# Integer Multicommodity Flows in Optical Networks

Diplomarbeit

vorgelegt von

Matthias A.F. Peinhardt

März 2003

*Technische Universität Berlin*
*Fakultät 2 – Mathematik und Naturwissenschaften*
*Institut für Mathematik*
*Studiengang Technomathematik*

Erstgutachter: Prof. Dr. M. Grötschel
Zweitgutachter: Prof. Dr. R. Möhring

# Acknowledgements

This work has not been possible without the help of many people, I am greatly thankful.

I am grateful to my supervisor Prof. Martin Grötschel, who taught me what mathematics is about.

I'd like to thank my advisers Dr. Arie Koster and Adrian Zymolka for their support and proof-reading, and their (sometimes challenging) comments. All people at the Konrad-Zuse-Zentrum deserve credit for being addressees of my questions.

All that would not be possible without the (not only material) support I received from the Studienstiftung des deutschen Volkes.

Special thanks to E.N. and Sophie-Charlotte for cheering me up in long nights of typing.

I thank my family, who has always been an anchor throughout my life, and no words could express what I owe them.

Finally, my gratitude goes to Hannah for supporting me in many ways, in any aspect of life. Without this it all would not have been worth it...

# Contents

**1 Introduction** **1**

1.1 Notation and preliminaries . . . . . 2

**2 Problem description** **5**

2.1 Background of the problem . . . . . 5

2.1.1 Optical networks . . . . . 5

2.1.2 Optical network design . . . . . 7

2.2 Specification . . . . . 9

2.3 Variations of the problem . . . . . 10

2.4 Related work . . . . . 12

2.4.1 Complexity . . . . . 13

2.4.2 Theoretical results . . . . . 13

2.4.3 Research on approximation . . . . . 16

2.4.4 Research on solution methods . . . . . 16

**3 Model and formulations** **19**

3.1 Model . . . . . 19

3.2 Formulations . . . . . 21

3.2.1 Edge-flow based formulation . . . . . 21

3.2.2 Path-flow based formulation . . . . . 23

3.2.3 Resource-directed formulation . . . . . 24

3.2.4 Convex cost formulation . . . . . 25

3.3 Polyhedral investigation . . . . . 27

**4 Subgradient method** **31**

4.1 Theoretical framework . . . . . 31

4.1.1 General scheme . . . . . 32
4.1.2 Constrained programs . . . . . 35
4.2 Application to IMCF-N . . . . . 39
4.2.1 Step length selection . . . . . 42
4.2.2 Projection method . . . . . 43
4.2.3 Penalty functions . . . . . 47
4.2.4 Barrier functions . . . . . 47
4.2.5 Using barrier-penalty functions . . . . . 47
4.2.6 Exact penalty approach . . . . . 48
4.2.7 Obtaining integral solutions . . . . . 49

**5 Branch-and-cut method 53**
5.1 Theoretical framework . . . . . 53
5.1.1 Cutting plane method . . . . . 53
5.1.2 Branch-and-bound . . . . . 55
5.1.3 Branch-and-cut . . . . . 58
5.2 Application to IMCF-N . . . . . 59
5.2.1 Branching rule . . . . . 59
5.2.2 Node selection . . . . . 60
5.2.3 Heuristics . . . . . 60
5.2.4 Cutting planes . . . . . 60

**6 Computational results 63**
6.1 Implementation . . . . . 63
6.2 Test instances . . . . . 63
6.3 Results . . . . . 65
6.3.1 Results of the subgradient method . . . . . 65
6.3.2 Results of the branch-and-cut method . . . . . 67
6.3.3 Comparison of the results . . . . . 68

**7 Conclusions 69**
7.1 Evaluation of the subgradient approach . . . . . 69
7.2 Evaluation of the branch-and-cut approach . . . . . 70
7.3 Outlook . . . . . 70

**Zusammenfassung** **71**

**Abstract** **73**

**Computational results tables** **81**

# Chapter 1

# Introduction

The cost efficient design of telecommunication networks has earned more and more attention in the last decades. The increasing demands for communication require ongoing replanning of communication structures. Mathematical programming has been shown to be a powerful tool for this task. The design of optical networks differs from previous planning tasks in the sense that the integer routing of demands is essential. In this diploma thesis we study the problem of integer routing from a mathematical point of view, i.e., other aspects of optical network design like dimensioning and wavelength assignment are neglected. We focus on alternative formulation and solution methodologies.

Before we start the discussions, Section 1.1 is devoted to some notation and preliminaries.

In Chapter 2 we explain the main features of optical networks, and outline problems arising in the design of optical networks. We focus on one of the important subproblems of the design process: the routing of transmissions through these networks. This problem has often been considered in the past, as many applications, ranging from traffic control over good delivery to scheduling problems, can be reduced to it. We survey some of the research on this problem and variants of it, and outline hardness results for them.

In Chapter 3 we give a model of the problem under consideration and formulations of it. We compare the formulations and give reasons for a choice of two of the formulations. The chosen formulations have to our knowledge not been considered so far. For the chosen formulations we develop a heuristic algorithm described in Chapter 4, and an exact branch-and-cut algorithm described in Chapter 5.

The heuristic method is based on a subgradient method, whose basic ideas are outlined in Chapter 4. We show how the method can be applied to our problem, and develop several variants of the method to obtain a method best fitted to our purposes. We extend the basic ideas of the subgradient method by two heuristics to meet an inherent aspect of our problem, the integrality.

In Chapter 5, branch-and-cut is described and its application to the formulation we have chosen.

We implement the described methods, and develop special test instances of our problem to study the behaviour of the solution methods. The obtained results are compared to a benchmark implementation of a standard approach. The test instances and results are described in Chapter 6.

We assess our approach in Chapter 7, and formulate future tasks related to the problem under consideration.

## 1.1 Notation and preliminaries

In the next chapters the following notations will be used. We denote by $\mathbb{Z}_0^+ := \{0, 1, \dots\}$ the natural numbers, while $\mathbb{N} := \{1, 2, \dots\}$ are the natural numbers without zero. Furthermore let $[p] := \{1, \dots, p\}$ serve as an index set for any $p \in \mathbb{N}$. We will denote by $\mathbf{0}$ a vector of zeros, and by $\mathbf{1}$ a vector of ones, with an appropriate size.

We assume that the reader is familiar with the basic notations of asymptotical analysis, i.e., the symbols $\mathcal{O}$, $o$, $\Theta$ etc.

Throughout this work several graph structures are used. A *graph* $G$ is a tuple $(V, E)$, where $V$ is a finite set of vertices and $E$ is a finite set of edges. Each edge $e \in E$ connects exactly two vertices, called the edge's end vertices. So we write $e = vw$ to denote that $v$ and $w$ are the vertices connected by $e$. A *directed graph* is a graph whose edges have a particular direction. We write $e = (vw) \in E$ to indicate that $v \in V$ is the edge's *source vertex*, and $w \in V$ its *target vertex*. Directed edges are also called *arcs*. For $e = vw \in E$ we say that $e$ is *adjacent* to $v$ and $w$, and that $v$ and $w$ are *incident* to each other. This is adopted similarly in the case of directed graphs and edges.

