked in Chapter 4, network flow problems typically have loosely coupled constraints, leading to well-suited target substructures for applying approaches based on local reformulations.

Consider a single junction in a gas distribution network with a central vertex of degree three together with the pipes adjacent to it. The vertices involved are denoted by v_1 to v_4, where v_3 is the central node with two incoming edges $a_1 = (v_1, v_3)$ and $a_2 = (v_2, v_3)$, and one outgoing edge $a_3 = (v_3, v_4)$, though the formal orientation of pipes does not imply a coinciding mass flow direction in the general case. In line with our naming convention so far, the pressure at vertex v_i is denoted by p_i and the flow value at pipe a_j is denoted by q_j (see Figure 6.1).

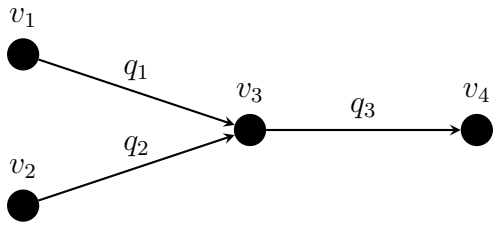


Figure 6.1: Our notation for an example junction of degree three.

As before, we use the formula

$$\lambda_a q_a |q_a| = p_i^2 - p_j^2 \tag{6.9}$$

to describe the pressure loss along a pipe a from v_i to v_j, where $\lambda_a > 0$ is some parameter that depends on the roughness of the pipe. Since our setting represents a passive gas network, pressure variables appear as squares only such that we may use variables $p_i^S := p_i^2$ for the squared pressure in the first place (cf. Subsection 2.1.5).

We aim to describe the convex hull of the feasible region for such a junction, which is given by the simultaneous convex hull

$$\begin{aligned}\operatorname{conv}\{(q_1, q_2, q_3, p_1^S, p_2^S, p_3^S, p_4^S) \mid \\ q_1|q_1| = p_1^S - p_3^S,\ q_2|q_2| = p_2^S - p_3^S,\ q_3|q_3| = p_3^S - p_4^S,\ q_3 = q_1 + q_2\}\end{aligned}$$

Note that all variables are uniquely determined by fixing two flow variables and a single pressure variable, say q_1, q_2, p_3^S. The other quantities can be computed by

$$p_1^S = f_1(q_1, q_2, p_3^S) := p_3^S + q_1|q_1|$$

$$\begin{aligned} p_2^S &= f_2(q_1, q_2, p_3^S) \coloneqq p_3^S + q_2|q_2| \\ p_4^S &= f_3(q_1, q_2, p_3^S) \coloneqq p_3^S - (q_1 + q_2)|q_1 + q_2| \\ q_3 &= f_4(q_1, q_2, p_3^S) \coloneqq q_1 + q_2. \end{aligned}$$

This leads to the set

$$\operatorname{conv}\{(q_1, q_2, p_3^S, y_1, y_2, y_3, y_4) \mid y_i = f_i(q_1, q_2, p_3^S),\ i = 1, \dots, 4\} = \operatorname{conv}_D[f_1, \dots, f_4],$$

where $D = [\underline{q_1}, \overline{q_1}] \times [\underline{q_2}, \overline{q_2}] \times [\underline{p_3^S}, \overline{p_3^S}]$ denotes the box domain of the variables q_1, q_2, p_3^S.

Furthermore, as sums can be convexified independently, we have

$$\begin{aligned} & (q_1, q_2, p_3^S, y_1, y_2, y_3, y_4) \in \operatorname{conv}_D[f_1, \dots, f_4] \\ \Leftrightarrow\quad & (q_1, q_2, p_3^S, y_1 - p_3^S, y_2 - p_3^S, y_3 - p_3^S) \in \operatorname{conv}_D[g(q_1), g(q_2), g(q_1 + q_2)] \\ & \wedge\, y_4 = q_1 + q_2, \end{aligned}$$

where $g(x) \coloneqq x|x|$. Therefore, the task of determining a simultaneous convexification for a junction of three nodes reduces to studying the object

$$\operatorname{conv}_D[g(q_1), g(q_2), g(q_1 + q_2)]. \tag{6.10}$$

Let us now assume fixed flow directions on all arcs according to their formal orientation. In this case, $g(x) \coloneqq x|x| = x^2$, which leads us to the well-studied realm of quadratic programming.

Well-known relaxations for this class of problems include the *relaxation-linearization technique (RLT)* [SA13] and the *positive semidefinite (PSD) relaxation* (see e.g. [VB96], also cf. [QBM12] concerning both techniques).

RLT Relaxation Let x_i and x_j be two variables from a quadratic program with domains $x_i \in [\underline{x_i}, \overline{x_i}]$ and $x_j \in [\underline{x_j}, \overline{x_j}]$, respectively. We introduce auxiliary variables X_{ij} to model the product $x_i x_j$. Then, the following *RLT constraints* (introduced in [McC76], and thus also known as *McCormick inequalities*) are valid inequalities:

$$\begin{aligned} X_{ij} - \underline{x_j} x_i - \underline{x_i} x_j &\geq -\underline{x_i}\,\underline{x_j} \\ X_{ij} - \overline{x_j} x_i - \overline{x_i} x_j &\geq -\overline{x_i}\,\overline{x_j} \\ X_{ij} - \overline{x_j} x_i - \underline{x_i} x_j &\leq -\underline{x_i}\,\overline{x_j} \\ X_{ij} - \underline{x_j} x_i - \overline{x_i} x_j &\leq -\overline{x_i}\,\underline{x_j}. \end{aligned} \tag{6.11}$$

Moreover, if x_i and x_j are independent variables, the convex hull of a single product $\operatorname{conv}_D[x_i x_j]$ is completely described by the RLT relaxation [AKF83]. [McC76]

PSD Relaxation Let x be a (column) vector of variables of a quadratic program. Again, we use auxiliary variables X_{ij} for the products $x_i x_j$. We relax the nonconvex matrix inequality $X - xx^T = 0$ to

$$X - xx^T \succcurlyeq 0.$$

By *Schur's complement*, this is equivalent to

$$\begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \succcurlyeq 0 \tag{6.12}$$

Note that (6.12) is a convex constraint, and an efficient separation of supporting hyperplanes can be done via eigenvector computations.

It should be remarked that the resulting PSD relaxation represents the first level of the *Lasserre hierarchy*, where the matrix in (6.12) is referred to as the *moment matrix* in this context. Relaxations from this hierarchy have been very successful in global polynomial optimization as well as for constructing approximation algorithms, see e.g. [Las01] and [CT12], respectively. More details on the relation to other relaxations can be found in the survey [Lau03]. In our special case, the first level of the hierarchy is sufficient, as we have the following result due to [AB10]:

Theorem 6.5 ([AB10, Theorem 2])**.** *Let $x = (x_1, x_2) \in D \subseteq \mathbb{R}^2$ and D be a box domain. Then*

$$\left\{ \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \mid (6.11), (6.12) \right\} = \operatorname{conv} \left\{ \begin{pmatrix} 1 \\ x \end{pmatrix} \begin{pmatrix} 1 \\ x \end{pmatrix}^T \mid x \in D \right\}. \tag{6.13}$$

This implies that PSD constraints (6.12) together with the RLT constraints (6.11) give a complete description of $\operatorname{conv}_D[g(q_1), g(q_2), g(q_1 + q_2)]$ if $x = (q_1, q_2)$ and the domain D is a box, though the result does not hold for higher dimensions as shown in [AB10], i.e. for junctions of degree four or higher in our context. However, using a semidefinite programming (SDP) solver within a branch-and-bound algorithm creates practical difficulties. In [QBM12], the authors mention a lack of efficient warm starting mechanisms as a major drawback of current SDP solvers, and therefore propose LP relaxations of PSD constraints.

In the next section, we will also consider convexification of squares of single variables for comparison. For $x_i \in \mathbb{R}$, the set $\operatorname{conv}_D[x_i^2]$ is given by a secant line as a concave overestimator and the function $x_i \mapsto x_i^2$ itself as a convex underestimator. The former is straightforward to compute, e.g. by using (6.11) for $x_j = x_i$. The latter constraint is nonlinear and can be reformulated as the PSD constraint

$$\begin{pmatrix} 1 & x_i^T \\ x_i & X_{ii} \end{pmatrix} \succcurlyeq 0, \tag{6.14}$$

where X is the auxiliary variable for x^2. Indeed, the determinant of the matrix in (6.14) is equal to $X_{ii} - x_i^2$, implying $X_{ii} \geq x_i^2$. We will call (6.14) *separate PSD contraints*—in contrast to the simultaneous PSD contraints (6.12). Note that (6.12) implies (6.14) for all variables involved, as every principal submatrix of a positive semidefinite matrix is also positive semidefinite.

If not all flow directions are fixed for a junction, computing the separate convex hull $\operatorname{conv}[q|q|]$ is still an easy exercise but PSD relaxations are not valid anymore and would require the introduction of binary variables to resolve the absolute value in (6.9).

6.3 Computational Experiments on the Potential of Simultaneous Convexification

When moving from separate to simultaneous convex relaxation, significant volume reductions of the feasible set have been demonstrated in [Bal13]. However, there is the question whether this translates into an improvement of the quality of the aforementioned relaxations in terms of their optimal value in practice.

In this section, we will show some proof-of-concept computations on this issue as a basis for the discussion. We will consider example instances on a small passive gas network and compare different relaxations. Our test network consists of 7 nodes and 9 arcs. Its topology is shown in Figure 6.2. Nodes v_1, v_2, v_3 will be sources whereas v_7 is the network's solitary sink. Demands and pressure values at the sources and sinks are not fixed (otherwise the feasible set is at most a single point, cf. [RMWSB02]), but are restricted to lie in an interval range. For reasonably tight bounds we apply the preprocessing implemented in the *Lamatto++* software framework for gas network optimization, described in [Gei11b, Chapter 7]. Simultaneous convexification techniques will be applied to the three middle junctions at nodes v_4, v_5 and v_6.

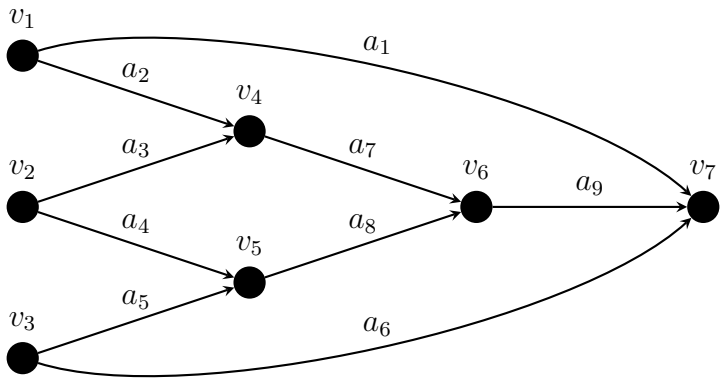


Figure 6.2: Test network for the computations in this chapter consisting of 7 nodes and 9 arcs.