A *path* $P$ in an undirected graph $G = (V, E)$ is an ordered set of edges and vertices of $G$. Let $P = \{v_0, e_1, v_1, e_2 \dots, e_k, v_k\}$, then $e_i$ and $e_{i+1}$ have to share an end vertex, for $i = 1, \dots, k-1$, that is, $e_i = v_{i-1}v_i$. For a path in a directed graph we additionally require that the target vertex of $e_i$ is the source vertex of $e_{i+1}$, for $i = 1, \dots, k-1$. We use the number of edges in $P$, $l_P = k$, to denote the length of path $P$. A circle $C$ is a closed path, i.e., $v_0 = v_k$ holds. We call paths and circles *simple*, if their vertices are distinct.

A (directed) graph $G$ is *planar*, if it can be drawn in the plane without any intersection of edges (arcs). For a planar graph $G$ and an embedding of $G$ in the plane, the plane is partitioned into connected subsets, called the *faces* of $G$. Every face of $G$ is bounded by a simple circle of $G$. If $G$ is planar and drawn in the plane, the edges (arcs) touching the unique infinite face of $G$ are the *boundary* of $G$. Note that the boundary of $G$ depends on the embedding of $G$ in the plane, so it is not uniquely described by $G$.

For some $S \subseteq V$ we write $\delta(S)$ to denote all edges that are adjacent to some vertex $v \in S$ and to some vertex $w \in V \setminus S$. In the case of directed graphs, we additionally use the notation $\delta^+(S)$ for the edge set whose edges have their source vertex in $S$ and their target vertex in $V \setminus S$. Furthermore, we may write $\delta^-(S)$ for $\delta^+(V \setminus S)$. For notational ease, for some vertex $v \in V$ we write $\delta(v)$ as a shorthand of $\delta(\{v\})$. When the graph we refer to should be noted explicitly, we write $\delta_G$.

Edge sets that can be expressed by $\delta(S)$, $\delta^+(S)$, and $\delta^-(S)$ arising for some $S \subset V$ are called *cuts*. If there are vertices $s, t \in V$ such that $s \in S$ and $t \notin S$ we say that $\delta(S)$ is a *s-t-cut*. If capacities $c_e \in \mathbb{R}$ are provided to the edges $e \in E$ of the graph, we consider

$c(\delta(S)) = \sum_{e \in \delta(S)} c_e$ as the capacity of the cut $\delta(S)$.

A *s-t-flow* in a directed graph $G$ is an assignment of nonnegative flow values to all arcs of $G$ such that the following holds: all flow leading into some vertex $v \neq s, t$ must leave $v$. Flow only arises at $s$ and vanishes at $t$. A $s$-$t$-flow in an undirected graph $G$ is given when there is an orientation of the edges in $G$ such that we obtain a flow in the achieved directed graph. The amount of flow effectively arising at $s$ (and vanishing at $t$) is called the *flow value.*

We state here the famous max-flow min-cut theorem, as it is one of the premium results in network theory and we make use of it throughout the next chapters.

**Theorem 1.1 (Ford, Fulkerson [19])** *The value of a maximum $s$-$t$-flow equals the value of a minimum $s$-$t$-cut.*

We recall here an extension of the cut definition, taking vertices into account, too. This will be useful as we have to deal with vertex capabilities.

**Definition 1.2 (General cut)** *Let $G = (V, E)$ be an undirected graph and $s, t \in V$ two distinct vertices. A* general $s$-$t$-cut *$\delta(S, B)$ is a pair $(L, B)$, $L \subseteq E$, $B \subset V$, with the following properties:*

- *$S \subseteq V$ contains $s$, but not $t$.*
- *$B$ contains only vertices that are incident to at least one edge in $\delta(S)$.*
- *The edge members $L$ of the general cut are the edges $\delta(S)$ except those incident to a vertex in $B$, that is $L = \delta(S) \setminus \delta(B)$.*

Stated differently, $(L, B)$ is a minimal pair of edges and vertices such that after removal of $L$ and $B$ no path from $s$ to $t$ exists. This definition is illustrated by some examples in Figure 1.1.

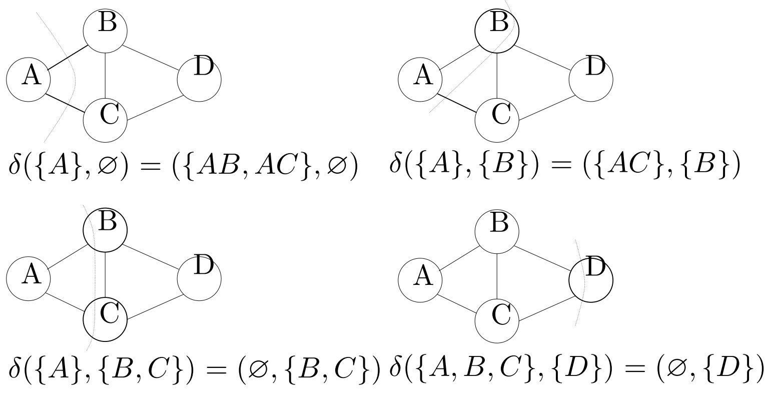

A
B
C
D
$\delta(\{A\}, \varnothing) = (\{AB, AC\}, \varnothing)$
$\delta(\{A\}, \{B\}) = (\{AC\}, \{B\})$
$\delta(\{A\}, \{B, C\}) = (\varnothing, \{B, C\})$
$\delta(\{A, B, C\}, \{D\}) = (\varnothing, \{D\})$

Figure 1.1: Some general $A$-$D$-cuts. The dotted lines illustrate where the graph is cut.

A *linear program* is an optimization problem with a linear cost function, whose feasible set is defined by linear inequalities. Every linear program can be stated as

$$\min c^T x, \tag{1.1a}$$

$$Ax \leq b, \tag{1.1b}$$

$$x \in \mathbb{R}^n, \tag{1.1c}$$

with $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n$. Note that the feasible set $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ always defines a polyhedron. For an introduction to linear programming we refer to [59].

A *mixed integer program* is a linear program with the additional constraint that specified variables must take integral values. Any mixed integer program can be stated as

$$\min c^T x + d^T y, \tag{1.2a}$$

$$Ax + By \leq b, \tag{1.2b}$$

$$(x, y) \in \mathbb{R}^{n_1} \times \mathbb{Z}^{n_2}, \tag{1.2c}$$

with $A \in \mathbb{Q}^{m \times n_1}, B \in \mathbb{Q}^{m \times n_2}, b \in \mathbb{Q}^m, c \in \mathbb{Q}^{n_1}, d \in \mathbb{Q}^{n_2}$. The restriction to rational constraint coefficients is necessary to ensure that the supremum of (1.1) is attained if it exists. For a detailed introduction on mixed integer programming see [53, 59, 69]. An *integer program* is a mixed integer program with $n_1 = 0$, i.e., it does not contain real variables, but only integral variables. A *LP-relaxation* (or linear relaxation) of a mixed integer program (1.2) is the linear program obtained by dropping the integrality constraints on the $y$-variables. In this thesis we assume familiarity with some basic theory on these topics.

# Chapter 2

# Problem description

Motivated by a practical application, we will describe the problem under consideration in this chapter. We give a short survey of variants of the problem and of related work to this class of problems. Finally, we resume some complexity results to throw light on the hardness of the problem.

## 2.1 Background of the problem

In the next paragraphs, we focus on optical networks, as they are the application of our work. We survey some properties of these networks and problems arising from their design.

### 2.1.1 Optical networks

Globalization and the tendency to knowledge based societies challenge many fields of science and technology. The increasing need for information exchange and communication capabilities yields rising demands for both technical and structural progress.