First, we consider all flow directions to be fixed according to the orientations shown in Figure 6.2. This allows us to locally apply the exact convex hull via semidefinite programming due to Theorem (6.13) as discussed in the previous section. We compare it to standard approaches that use separate convexification, either by just applying the RLT relaxation (see (6.11)), or the RLT relaxation together with the convex hull of all squares of single flow variables $q_i \mapsto q_i^2$. The latter is enforced by the PSD constraint (6.14). Computations are performed using YalMIP [Lö04] together with the SDP solver SeDuMi [Stu99].

Moreover, we compute global optima using the MINLP solver BARON [TS05]. In Table 6.1, the success of a relaxation is measured with respect to the amount of gap clo-

sed between the optimal solution and BARON's dual bound at the root node. BARON has access to a number of range reduction techniques, details and results of which are not transparent. To ensure that all approaches start with the same quality of variable bounds, we shut off all range reduction options, which includes the parameters *TDo, MDo, LBTTDo, OBTTDo, PDo* (see [Sah14] for more details). After that, the root dual bound of BARON (version 16.5.16) essentially agrees with the RLT relaxation for all scenarios. Moreover, BARON is prevented from doing local searches during preprocessing (controllable via the parameters *NumLoc* and *DoLocal*).

scenario	gap closed [%]	
	sep. conv.	sim. conv.
1	0.00	0.00
2	–	–
3	13.90	26.78
4	87.51	99.79
5	39.58	49.49
6	0.00	0.00
7	–	–
8	42.09	43.37
9	31.36	35.89
10	63.77	91.93

Table 6.1: Percentage of gap closed between optimal solution and root relaxation due to separate and simultaneous convexification for scenarios on a small test network.

Table 6.1 shows results on the network from Figure 6.2 for 10 selected scenarios using different types of objective function including minimizing or maximizing the demand of a specific node (Scenarios 1,2,5,6), the pressure at a specific node (3,4), the pressure difference between nodes (9,10) and linear combinations of flow variables (7,8). This experiment gives an idea on the improvement of the root relaxation that can be achieved by simultaneous convexifications applied to 3-junctions.

We see that for the majority of scenarios, a clear improvement can already be achieved when using an exact separate convexification, i.e. ensuring Constraints (6.14). In those cases, the simultaneous convexification further improves the relaxation, in two cases even closing more than 90% of the gap. Hence, there is a lot to be gained from convexifiying junctions, though it should be noted that for some examples the tighter relaxation does not pay off for the specific objective function: Scenarios 2 and 7 were trivial in the sense that the root relaxation already gave the optimal bound. In Scenarios 1 and 6 neither separate nor simultaneous convexification were able to close any amount of the remaining gap.

For a second test, we do not assume fixed flow directions on all arcs anymore. Demand intervals for sources and sinks are adjusted such that arcs a_3 and a_5 have a

wide range of possible flow values in both directions (roughly centered around 0) after preprocessing. We test an approach based on the reformulation of the simultaneous convex hull in (6.7). The target structures for application again are the central junctions such that we have to (approximately) convexify 5-dimensional objects of type (6.10). This is done via sampling uniformly distributed values for α on the sphere $S^4 \subseteq \mathbb{R}^5$. After that, a scaling is applied to compensate for the different orders of magnitude for flow and pressure drop terms. For each alpha, we solve (6.8), obtaining cutting planes that can be added to the formulation that is handed to BARON. Again, we measure the percentage of gap reduction between the value of the root relaxation and a global optimum.

scenario	gap closed [%]	
	100 α samples	1000 α samples
1	–	–
2	–	–
3	19.90	39.33
4	19.75	49.36
5	0.00	0.00
6	–	–
7	–	–
8	19.31	48.49
9	14.53	33.12
10	33.65	59.76

Table 6.2: Percentage of gap closed between optimal solution and root relaxation due to applying a sampling approach based on the outer description, using 100 and 1000 weight vectors, respectively, for scenarios on a small test network.

Table 6.2 shows the results in terms of relaxation quality for 100 and 1000 weight vectors α. Note that the smaller amount of samples is a strict subset of the larger one such that it leads to a relaxation of the formulation for the latter. Scenarios 1-10 use the same objective functions as in Table 6.1, though note that the feasible sets differ due to not fixing all flow directions and adjustments of the demand ranges. In this case, 4 instances happen to be trivial, whereas the root relaxation for Scenario 5 could not be improved whatsoever. For the remaining 5 instances, using 1000 samples led to a significant improvement of 30-60%. This level could not be reached by the formulation based on just 100 samples, though the observed reduction (about half the amount compared to 1000 samples) is still notable.

However, it should not be withheld that the large amount of cutting planes severely slows down the solver such that the optimum is computed fastest by far in the version without any cutting planes. Still, a clear reduction in branch-and-bound nodes needed by the solver has been observed, especially for the version with 1000 samples for α.

6.4 Further Remarks and Outlook

The experiments in the previous section have shown that simultaneous convexification has the potential for significant improvement of the standard relaxation employed by a leading MINLP solver. However, further algorithmic progress is needed to unlock this potential.

One possibility consists of incorporating SDP-constraints. However, as mentioned in Section 6.2, SDP solvers are not ready to be used together with efficient branch-and-cut algorithms without further ado. It seems more promising to construct linear relaxations of PSD constraints as proposed in [QBM12], though this still only provides the simultaneous convex hull for fixed flow directions.