Communication is organized with hierarchical networks. On the lowest level of the hierarchy, connection demands of single users or computers are processed in a network connected to the next higher level of the hierarchy. In this second level a whole city or region is connected and linked to the next level and so on. On the most upper level, countries and continents must be connected via a wide-area network. Clearly, the traffic to be managed increases from level to level. The core network (or backbone network) at top level has to handle traffic amounts in the order of terabits or even petabits per second.

Large global and national networks capable of transporting huge amounts of data are needed to meet the desired capabilities. Optical networks are telecommunication networks based on optical, digital processing of signals. These networks provide high speed, high bandwidth data and communication ability. Since optics are superior to previous electronic technology in terms of speed, bandwidth, and costs, optical networks are mostly used as wide-area backbone networks connecting continents, countries, or regions.

Optical networks consist of a set of nodes connected by optical fibers. We distinguish different layers, depicting different aspects of the network. The *physical layer* contains the hardware to carry the signals and to process them on the links and in the switching nodes. Signals are fed into the network at a node, transported along the links, and arrive at some other switching node as their destination. Hence, the physical layer can be represented by a graph, where switching nodes are represented by vertices and fiber links by edges. We consider the links to be able to carry signals in both directions, so the graph of the physical layer is undirected.

In first generation optical networks, the optical signal was transformed to an electronic signal, switched, and then transmitted again as an optical signal to the next switching node. This optic-electronic-optic conversion (*o-e-o conversion*) used to be a bottleneck of optical networks due to the limited switching speed of electronic components. This has been overcome in second generation optical networks, where the switching can be done fully optical without any o-e-o conversion. This is the reason why these networks are often called *all-optical networks*.

A *lightpath* consists of a bidirectional optical channel connecting two nodes in the network via a path of physical links. On each such link, a lightpath uses a wavelength inside an optical fiber to transmit information. At each crossed node, the lightpath is switched to the next physical link. These lightpaths form the *logical layer* of the network, as a lightpath allows communication between the lightpath's end nodes, see Figure 2.1. A lightpath establishes a direct communication channel as no intermediate conversion to electric signal must be performed. Clearly, multiple lightpaths can connect the same two nodes. How

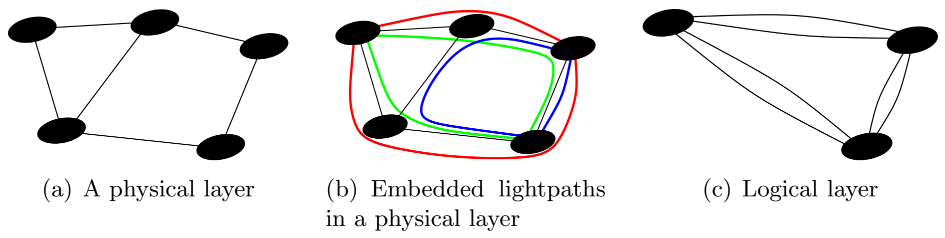

(a) A physical layer (b) Embedded lightpaths in a physical layer (c) Logical layer

Figure 2.1: A physical layer and an embedded logical layer.

many lightpaths must be established between two nodes depends on the bandwidth that shall be provided. In general, lightpaths can be operated with different bandwidths, but we focus on lightpaths with equal bandwidth only. Otherwise more aspects of communication would have to be considered. For example, we were asked to decide whether lightpaths with different bandwidth may be connected, and if so, how this connection must be handled.

After introducing the main concepts of optical networks, we explain the used hardware in more detail. The switching nodes consist of *optical cross-connects* (OXCs). These OXCs provide a number of input- and output-ports, connected by a switching grid. Lightpaths attending a switching node can be switched independently onto the next link. At its two end nodes the lightpath is fed into the network via transmitters and receivers. Transmitters generate the optical signal by use of a laser device, whereas receivers transform optical signals back into electronic form by photodiodes. In addition, regenerators are usually placed at the nodes. These regenerators are placed between two lightpaths to refresh a signal that uses both lightpaths. This regeneration is mainly an o-e-o conversion.

Optical fibers can carry multiple signals at the same time using *wavelength division multiplexing* (WDM). That means, that different signals use different wavelengths on a fiber in parallel. In *simplex fibers* both directions for optical signals are realized on the same glass core, whereas in *duplex fibers* the same capacity is provided in both directions by a pair of glass cores. However, these capacities can be used only in the predetermined direction.

For each fiber a WDM system is required, essentially consisting of a *multiplexer* and a *demultiplexer*. A multiplexer bundles optical signals carried by different wavelengths and sends the combined signal onto the fiber. Entering the next node, the demultiplexer splits such signals in order to switch the single signals.

The optical signal is subject to signal intensity loss due to interference, attenuation, and dispersion, depending on the fiber's quality. Consequently, the installable WDM systems depend on this quality, too. Moreover, the signal must possibly be amplified, reshaped, or retimed along the fiber. Unfortunately, a full signal regeneration can not be done fully optical yet. That's why there is a length bound for lightpaths depending on the hardware (e.g., the fiber quality) and the technology used (e.g., the bit rate). If a lightpath would violate this length bound, we must provide several lightpaths instead and a signal regeneration must be supplied between these partial lightpaths.

Finally, *wavelength conversion* must be taken into account. Possibly, a lightpath can not use the same wavelength throughout the used fibers, mainly because of limited resources. In such cases, the optical signal must be converted onto a different wavelength at some node. Presently, no optical wavelength conversion is possible, so it is supplied by a receiver-transmitter-pair. This conversion allows a single lightpath on any wavelength to be transformed to any other available wavelength. In comparison to this *full range* conversion other techniques may be used, e.g. *limited range* conversion. Here, a wavelength can only be converted to a subset of available wavelengths, namely its spectral neighbours. Another possibility is to assign conversion capability to the whole node, that is, any lightpath visiting this node can change its wavelength.

### 2.1.2 Optical network design

We next turn to the design process of optical networks. First of all, we have to define the demands for the network. Demands in optical networks are often considered to be *quasi-static*. That means, the demand to be fulfilled is assumed to be static. This is due to the fact that in backbone networks many demands from lower levels in the whole network hierarchy are multiplexed. Moreover, the demands are considered as static because changing the logical layer is somewhat difficult. So the demands will be given, e.g., from past requirements or forecasts. Clearly, in larger time intervals the demands can change, and even the network must be updated. Then the design process must be repeated. So it might be part of the optical network design problem that an existing optical network has to be extended instead of greenfield planning. Finally, the demands are specified in numbers of lightpaths needed to satisfy the actual requirements.

Demands can be directed or undirected. Undirected demands may reflect symmetrical demands in both directions. Further restrictions on the routing may be considered. For example it can be required that symmetrical demands must be routed symmetrically, that means, their lightpaths must use the same links.

Designing an optical network consists of three parts:

1. Dimensioning

   Given the demands, enough hardware has to be installed to provide the necessary routing capacities. Possibly, this capacitation shall be done on top of an existing network.

   Thereby, topology decisions can be included, like decisions on which of the links can be established at all.

   There can be restrictions how much hardware of some type can be installed at a special site.

2. Routing

   Once routing capacities are provided by the physical layer, the routing itself has to be established. That means, given the network capacities and the network demands, the logical layer has to be set up. If needed, survivability requirements must be met. Additionally, length bounds of the lightpaths must be considered. Thus, two or more lightpaths have possibly to be routed to establish a single channel.

3. Wavelength assignment

   When the routing of the lightpaths is given, a wavelength must be assigned to each lightpath. This wavelength must be provided on all links the lightpath uses. In case that wavelength conversion is possible, a wavelength must be provided to any link of the lightpath.

The main goal is usually to satisfy all demands at least cost, where costs arise for each (newly) installed hardware component. Minimizing the costs is naturally the network operator's goal. Although all three tasks described above influence each other, solving them altogether is usually a very hard problem. The hardness of the design process arises from the fact that many decisions must be made, so the design problem yields naturally large instances with respect to the number of singular decisions. Addionally, the relations between all these decisions are complicated, and the incorporation of costs and the objective complicates the design problem even more. Because of the hardness of the design problem decomposing the problem is a common approach. That is, dimensioning, routing, and wavelength assignment are carried out sequentially. Some research deals with solving two of the mentioned parts together. Usually though this is still hard for relevant instance sizes, i.e., arising in real world networks.

We next focus on survivability concepts. Survivability means to ensure that the impact of network failures is as low as possible within reasonable additional costs. Due to its high performance nature, a break-down of even a single component may lead to dramatic loss of traffic capability of the network. Survivability concepts affect the dimensioning of the network and the routing. In the dimensioning part enough capacities must be provided to admit an appropriate routing. In the routing part such a routing must be established that fulfils the survivability requirements.

Different approaches were made to keep the impact of network failures as low as possible. Among them are path-restoration, link-restoration, and *diversification*, see [68] for an introduction. The latter concept will be considered in this work. Diversification is a concept that bounds the traffic through any network link or node of any demand. So, a break-down of one component of the network can only affect a certain portion of each demand.

## 2.2 Specification

In this section we will state the problem under consideration in this work. As already mentioned, the network design problem is very hard, so decomposing this problem is a reasonable approach. We use the decomposition mentioned in Section 2.1.2. Our work focus on the routing problem, that is part of the overall network design problem.

Within the dimensioning process the physical layer has been determined. So we are given a supply graph $G = (V, E)$ associated with the physical layer. For each established link there is an edge in $E$, and for each switching node there is a vertex in $V$.

By the installation of fibers and WDM systems, routing capacity is provided at the links. We regard all routing capacities as homogeneous. That is, we neither distinguish between channels of different WDM systems and fibers of the same link, nor do we distinguish between channels of the same WDM system. Hence, for any edge $e \in E$ there is an associated *edge capacity* $c_e$. This edge capacity is defined by the number of lightpaths that can be routed across the link associated with $e$. All edge capacities are positive integers, as all hardware is addressed to whole optical channels only, see Section 2.1.1.

Similarly, by the installation of OXCs there is a routing capacity for all nodes. Each pair of input and output ports in an OXC admits the routing of one lightpath through the switching node. The routing capacity of a node is the number of lightpaths that can cross the node. For each node there is an associated vertex $v \in V$, and the routing capacity of the node yields a *vertex capacity* $c_v$. Since we deal with port pairs, the vertex capacity is again a positive integer.

After we have described the supply graph, we next describe how demands are incorporated. The demands are given by an undirected demand graph $H = (T, F)$, where the nodes $T \subseteq V$ are called *terminals*. We denote by $\mathcal{Q}$ the set of demands or commodities. To any demand $k \in \mathcal{Q}$, there is an associated demand edge $f_k \in F$ with end nodes $s_k$ and $t_k$. Additionally, there is a demand value $d_k$, an integer describing the number of lightpaths to be routed between $s_k \in T$ and $t_k \in T$. Finally, we have a diversification parameter $\rho_k \in (0, 1]$ for each demand $k \in \mathcal{Q}$.

The objective is to find a maximum number of $s_k$-$t_k$-paths in $G$, but at most $d_k$ for each demand $k$. Each path consumes one unit of capacity on each edge and node it crosses, including its end terminals. The overall set of lightpaths must not exceed the mutual edge and node capacities given. That means, the number of paths crossing an edge or vertex is bound by the edge or vertex capacity, respectively, regardless of the demand the paths are serving.

Furthermore, the set of paths belonging to a single demand $k$ has to fulfil the diversification requirements, that is, no more than $\lfloor \rho_k d_k \rfloor$ of these lightpaths use the same edge or vertex, except the end terminals of this demand. This ensures that in case of a failure in any physical link or node (except $s_k, t_k$) at least $d_k - \lfloor \rho_k d_k \rfloor$ paths survive for demand $k$. Of course a failure of $s_k$ or $t_k$ causes the loss of all paths for this demand.

A maximum number of paths has to be established at minimum cost. In the network design problem as stated in Section 2.1.2 we don't consider costs of routings. However, in order to use the resources "economically" we aim to find routings that use as few capacities as possible. Note, that free capacities can be used for future enlargements of demands. There is still another advantage of saving capacities: if some node or link carries less paths, then a failure at this component will break less paths.

To minimize the utilization we assign costs to it. We aim to minimize the overall utilization of only the edges. Since the number of vertices touched by a path depends on the number of edges it passes, the utilization of vertices is already considered implicitly. That's why we will not take vertex utilization into account for the cost function explicitly. Formally, each path $P$ causes $l_P$ cost units. Recall that $l_P$ is the number of edges of the path.

The routing of each demand can be seen as a network flow. For that, we consider $s_k$ as a source of commodity $k$. Additionally, $t_k$ is the target of commodity $k$. We aim to send flow from the source to the target, and at all vertices in between no flow can vanish or arise. As we are interested in whole paths only, the flow of any commodity on any edge must be integral. The integral flow for each demand can be decomposed in a number of paths and cycles. Since we want to obtain a lightpath routing, we restrict the flow for every demand to be cycle-free. That means, the flow must be decomposable into paths only.

The mutual capacity constraints for the path routing correspond directly to capacity constraints for flows. That means, that all flow sent through any vertex or edge by any demand must not exceed the vertex' or edge's capacity.

As the main ingredients of our problem are the **i**ntegrality, the **m**ulti**c**ommodity **f**low structure, and the **n**ode capacities, from here on we will denote our problem as IMCF-N.

## 2.3 Variations of the problem

The above specified problem can be varied in many ways. We want to outline some variations of the multicommodity flow problem. These variations concern different flow definitions, different mutual capacity constraints, different graph structures, or different objectives. Among the different flow types described in literature we mention fractional flows, integral flows, and unsplittable flows. Widely studied objectives are maximum flow, minimum cost flow, minimum congestion flow, and maximum concurrent flow. Graph structures of interest are therefore planar graphs and graphs with even degree of every vertex. Another possibility is to subdivide multicommodity flow problems for directed and undirected graphs. Since many combinations of objectives, flows, and special graph structures are possible, a wide variety of problems may arise.

### Fractional multicommodity flow

The somewhat “standard” multicommodity flow problem concerns directed supply and demand graphs. This can be seen as the base problem, since undirected supply edges and demands can usually be modelled in a directed setting, see [1] for the basic ideas of the transformation. Thus the demand’s terminals become sinks and sources, indicating where the paths start and end. Furthermore, no vertex capacities are assumed, as capacitated vertices can be modelled as (directed) edges, see again [1]. In this variant, no diversification or other survivability constraints are given, as such constraints heavily depend on the application. Finally we may drop the integrality constraint, this means that any path can carry some arbitrary positive amount of flow instead of just one unit. This is why the term ’fractional’ is in the problem’s name. The flows of each demand can be generalized further to admit multiple sinks and sources for each commodity.