A major drawback of computations based on the outer description (6.7) is that we have to solve (or bound) many instances of the nonlinear optimization problem (6.8). Though each problem is low-dimensional, in total they take a huge amount of time, while many values for α will lead to redundant information. In order to make this approach competitive, one needs to save a lot of time with respect to those computation, for instance by using prior knowledge on the most relevant weight vectors, by computing good bounds for several subproblems (6.8) at once, or by developing separation routines. The latter seems especially promising for further research in conjunction with the object (6.10), possibly assuming that some of the flow directions are known, which is not uncommon in practice after preprocessing. As the simultaneous convex hull heavily depends on the variable domains, updating possible cutting planes may be important as bounds are tightened during the solution process. Such an implementation is far from trivial apart from the fact that many MINLP solvers—like BARON—do not allow this kind of intrusion at all.

Finally, note that the approaches presented and discussed in Sections 6.2 and Sections 6.3 are in principle applicable to junctions of arbitrary degree. Moreover, similar to the piecewise linear setting, they can be applied to aggregate nodes (cf. Section 4.4).

Chapter 7

Conclusions and Outlook

In this thesis, we have seen several methods for solving different network optimization problems based on local adaptations of the formulation. At best, the presentation made the reader want to apply the findings of this thesis to his or her own network optimization problem that might not have been covered explicitly. Therefore, besides giving a summary, I want to highlight some observations that might be valuable in that case.

In Chapter 3, we examined a new algorithmic framework for the solution of network design problems which is based on iterative graph aggregation. An aggregated version of the network graph is iteratively refined until it represents the whole graph sufficiently well in the sense that an optimal solution to the aggregated problem can easily be extended to an optimal solution to the original problem. Computational experiments clearly show that for the single-commodity network expansion problem, especially for instances with relatively high preinstalled capacities, we could indeed achieve a significant reduction in graph size as well as solution time when compared to directly solving the problem with an MIP-solver.

When considering to employ an aggregation scheme as presented in this work, one should bear in mind the following:

- This approach is designed for network optimization problems where the main challenge can be attributed to the relatively large size of the network. As prerequisites for applying aggregation, it should be feasible to solve the problem on a reduced graph several times (the aggregated master problem). Also, it is very beneficial if a less complex problem can still be solved on the whole graph in order to allow for a global subproblem.

- An implicit assumption of the scheme is that the high detail of the topology is somewhat unnecessary in some areas of the graph. Therefore, one should have observed optimal solutions that are reasonably sparse prior to using aggregation (as is the case in the motivating example from railway networks, cf. Section 3.1).

- The possibility to integrate aggregation into the solver's branch-and-bound-tree allowed to design significantly more successful algorithms.

- The algorithm in principle is highly extendible to other network design problems. However, if the relaxation given by the master problem is too weak, aggregation is not competitive in its basic version and a combination with additional algorithmic ideas should be explored (see Sections 3.6 and 3.7).

In Chapter 4, we studied a setting of linearized network flows that covers the situation, in which nonlinearities of a problem defined on a transportation network are modeled as a piecewise linear function of the flow. We derived several structural results, most importantly for the case of paths of arbitrary length. For those substructures we gave a complete description of the polyhedron defined by the feasible binary decisions. The proof uses results on perfect graphs, and we introduced a class of perfect graphs that is new to our knowledge. Our computational experiments showed that Gurobi, a state-of-the-art commercial MIP-solver, drastically benefits from using our implementations of cutting plane separation.

Please note the following observations and directions for further research:

- Piecewise linearization of network flows leads to a rich and interesting structure that can successfully be exploited by cutting-plane methods. Further research in this direction seems very relevant as piecewise linear approximation and relaxation are frequently-used approaches.

- The cutting planes developed in Section 4.2 are useful for a number of methods for piecewise linearization, including the Multiple Choice Method, the Convex Combination Method and the Incremental Method.

- As remarked in Section 4.4, the cutting planes can be applied to aggregate structures. It would be interesting to find a good selection rule for target structures, possibly in combination with approaches from Chapter 3.

- Applying results from graph theory to compatibility graphs did allow for new insights. In particular, the theory of perfect graphs seems especially interesting, as it characterizes graphs on which a certain complete description is available.

In Chapter 5, we became acquainted with the notion of staircase compatibility, which generalizes compatibility structures known from different areas of application, such as project scheduling and piecewise linearization. We showed that the convex hull of feasible solutions of the clique problem with multiple-choice constraints can be described by a totally unimodular constraint matrix of polynomial size if the compatibility graph is given by a staircase relation. Furthermore, the constraint matrix is cographic, which yields a dual-flow formulation for the problem. For two example applications, we observed that using our reformulations represents a huge improvement over a naive standard formulation.

The following are some key points to be highlighted:

- The definition of staircase compatibility gives a unified view on multiple applications that essentially share the same underlying polytope. Furthermore, it represents a strict generalization.

- Clique problems with multiple-choice constraints have the property that it is sufficient to consider pairwise conflicts for the binary variables to fully describe the problem. However, the results from Chapter 5 will still give valid cutting planes otherwise (via a *staircase relaxation*), though they will likely be less impactful.

- For clique problems with multiple-choice constraints under staircase compatibility, there is a huge performance difference between the tested formulations, even between formulations that are both totally unimodular. There is an interesting link to linearization methods, where indeed similar results can be observed.

- Although recognition of staircase compatibility might theoretically be hard in general as soon as the order on each partition is not given, solving the recognition problem via the MIP-formulation given in Subsection 5.5.2 is realistic. This allows us to compute staircase relaxations that may yield strong reformulations if a sensible partitioning is available from the application context.

In Chapter 6, we considered the approach of strengthening the convex relaxation of a nonlinear problem by investigation of the convex hull of a part of the formulation associated with a specific network substructure. This can be seen as a continuous counterpart to our strategy in Chapter 4. We may record the following outlook:

- Computing the simultaneous convex hull can give a very strong convex relaxation when compared to a solver's root relaxation and also the standard approach of separately convexifiying all constraint functions, even for substructures as small as 3-junctions.

- Further research is needed to make an approach based on simultaneous convexification for gas networks competitive with respect to runtime. In particular, having (heuristic) separation routines might give a large boost to the approach.

- Gas network optimization problems seem to be well suited for aiming at local simultaneous convexifications, as nonlinearities are sufficiently nontrivial to give enough room for improving solvers but also accessible for theoretical analysis (e.g. due to the connection to semidefinite programming, see Section 6.2). Moreover, the loosely coupled constraints due to the network structure allow for natural target substructures.

Altogether, this thesis aimed at devising methods that help to cope with the raising challenges posed by modern network optimization applications. Hopefully, by now the reader is convinced of the potential of our methods even beyond the specific problems where computations have been provided. In any case, the investigations have opened up a lot of interesting topics for further research.

Bibliography

[AB02] Réka Albert and Albert-László Barabási, *Statistical mechanics of complex networks*, Reviews of Modern Physics **74** (2002), 47–97.

[AB10] Kurt M. Anstreicher and Samuel Burer, *Computable representations for convex hulls of low-dimensional quadratic forms*, Mathematical Programming **124** (2010), no. 1, 33–43.

[AKF83] Faiz A Al-Khayyal and James E Falk, *Jointly constrained biconvex programming*, Mathematics of Operations Research **8** (1983), no. 2, 273–286.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin, *Network flows: Theory, algorithms, and applications*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[Bä16] Andreas Bärmann, *Solving network design problems via decomposition, aggregation and approximation*, Ph.D. thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2016.

[Bal65] Egon Balas, *Solution of large-scale transportation problems through aggregation*, Operations Research **13** (1965), no. 1, 82–93.

[Bal13] Martin Ballerstein, *Convex relaxations for mixed-integer nonlinear programs*, Ph.D. thesis, Eidgenössische Technische Hochschule ETH Zürich, 2013.

[Bar96] Francisco Barahona, *Network design using cut inequalities*, SIAM Journal on Optimization **6** (1996), no. 3, 823–837.

[BCD+08] Pierre Bonami, Gérard Cornuéjols, Sanjeeb Dash, Matteo Fischetti, and Andrea Lodi, *Projected Chvátal–Gomory cuts for mixed integer linear programs*, Mathematical Programming **113** (2008), no. 2, 241–257.

[Ben62] Jacques F. Benders, *Partitioning procedures for solving mixed-variables programming problems*, Numerische Mathematik **4** (1962), no. 1, 238–252.

[BGMS16] Andreas Bärmann, Thorsten Gellermann, Maximilian Merkert, and Oskar Schneider, *Staircase compatibility and its applications in scheduling and piecewise linearization*, Tech. report, FAU Erlangen-Nürnberg, 2016.

[BHJS94] Cynthia Barnhart, Christopher A. Hane, Ellis L. Johnson, and Gabriele Sigismondi, *A column generation and partitioning approach for multi-commodity flow problems*, Telecommunication Systems **3** (1994), no. 3, 239–258.

[BLM+15] Andreas Bärmann, Frauke Liers, Alexander Martin, Maximilian Merkert, Christoph Thurner, and Dieter Weninger, *Solving network design problems via iterative aggregation*, Mathematical Programming Computations **7** (2015), no. 2, 189–217.

[BMF16] Fani Boukouvala, Ruth Misener, and Christodoulos A. Floudas, *Global optimization advances in mixed-integer nonlinear programming, minlp, and constrained derivative-free optimization, cdfo*, European Journal of Operational Research **252** (2016), no. 3, 701 – 727.

[BMMN95] Michael O. Ball, Thomas L. Magnanti, Clyde L. Monma, and George L. Nemhauser (eds.), *Network models*, Elsevier Science, 1995.

[BMS17] Andreas Bärmann, Alexander Martin, and Oskar Schneider, *A comparison of performance metrics for balancing the power consumption of trains in a railway network by slight timetable adaptation*, Public Transport (2017), 95–113.

[BT70] Evelyn Martin Lansdowne Beale and John A Tomlin, *Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables*, OR **69** (1970), no. 447-454, 99.

[CCZ14] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli, *Integer programming*, vol. 271, Springer, 2014.

[CH77] Václav Chvátal and Peter L. Hammer, *Aggregation of inequalities in integer programming*, Studies in Integer Programming (P.L. Hammer, E.L. Johnson, B.H. Korte, and G.L. Nemhauser, eds.), Annals of Discrete Mathematics, vol. 1, Elsevier, 1977, pp. 145–162.

[Chv73] Václav Chvátal, *Edmonds polytopes and a hierarchy of combinatorial problems*, Discrete Mathematics **4** (1973), no. 4, 305 – 337.