### Unsplittable multicommodity flow

In the unsplittable multicommodity flow problem (also called non-bifurcated flow problem), the routing of each demand has to use only one path in the network. We might have to choose to route the whole demand on a single path or not, or we’re allowed to route any fraction of the demand on its single path. Unsplittable flows can be considered with specified demands, for example in computer networks, but usually no demand values are given.

### Edge-disjoint paths

Another classical problem is the edge-disjoint paths problem. Given an, usually undirected, graph and a (multi-)set of vertex pairs, we want to find a maximum number of paths connecting these vertex pairs. These paths must be edge-disjoint. This problem can be seen as an integer multicommodity flow problem, and generalizes naturally to an integer unsplittable flow problem, when taking edge capacities and demand values into account. To see this, assume we have an integral multicommodity flow problem where edge capacities are given. Then we can replace an edge with edge capacity $c_e$ by $c_e$ parallel edges with unit capacity. If demands are specified, we can do the same replication in the demand graph.

Next we mention some objectives that can be applied.

### Maximum multicommodity flow (Max MCF)

In a maximum multicommdity flow, no demand values are specified. The objective is to maximize the sum of flow established between any terminal pair.

### Minimum cost multicommodity flow (MinCost MCF)

Here, it is assumed that all demands can be satisfied. The aim is to find a feasible routing of all demands at least cost, with respect to some cost function. Note, that our problem IMCF-N can be seen as a two stage combination of MaxMCF and MinCost MCF, as we first want to find a maximum flow, and then to obtain this maximum flow with minimum cost.

#### Minimum congestion flow (MinCong)

In this variant, specified demands must be completely satisfied. We consider the congestion of edges, that is the ratio of utilization of the edge and its capacity. The congestion $\theta$ of the whole network is the maximum congestion of the edges. This network congestion shall be minimized. If the edge capacities are $c_e$ for edge $e$, a congestion $\theta$ means that edge capacities $\theta c_e$ would suffice to fulfil all demands without overutilization. This is a matter of interest in computer network communication, as we consider to route the all commodities in $\lceil\theta\rceil$ rounds. In our problem IMCF-N, a routing would be feasible if the network congestion is less than or equal to one.

#### Maximum concurrent flow (MaxConc)

Here, we are interested in the portion of each demand that can be routed in the network. That is, we want to maximize the concurrent $\lambda$, where $\lambda d_k$ of each demand shall be routed. The minimum congestion flow problem can be seen as equivalent to the maximum concurrent flow problem, as a network congestion $\theta$ allows to route $\theta^{-1}$ of the demands, and a flow with concurrent $\lambda$ yields a routing of the whole demands with congestion $\lambda^{-1}$.

## 2.4 Related work

As there are many variants of the problem, and quite a lot of mathematical formulations for them, a wide spread research related to the problem has been done. We give a short survey on the work about theoretical aspects, approximation algorithms, and proposed solution methods. This survey whatsoever does not claim to be exhaustive. For a more detailed survey on such results and related topics see [44]. The considerations given here aim to explain the hardness of our problem IMCF-N, and to outline the variety of solution approaches.

First, we want to state the *cut condition*, as it is of crucial importance throughout the rest of this work. It is closely related to the max-flow min-cut theorem, but extending to multicommodity flows.

**Definition 2.1** *Let $G = (V, E)$ be a graph with edge capacities $c_e$ for every edge $e \in E$. Let there be a demand graph $H$ as described in Section 2.2, defining $q$ demands. The demands have terminals $s_i$, $t_i$ and a positive real number $demand_i$ specifying the demand value of demand $i$.*

*Let the min-ratio cut be given by*

$$\rho^* = \min_{\varnothing \neq S \subset V} \frac{\sum_{e \in \delta_G(S)} c_e}{\sum_{f \in \delta_H(S)} d_f}.$$

*The cut condition holds, when $\rho^* \geq 1$*

In simple words, the cut condition holds when the capacity of any cut is at least as large as the demand separated by the cut. We easily see that for $q = 1$ the cut condition is equivalent to the existence of a $s_1$-$t_1$-flow of value $d_1$, due to the max-flow min-cut theorem 1.1.

### 2.4.1 Complexity

Before investigating the problem under consideration, we classify it and some variants in terms of complexity. We first reflect the hardness of the considered problem and then provide related results.

First of all, the fractional multicommodity problem with any objective function is solvable in polynomial time, since it can be formulated as a linear program with a polynomial number of variables and inequalities, and linear programming is solvable in polynomial time with the ellipsoid method [30].

The maximum edge-disjoint paths problem is known to be $\mathcal{NP}$-hard, see [25]. If we use a diversification parameter $\rho = 1$ for all demands and set the vertex capacities arbitrary high, and set the demand values $d_k$ arbitrary high for all demands, our problem IMCF-N turns out to be the maximum edge-disjoint paths problem. Since we can conclude that the maximum edge-disjoint path is contained in IMCF-N, our problem is $\mathcal{NP}$-hard as well.

The integer maximum multicommodity flow problem was proven to be MAX SNP-hard for undirected graphs, see [26]. That is, there is no polynomial approximation scheme for this problem unless $\mathcal{P} = \mathcal{NP}$. Moreover, Ma and Wang [50] showed that this problem cannot be approximated for directed graphs within ratio $2^{\log^{1-\epsilon} n}$ unless $\mathcal{NP} \subseteq DTIME[2^{\text{polylog}(n)}]$. Here, polylog$(k)$ denotes a function asymptotical bounded by some polynomial of fixed degree of $\log(k)$ (to put it differently: polylog$(k) := \log^{\mathcal{O}(1)} k$).

Additionally, for every fixed nonnegative integer $K$ it is still $\mathcal{NP}$-complete to decide whether an instance is solvable with the demand values decreased by $K$, even when the original instance is fractionally solvable. Furthermore, the maximum integer multicommodity problem is MAX SNP-hard even when $G$ is a tree and the edge capacities are large (i.e. $c_e \geq C$ with $C \in \mathcal{O}(\text{poly}(m))$ ), see [65]. Similar to the notation used above, poly$(k)$ denotes $k^{\mathcal{O}(1)}$.

### 2.4.2 Theoretical results

A series of results deals with the max-flow min-cut ratio for fractional multicommodity flows. This ratio describes how large the min-ratio cut must be to allow all demands to be satisfied. The max-flow min-cut ratio is defined by $\kappa^* = \frac{\rho^*}{\theta^*}$, where $\theta^*$ is the maximum concurrent. Clearly, $\kappa^* \geq 1$, since on every cut $\delta_G(S)$ we need capacity at least as large as the portion $\theta^*$ of the separated demands. Plotkin and Tardos [57] were the first to show an upper bound on $\kappa^*$ independent of the capacities and demand values. They obtained an approximation of $\mathcal{O}(q^2)$ for $\kappa^*$. This result was improved to $\mathcal{O}(q)$ by Aumann and Rabani [4]. Finally, Günlük [31] showed that $\kappa^*$ is $\mathcal{O}(q^*)$ when $q^* > 1$, where $q^*$ is the cardinality of the minimal vertex cover of the demand graph. The improvement of the last bound can be seen when considering a demand graph consisting of two stars, spanning all vertices of $V$. Then $q = |V| - 2$, but $q^* = 2$. Günlük also showed that his bound is tight in the sense that for any $n \in \mathbb{N}$ there is an instance with $n$ vertices, where this bound is tight up to a constant.

There is a famous characterization of the solvability of fractional multicommodity flows with specified demands. Here, solvability means that all demands must be fully satisfied. The result is based on duality theory of linear programming, and extends the (obviously necessary) cut condition to a sufficient condition for solvability of the fractional multicommodity flow problem.

**Theorem 2.2 (Kakusho and Onaga, [39]; Iri, [38])** *A capacity vector c yields a feasible multicommodity flow if and only if*

$$\sum_{e \in E} \mu_e c_e \geq \sum_{k \in Q} \pi_k d_k, \tag{2.1}$$

*for all edge weights $\mu_e$, $e \in E$. Here, $\pi_k$ is the value of a shortest $s_k$-$t_k$ path in $G$ with respect to the edge weights $\mu$.*

The inequalities (2.1) are called *metric inequalities*, and play an important role for the dimensioning problem of networks, see for example [68].

We next turn to integral flows. As mentioned above, integral flows can be seen as the question for edge-disjoint paths connecting predefined terminal pairs. In case of one commodity, the maximum edge-disjoint paths problem was considered as early as 1927 by Menger [52]. He proved that the maximum number of edge-disjoint $s$-$t$ paths in a graph $G$ for two vertices $s$, $t$ in $G$ is equal to the minimum number of edges in a $s$-$t$ cut.

Extending to multiple commodities, much research was done on the characterizations of instances that admit an integral multicommodity flow satisfying all given demands. A necessary condition for that is obviously the cut condition 2.1, but it is far from being sufficient. Consider the example in Figure 2.2, where all edge capacities and both values are one. The cut condition holds, but not both demands can be connected by one path.

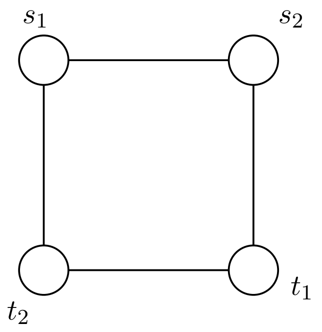


Figure 2.2: An example where the cut condition don't suffice.

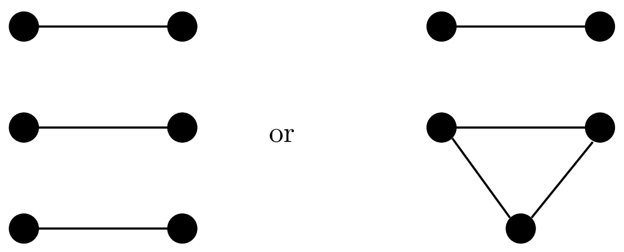


Figure 2.3: Forbidden configurations for Frank's theorem.

Among other results, Frank [23] proved the following characterization: if all edge capacities are integral and the demand graph does not contain any of the two configurations in figure 2.3, then the cut condition is equivalent to the existence of a half-integral multicommodity flow. A half-integral multicommodity flow is a multicommodity flow where the flows of all demands on all edges are a half of an integer. Excluding the configurations in figure 2.3 means that the demand graph is either (i) the complete graph $K_4$, (ii) the circuit $C_5$, or (iii) the union of two stars.

We next mention some results dealing with planarity of the incorporated graphs and the *parity condition*. The parity condition holds when for all vertices $v$ in the supply graph $\sum_{vw \in E} c_{vw} + \sum_{i \in \mathcal{Q}: v \in \{s_i, t_i\}} d_i$ is even, that is when the capacities of all edges adjacent to $v$ has the same parity as the demand emanating at $v$. Here, $\mathcal{Q}$ is again the set of demands and the $d_i$ are given demand values. Considering an integral multicommodity flow problem as an edge-disjoint paths problem by the replication construction mentioned above, the parity condition states that $G \cup H$ is Eulerian.

Okamura and Seymour [55] showed, that if the supply graph $G$ is planar, the parity condition holds, and all demand terminals lie on the boundary of some face of $G$, then the cut condition is equivalent to the existence of an integral multicommodity flow satisfying all demands. The proof given by the authors is constructive, yielding an algorithm of finding such a multicommodity flow. Wagner and Weihe [67] improved this to an algorithm with linear running time in the number of vertices. Okamura [54] weakened his result to the condition that the demand terminals may lie on two boundaries of faces of $G$, but both terminals of any demand must lie on the same of the two boundaries.

Finally, we consider the case when $G \cup H$ is planar. In this case, and if the parity condition holds, then there is an integral multicommodity flow satisfying all demands if and only if the cut condition holds, see [61]. When we don't assume the parity condition, similar results hold. If $G \cup H$ is planar and the number of demands is fixed (or bounded), Sebö [60] showed that the integer multicommodity flow problem is polynomially solvable. Finally, the two last mentioned results were generalized by Korach and Penn [43]. In case $G \cup H$ is planar and if the cut condition holds, they gave a polynomial time construction of a solution.

We say that $G$ is *inner Eulerian*, if the sum of capacities of the edges adjacent to some non-terminal vertex is even, for all non-terminal vertices. Among other results, Karzanov [40] showed that for inner Eulerian supply graphs together with the condition, that the graph of anticliques $A(H)$ of $H$ is bipartite, the problem has an integer-valued optimal solution, for a simpler proof see [24]). The anticliques of the demand graph are the maximal, with respect to inclusion, independent vertex sets of $H$. The graph of the anticliques has a vertex for every anticlique of $H$, and an edge between two such vertices when the corresponding anticliques have a common terminal in $H$. $A(H)$ is said to be bipartite if its vertice set can be partitioned into two independent sets. Karzanov gave a polynomial time algorithm constructing such a solution, too. Examples of such demand graphs are for instance complete graphs.

Summing up the result presented here, we can conclude that easy solvable instances are rare, and we can not hope that such instances for our problem IMCF-N might appear.

### 2.4.3 Research on approximation

Skutella [63] gave approximations for the unsplittable multicommodity flow problem. In the version under his consideration all demands have to share a common source vertex. In this case it is still $\mathcal{NP}$-complete to decide whether all demands can be satisfied, since it contains the knapsack problem. The approximations obtained for various objectives like minimum congestion, maximum flow are all constant factor approximations.

Baveja and Srinivasan [8] studied approximation algorithms for the integer unsplittable flow problem, and obtained an $\mathcal{O}(\sqrt{m})$ approximation, where $m$ is the number of edges in the graph. If the objective value is large enough, i.e. if it is larger than $m$, they could even show a $\Theta(1)$ approximation.

In [64], the integral minimum congestion problem, both in unsplittable and splittable setting, is studied by Srinivasan. Supply graphs and demand graphs may be directed or undirected. Using the *Randomized Rounding* approach, they get an $1 + o(1)$ approximation when the fractional solution, that is the minimum congestion without fixing the flows to integers, grows faster than $\log m$. Randomized Rounding relies on the idea of rounding fractional solutions due to some sophisticated rounding scheme, that exploits the structure of both constraints and objectives.

Fleischer [18] gave the fastest polynomial approximation schemes for fractional multicommodity flow problems to this day. Most important, for the maximum multicommodity flow problem, their algorithm's runtime is independent of the number of commodities. When no costs are given, the approximation schemes are fully polynomial.