[CJL+16] Valentina Cacchiani, Michael Jünger, Frauke Liers, Andrea Lodi, and Daniel R. Schmidt, *Single-commodity robust network design with finite and hose demand sets*, Mathematical Programming **157** (2016), no. 1, 297–342.

[CML15] Josu Ceberio, Alexander Mendiburu, and Jose A. Lozano, *The linear ordering problem revisited*, European Journal of Operational Research **241** (2015), no. 3, 686 – 696.

[Cos05] Alysson M. Costa, *A survey on benders decomposition applied to fixed-charge network design problems*, Computers and Operations Research **32** (2005), no. 6, 1429–1450.

[CPSM14] Carlos M. Correa-Posada and Pedro Sánchez-Martín, *Gas network optimization: A comparison of piecewise linear models*, Optimization Online, 2014.

[CRST06] Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas, *The strong perfect graph theorem*, Annals of Mathematics **164** (2006), no. 1, 51–229.

[CT12] Eden Chlamtac and Madhur Tulsiani, *Convex relaxations and integrality gaps*, pp. 139–169, Springer US, Boston, MA, 2012.

[Dan60] George B. Dantzig, *On the significance of solving linear programming problems with some integer variables*, Econometrica **28** (1960), no. 1, 30–44.

[DGJ06] Camil Demetrescu, Andrew Goldberg, and David Johnson, *9th dimacs implementation challenge – shortest paths*, `http://www.dis.uniroma1.it/~challenge9/`, 2006.

[DM02] Elizabeth Dolan and Jorge J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical Programming A **91** (2002), no. 2, 201 – 213.

[DRV87] Lev Michailovich Dudkin, Ilya Rabinovich, and Ilya Vakhutinsky, *Iterative aggregation theory*, Pure and applied mathematics, no. 111, Dekker, New York [u.a.], 1987.

[Fei93] Miloslav Feistauer, *Mathematical methods in fluid dynamics*, Pitman Monographs and Surveys in Pure and Applied Mathematics Series 67, Longman Scientific & Technical, 1993.

[FM05] Armin Fügenschuh and Alexander Martin, *Computational integer programming and cutting planes*, Handbooks in Operations Research and Management Science **12** (2005), 69–121.

[Fou84] Robert Fourer, *Staircase matrices and systems*, SIAM Review **26** (1984), no. 1, 1–70.

[Fra85] Vernon Edward Francis, *Aggregation of network flow problems*, Ph.D. thesis, University of California, 1985.

[FSZ10] Matteo Fischetti, Domenico Salvagnin, and Arrigo Zanette, *A note on the selection of Benders' cuts*, Mathematical Programming, Series B **124** (2010), 175–182.

[Gei11a] Robert Geisberger, *Advanced route planning in transportation networks*, Ph.D. thesis, Karlsruhe Institute of Technology, 2011.

[Gei11b] Björn Geißler, *Towards globally optimal solutions for minlps by discretization techniques with applications in gas network optimization*, Ph.D. thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2011.

[Geo74] Arthur M. Geoffrion, *Lagrangean relaxation for integer programming*, Approaches to Integer Programming (Michel L. Balinski, ed.), Mathematical Programming Studies, vol. 2, Springer Berlin Heidelberg, 1974, pp. 82–114 (English).

[GH62] Alain Ghouila-Houri, *Caracterisation des matrices totalement unimodulaires*, Comptes Rendus Hebdomadaires des Séances de 1'Académie des Sciences (Paris) **254** (1962), no. 1, 1192–1194.

[GJ79] Michael R. Garey and David S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W.H. Freeman and Company, New York, 1979.

[GJS74] Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer, *Some simplified np-complete problems*, Proceedings of the Sixth Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '74, ACM, 1974, pp. 47–63.

[GLS88] Martin Grötschel, László Lovász, and Alexander Schrijver, *Geometric algorithms and combinatorial optimization*, Springer, 1988.

[GMMS12] Björn Geißler, Alexander Martin, Antonio Morsi, and Lars Schewe, *Using piecewise linear functions for solving MINLPs*, Mixed Integer Nonlinear Programming (Jon Lee and Sven Leyffer, eds.), The IMA Volumes in Mathematics and its Applications, vol. 154, Springer New York, 2012, pp. 287–314.

[Gol80] Martin C. Golumbic, *Algorithmic graph theory and perfect graphs*, Academic Press, 1980.

[GT88] Andrew V. Goldberg and Robert E. Tarjan, *A new approach to the maximum-flow problem*, J. ACM **35** (1988), no. 4, 921–940.

[GT89] ———, *Finding minimum-cost circulations by canceling negative cycles*, J. ACM **36** (1989), no. 4, 873–886.

[Gur17] Gurobi Optimization, Inc., *Gurobi optimizer reference manual*, `http://www.gurobi.com`, 2017.

[Hel23] Eduard Helly, *Über Mengen konvexer Körper mit gemeinschaftlichen Punkten*, Jahresbericht der Deutschen Mathematiker-Vereinigung **32** (1923), 175–176.

[HO03] John N. Hooker and Greger Ottosson, *Logic-based Benders decomposition*, Mathematical Programming **96** (2003), no. 1, 33–60.

[HS90] Asa Hallefjord and Sverre Storoy, *Aggregation and disaggregation in integer programming problems*, Operations Research **38** (1990), no. 4, 619–623.

[Iji71] Yuji Ijiri, *Fundamental queries in aggregation theory*, Journal of the American Statistical Association **66** (1971), no. 336, 766–782.