For the maximum integral multicommodity flow problem, Garg, Vazirani, and Yannakakis [26] gave a $\frac{1}{2}$-approximation. In the version they considered no demand values are specified, and the objective is to maximize the total demand that can be fulfilled. The supply graph is restricted to be a tree.

### 2.4.4 Research on solution methods

A general introduction to solution methods for fractional multicommodity flow problems can be found in [41, 1], the latter from a more modern point of view. A survey on models and solution approaches for rail transportation can be found in [3]. In rail transportation usually a timetable has to be served at minimum cost. Costs arise by routings for trains and possibly by extending the capacities of the track network. When the capacities may be extended we obtain an extension of multicommodity flow problems, known as *multicommodity fixed charge network design problem.* In terms of Section 2.1.2, the dimensioning and routing problem is solved together. The same railway routing problem is considered in [10], too, where the instances are solved by simulated annealing.

In [51], McBride and Mamer solved the fractional multicommodity problem on undirected graphs. They exploited the undirected structure within a formulation with piecewise linear convex objective and constraints. They modified the primal simplex algorithm to deal with this convexity. Additionally, they proposed a special pricing procedure, thus keeping the size of the formulation relatively small. An exploitation of the simplex method with similar ideas is given in [17]. The main goal of the authors is to keep the matrices involved in computation small. They interpret the derived data structures in terms of the original network structures.

The computational results in [51] suggest that their approach is superior to standard techniques for the undirected setting. Such standard techniques mostly deal with directed graphs derived from the original undirected graph, leading to larger graphs than the original graph. In Section 3.2 we give an example for such constructions.

Barnhart, Hane, and Vance [7] studied a branch-and-price algorithm for solving the integer minimum cost unsplittable multicommodity flow problem. They used a path based formulation and proposed a branching rule based on arc flows. They point out that their approach outperforms standard branching rules.

Alvelos and de Carvalho [2] extended this idea to problems without the unsplitting constraint. Their results show that this approach is competitive to an edge based formulation solved with a standard integer programming code.

A more general integer multicommodity flow problem arising as a transportation problem in the airplane industry is studied in [66]. The extra difficulties are due to the fact that commodities can not be shipped independently from each other, producing nonlinear side constraints. Both approximation and exact solution methods are proposed. The exact approach is a branch-and-price algorithm working on a path based formulation, thus overcoming the problem of nonlinear constraints.

Löbel [48] studied vehicle scheduling problems, and derived a minimum cost integer multicommodity flow model with unit demands for them. He obtained an integer program via an edge-flow formulation, that was solved by branch-and-cut. His solution approach included column generation schemes as well as Lagrangian relaxations to obtain lower bounds. The Lagrangian relaxations were solved by a subgradient method. He reported promising results for instances with up to 25000 commodities.

In [22] bundle methods are used to solve minimum cost fractional multicommodity flow problems. Bundle methods are extensions of subgradient methods that we study in chapter 4. The authors applied their codes on so called cost-decomposition formulations, and found them competitive to general purpose LP-solvers. The bundle approach is extended to multicommodity fixed charge network design problems in [13]. There, the LP relaxation of of a typical edge-flow formulation (see Section 3.2.1) had to be solved. Again, the authors found the bundle methods to be superior to a standard linear programming techniques like simplex and barrier optimizers.

The short survey in this section shows that due to the large number of variations of multicommodity flow problems many solution approaches have been considered. Up to this point, no approach superior to any other can be detected.

# Chapter 3

# Model and formulations

In this chapter we present a model for the problem IMCF-N introduced in chapter 2 and formulate the optimization task as several integer programs. We examine some properties of the underlying mathematical objects, namely polyhedra, to motivate why we use the presented formulations.

## 3.1 Model

As already pointed out in Section 2.2, we can model the desired routing as network flows, one for each commodity. Network flows are extensively studied objects in combinatorial optimization and well understood. For a survey see e.g., [1]. A flow of commodity $k \in \mathcal{Q}$ is a map that assigns a flow value and a direction to every edge of the supply graph. So, for every edge $vw$, a flow for demand $k$ determines which amount of commodity $k$ is shipped from $v$ to $w$ or from $w$ to $v$.

For every commodity $k$, we assign a deficit of commodity $k$ to every vertex. The deficit of commodity $k$ at some vertex $v$ determines how much the flow towards $v$ must exceed the flow away from $v$. This property is called *flow conservation*, since no flow vanishes or raises except to balance the vertex deficits.

In section 2.2 we defined light paths and routings as undirected. We consider directed flows only, essentially because modelling undirected flows is much more circumstantial than for directed flows. The direction of flows has no influence on their treatment with respect to capacities. Since we deal with directed flows only, we have to assign an arbitrary direction to every demand.

The variable $a_k$ describes how many of the $d_k$ desired paths can not be established, and hence how much flow cannot be sent from $s_k$ to $t_k$. We model the terminal $s_k$ as a source of flow where an excess of $d_k - a_k$ is located, and assign a deficit of $-(d_k - a_k)$ of commodity $k$ to $s_k$. The second terminal $t_k$ is modelled as the target of the flow for demand $k$, and we assign a deficit of $d_k - a_k$ to it. All other vertices except $s_k$ and $t_k$ are assigned a deficit 0 with respect to commodity $k$. The reason for introducing variables $a_k$ instead of, e.g., variables for the flow really sent, is that on the one hand we want to minimize the flow costs. On the other hand, we want to maximize the flow sent between the commodity's terminals. To incorporate both goals the latter one is modelled as to minimize the flow not sent. So both goals are minimizing objectives.

The objective is firstly to route as much demand as possible, and secondly to minimize the edge utilization, without any preference among the commodities or edges.

The first goal is equivalent to minimize the sum of the $a_k$, while the second goal is equivalent to minimize the sum of edge flows for every commodity. To control the trade-off between the two goals, the $a_k$ are assigned a cost coefficient $M$, while the cost coefficients for all edge flows are 1. A large value of $M$ emphasizes on minimizing the demand not served, while a small $M$ emphasizes to save capacities. The problem specified in section 2.2 asks for a maximum satisfaction of the demands as a primary goal. How large should $M$ be to provide this emphasis on routing demands? As any path connecting two terminals could only use $|V| - 1$ edges without crossing any node twice, $M = |V|$ suffices. Choosing this value for $M$, self-intersecting paths with more than $|V|$ edges can not occur in an optimal solution. However, this does not destroy the emphasis on demand satisfaction. Suppose we are given an optimal solution, and there is a self-intersecting path $p$ satisfying another unit of demand. Then clearly $p$ contains cycles. Leaving out $p$'s cycles yields a shorter path $p\prime$ with at most $|V| - 1$ edges. The new path $p\prime$ without cycles will not utilize any edge or vertex more than the original path $p$ does. So $p\prime$ can be used instead of $p$, and the objective-driven restriction to paths with at most $|V|$ does not affect the validity of our model.

As already mentioned, while minimizing the edge utilization we also minimize the node utilization. The node utilization depends directly on the edge utilization, since a path using $t$ edges will use exactly $t + 1$ nodes.

A well known property of network flows is that such a flow can be decomposed into a set of paths and cycles, see e.g., [9] for an algorithm for that decomposition. Moreover, the algorithms for such a decomposition task have polynomial running time. Note however, that this decomposition is not unique, that means, a flow for some commodity can be decomposed into paths and cycles in several ways.