[JL84] Robert G. Jeroslow and J.K. Lowe, *Modelling with integer variables*, Mathematical Programming at Oberwolfach II (Bernhard Korte and Klaus Ritter, eds.), Mathematical Programming Studies, vol. 22, Springer Berlin Heidelberg, 1984, pp. 167–184.

[JLK78] David S. Johnson, Jan Karel Lenstra, and A. H. G. Rinnooy Kan, *The complexity of the network design problem.*, Networks **8** (1978), no. 4, 279–285.

[Kar72] Richard M. Karp, *Reducibility among combinatorial problems*, Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York (Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, eds.), Springer US, Boston, MA, 1972, pp. 85–103.

[KdFN06] Ahmet B. Keha, Ismael R. de Farias, and George L. Nemhauser, *A branch-and-cut algorithm without binary variables for nonconvex piecewise linear optimization*, Operations Research **54** (2006), no. 5, 847–858.

[KHPS15] Thorsten Koch, Benjamin Hiller, Marc Pfetsch, and Lars Schewe (eds.), *Evaluating gas network capacities*, MOS-SIAM Series on Optimization, 2015.

[KR79] Mark H. Karwan and Ronald L. Rardin, *Some relationships between lagrangian and surrogate duality in integer programming*, Mathematical Programming **17** (1979), no. 1, 320–334 (English).

[KS12] Aida Khajavirad and Nikolaos V Sahinidis, *Convex envelopes of products of convex and component-wise concave functions*, Journal of global optimization **52** (2012), no. 3, 391–409.

[KV07] Bernhard Korte and Jens Vygen, *Combinatorial optimization: Theory and algorithms*, 4th ed., Springer Publishing Company, Incorporated, 2007.

[KW84] R. John Kaye and Felix F. Wu, *Analysis of linearized decoupled power flow approximations for steady-state security assessment*, IEEE Transactions on Circuits and Systems **31** (1984), no. 7, 623–636.

[Lö04] Johan Löfberg, *Yalmip : a toolbox for modeling and optimization in matlab*, 2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508), Sept 2004, pp. 284–289.

[Las01] Jean B. Lasserre, *Global optimization with polynomials and the problem of moments*, SIAM Journal on Optimization **11** (2001), no. 3, 796–817.

[Lau03] Monique Laurent, *A comparison of the Sherali-Adams, Lovász-Schrijver, and Lasserre relaxations for 0–1 programming*, Mathematics of Operations Research **28** (2003), no. 3, 470–496.

[Lei95] Rainer Leisten, *Iterative Aggregation und mehrstufige Entscheidungsmodelle: Einordnung in den planerischen Kontext, Analyse anhand der Modelle der linearen Programmierung und Darstellung am Anwendungsbeispiel der hierarchischen Produktionsplanung*, Produktion und Logistik, Physica-Verlag, 1995.

[Lei98] ———, *An LP-aggregation view on aggregation in multi-level production planning*, Annals of Operations Research **Bd. 82** (1998), S. 413–434.

[Lem01] Claude Lemaréchal, *Lagrangian relaxation*, Computational Combinatorial Optimization (Michael Jünger and Denis Naddef, eds.), Lecture Notes in Computer Science, vol. 2241, Springer Berlin Heidelberg, 2001, pp. 112–156 (English).

[LM16] Frauke Liers and Maximilian Merkert, *Structural investigation of piecewise linearized network flow problems*, SIAM Journal on Optimization **26** (2016), no. 4, 2863–2886.

[LMT09] Jeff Linderoth, François Margot, and Greg Thain, *Improving bounds on the football pool problem by integer programming and high-throughput computing*, INFORMS Journal on Computing **21** (2009), no. 3, 445–457.

[LS13] Marco Locatelli and Fabio Schoen, *Global optimization: theory, algorithms, and applications*, SIAM, 2013.

[LT03] Igor Litvinchev and Vladimir Tsurkov, *Aggregation in large-scale optimization*, Applied Optimization, Springer, 2003.

[MAdC+11] Rita Macedo, Cláudio Alves, José M. Valério de Carvalho, François Clautiaux, and Saïd Hanafi, *Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model*, European Journal of Operational Research **214** (2011), no. 3, 536 – 545.

[McC76] Garth P. McCormick, *Computability of global solutions to factorable nonconvex programs: Part i — convex underestimating problems*, Mathematical Programming **10** (1976), no. 1, 147–175.

[MD77] Dale McDaniel and Mike Devine, *A modified Benders' partitioning algorithm for mixed integer programming*, Management Science **24** (1977), no. 3, 312–319.

[MM57] Harry M. Markowitz and Alan S. Manne, *On the solution of discrete programming problems*, Econometrica **25** (1957), no. 1, pp. 84–110.

[MR11] Rafael Martí and Gerhard Reinelt, *The linear ordering problem: exact and heuristic methods in combinatorial optimization*, vol. 175, Springer Science & Business Media, 2011.

[MSSU01] Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz, *On project scheduling with irregular starting time costs*, Operations Research Letters **28** (2001), no. 4, 149–154.

[NK07] Alexandra M. Newman and Mark Kuchta, *Using aggregation to optimize long-term production planning at an underground mine*, European Journal of Operational Research **176** (2007), no. 2, 1205 – 1218.

[Opa79] Jaroslav Opatrny, *Total ordering problem*, SIAM Journal on Computing **8** (1979), no. 1, 111–114.

[OPTW07] Sebastian Orlowski, Michal Pióro, Artur Tomaszewski, and Roland Wessäly, *SNDlib 1.0–Survivable Network Design Library*, Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium, 2007.

[Oxf] *Oxford dictionaries*, https://en.oxforddictionaries.com, Accessed: 2017-06-10.

[Oxl06] James G. Oxley, *Matroid theory (oxford graduate texts in mathematics)*, Oxford University Press, Inc., New York, NY, USA, 2006.

[Pac16] Jörn Pachl, *Systemtechnik des Schienenverkehrs: Bahnbetrieb planen, steuern und sichern*, Springer Vieweg, 2016.

[Pad73] Manfred W. Padberg, *On the facial structure of set packing polyhedra*, Mathematical Programming **5** (1973), 199–215.

[Pad00] ———, *Approximating separable nonlinear functions via mixed zero-one programs*, Oper. Res. Lett. **27** (2000), no. 1, 1–5.

[PFG+15] Marc E. Pfetsch, Armin Fügenschuh, Björn Geißler, Nina Geißler, Ralf Gollmer, Benjamin Hiller, Jesco Humpola, Thorsten Koch, Thomas Lehmann, Alexander Martin, Antonio Morsi, Jessica Rövekamp, Lars Schewe, Martin Schmidt, Rüdiger Schultz, Robert Schwarz, Jonas Schweiger, Claudia Stangl, Marc C. Steinbach, Stefan Vigerske, and Bernhard M. Willert, *Validation of nominations in gas network optimization: Models, methods, and solutions*, Optimization Methods and Software **30** (2015), no. 1, 15–53.

[QBM12] Andrea Qualizza, Pietro Belotti, and François Margot, *Linear programming relaxations of quadratically constrained quadratic programs*, Mixed Integer Nonlinear Programming, Springer, 2012, pp. 407–426.

[RMWSB02] Roger Z. Rios-Mercado, Suming Wu, L. Ridgway Scott, and E. Andrew Boyd, *A reduction technique for natural gas transmission network optimization problems*, Annals of Operations Research **117** (2002), no. 1-4, 217–234.

[Ros74] Ivo G. Rosenberg, *Aggregation of equations in integer programming*, Discrete Mathematics **10** (1974), no. 2, 325 – 341.

[RPWE91] David F. Rogers, Robert D. Plante, Richard T. Wong, and James R. Evans, *Aggregation and disaggregation techniques and methodology in optimization.*, Operations Research **39** (1991), no. 4, 553.

[SA13] Hanif D Sherali and Warren P Adams, *A reformulation-linearization technique for solving discrete and continuous nonconvex problems*, vol. 31, Springer Science & Business Media, 2013.

[Sah14] Nikolaos V. Sahinidis, *BARON 16.5.16: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual, 2014.

[Sch86] Alexander Schrijver, *Theory of linear and integer programming*, John Wiley & Sons, Inc., New York, NY, USA, 1986.

[Sch15] Jonas Scholz, *Aggregation zur Lösung von Topologieplanungsproblemen auf Gasnetzwerken*, Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2015.

[Sey80] Paul D. Seymour, *Decomposition of regular matroids*, Journal of Combinatorial Theory, Series B **28** (1980), no. 3, 305 – 359.

[SLL13] Srikrishna Sridhar, Jeff Linderoth, and James Luedtke, *Locally ideal formulations for piecewise linear functions with indicator variables.*, Oper. Res. Lett. **41** (2013), no. 6, 627–632.

[SS05] Tommaso Schiavinotto and Thomas Stützle, *The linear ordering problem: Instances, search space analysis and algorithms*, Journal of Mathematical Modelling and Algorithms **3** (2005), no. 4, 367–402.

[Stu99] Jos F Sturm, *Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones*, Optimization methods and software **11** (1999), no. 1-4, 625–653.

[SW12] Domenico Salvagnin and Toby Walsh, *A hybrid MIP/CP approach for multi-activity shift scheduling*, Principles and Practice of Constraint Programming (Michela Milano, ed.), Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 633–646 (English).

[SZ15] Christoph Schwindt and Jürgen Zimmermann (eds.), *Handbook on project scheduling (vol. 1 + vol. 2)*, Springer, 2015.

[Taw10] Mohit Tawarmalani, *Inclusion certificates and simultaneous convexification of functions*, Optimization Online, 2010.

[TS05] Mohit Tawarmalani and Nikolaos V. Sahinidis, *A polyhedral branch-and-cut approach to global optimization*, Mathematical Programming **103** (2005), 225–249.

[VAN10] Juan Pablo Vielma, Shabbir Ahmed, and George Nemhauser, *Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions*, Operations Research **58** (2010), no. 2, 303–315.

[VB96] Lieven Vandenberghe and Stephen Boyd, *Semidefinite programming*, SIAM review **38** (1996), no. 1, 49–95.

[Vie15] Juan Pablo Vielma, *Mixed integer linear programming formulation techniques*, SIAM Review **57** (2015), no. 1, 3–57.

[VN11] Juan Pablo Vielma and George L. Nemhauser, *Modeling disjunctive constraints with a logarithmic number of binary variables and constraints*, Mathematical Programming **128** (2011), no. 1, 49–72.

[Wen16] Dieter Weninger, *Solving mixed-integer programs arising in production planning*, Ph.D. thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2016.

[ZdF13] Ming Zhao and Ismael R. de Farias, *The piecewise linear optimization polytope: new inequalities and intersection with semi-continuous constraints*, Mathematical Programming **141** (2013), no. 1-2, 217–255.

[Zip77] Paul H. Zipkin, *Aggregation in linear programming*, Ph.D. thesis, Yale University, 1977.