As we stated the problem in section 2.2, the flow of every commodity must be cycle-free. In the presented models, we do not restrict explicitly the flows to be cycle-free. The reason for this relaxation is that we can derive a cycle-free flow from any given flow easily. Suppose we are given a flow for some commodity $k$, and we decompose it in a number of paths and cycles. By simply dropping the cycles we obtain a new flow for commodity $k$, that does not use more edge and node capacities than the originally given flow. Moreover, as any cycle contains edges, dropping the cycles will decrease the cost for the commodity's flow. Clearly, as we keep all paths of the flow decomposition, we sent the same amount of the commodity $k$ from $s_k$ to $t_k$. Note, that in an optimal multicommodity flow the flow of every commodity is already cycle-free. If any of the flows for the particular commodities contain cycles, dropping these cycles would lead to a better solution.

Finally, in our model we meet the diversification requirements by bounding the flow of commodity $k$ through vertices and edges to $\lfloor \rho_k d_k \rfloor$, except at the vertices $s_k$, $t_k$. Since the flows are directed, we are able to determine how much flow of a particular commodity enters or leaves some vertex. Note that for all commodities, the flow of commodity $k$ entering $t_k$ utilizes $t_k$, while the flow of commodity $k$ leaving $s_k$ utilizes $s_k$. Measuring utilization this way reflects the fact that paths consume capacity at their terminals, too. At all other vertices (except $s_k$, $t_k$) the in-flow equals the out-flow of commodity $k$, so utilization can be measured from both in-flow and out-flow. Obviously, each commodity needs its own flow bounds.

## 3.2 Formulations

Here, we give some formulations of the model above as integer linear programs and as an integer convex program. We state the standard edge-flow based formulation that we will use to compare with the two formulations we extensively study. The former will be used as a benchmark for our approach. For the sake of completeness we state a path-flow based formulation as mentioned in section 2.4. Finally, we introduce the two formulations based on resource-directed decomposition. These will be the formulations to be studied further in Chapters 4 & 5.

### 3.2.1 Edge-flow based formulation

The most common idea for modelling multicommodity flow problems is a flow formulation with edge variables. We use variables $f^k_{(vw)}$, $f^k_{(wv)}$ to denote the flow of commodity $k \in \mathcal{Q}$ on edge $vw \in E$ in the direction from $v$ to $w$ or from $w$ to $v$, respectively. Note, that we distinguish between $f^k_{(vw)}$ and $f^k_{(wv)}$, although the edge $vw \in E$ is equal to the edge $wv \in E$. The variables $a_k$ describe the amount of demand $k$ that cannot be satisfied. The formulation reads:

$$\min \sum_{k=1}^{q}\Big(Ma_k + \sum_{(vw)\in E}\Big(f^k_{(vw)} + f^k_{(wv)}\Big)\Big), \tag{EFFa}$$

$$\sum_{k=1}^{q}\Big(f^k_{(vw)} + f^k_{(wv)}\Big) \le c_{vw}, \qquad \forall vw \in E, \tag{EFFb}$$

$$\sum_{\substack{k\in\mathcal{Q}\\ t_k\neq v}}\sum_{vw\in\delta(v)} f^k_{(vw)} + \sum_{\substack{k\in\mathcal{Q}\\ t_k=v}}\sum_{wv\in\delta(v)} f^k_{(wv)} \le c_v, \qquad \forall v \in V, \tag{EFFc}$$

$$\sum_{vt_k\in\delta(t_k)}\Big(f^k_{(vt_k)} - f^k_{(t_kv)}\Big) = d_k - a_k, \qquad \forall k \in \mathcal{Q}, \tag{EFFd}$$

$$\sum_{vw\in\delta(v)}\Big(f^k_{(vw)} - f^k_{(wv)}\Big) = 0, \qquad \forall k \in \mathcal{Q},\ v \in V\setminus\{s_k,t_k\}, \tag{EFFe}$$

$$f^k_{(vw)} + f^k_{(wv)} \le \lfloor \rho_k d_k \rfloor, \qquad \forall vw \in E,\ k \in \mathcal{Q}, \tag{EFFf}$$

$$\sum_{vw\in\delta(v)} f^k_{(vw)} \le \lfloor \rho_k d_k \rfloor, \qquad \forall k \in \mathcal{Q},\ v \in V\setminus\{s_k,t_k\}, \tag{EFFg}$$

$$f^k_{(vw)}, f^k_{(wv)} \in \mathbb{Z}_0^+, \qquad \forall vw \in E,\ k \in \mathcal{Q}, \tag{EFFh}$$

$$a_k \in \mathbb{Z}_0^+, \qquad \forall k \in \mathcal{Q}. \tag{EFFi}$$

The conditions (EFFb), (EFFc) refer to the capacity constraints for edges and vertices, respectively. Note that in (EFFc) we count the flows of all commodities that leave vertex $v$, except those flows that have target $v$. As mentioned in Section 3.1, the utilization of the flow of commodity $k$ at its target terminal $t_k$ is measured by the flow entering $t_k$. The flow leaving $v$ for commodities that have $v$ as its end terminal is added in the second part of the left-hand side of (EFFc).

In (EFFd) the flow conservation for the target terminal is expressed, for every commodity. It describes that the actual flow of commodity $k \in \mathcal{Q}$ entering the end terminal $t_k$ (without counting flow passing through). The demand of commodity $k$ that cannot be satisfied, described by the $a_k$, is related to this entering flow of $t_k$ by (EFFd).

The flow conservation constraints for all non-terminal vertices of commodity $k$ are given in (EFFe). Together with (EFFd), the flow conservation at terminal $s_k$ is assured: since no flow of commodity $k$ vanishes or arises at non-terminals, and exactly $d_k - a_k$ units of flow arise in $t_k$, the same amount has to arise at $s_k$.

Constraints (EFFf), (EFFg) establish the diversification restrictions at the edges and nodes, respectively. The restriction of the $a_k$ to integer values in (EFFi) is not necessary in general, as (EFFh) and (EFFd) imply that in an optimal solution the $a_k$ will be integral automatically. However, since we are interested in an integral number of paths for each commodity, and since all demands $d_k$ are integral, we claim that only an integral part of each demand is not routed.

In other multicommodity flow problems the demands arising at a common vertex can be aggregated, that means that they are merged into a single demand. This has the advantage that less demands have to be considered, while multiple sinks are established. This approach is not possible for our problem due to the diversification constraints. We could not decide which amount of flow of an aggregated demand crosses an edge or vertex.

An obvious disadvantage of formulation (EFF) is that in some optimal solution at most one of the variables in the pairs $f^k_{(vw)}, f^k_{(wv)}$ will be positive. If both variables of such a pair would be positive, they would form a cycle flow that can be removed. That is the reason why we examine some more formulations. We consider some more properties of formulation (EFF) in Section 3.3, when we compare it to another formulation given in Section 3.2.3.

### 3.2.2 Path-flow based formulation

Alternatively, the flow can be formulated with path variables. We state it here for completeness, as it was mentioned in Chapter 2. Let $\mathcal{P}_k$ be the set of all simple $(s_k, t_k)$-paths in G. Furthermore let $\mathcal{P} := \bigcup_{k=1}^{q} \mathcal{P}_k$. We introduce variables $f_P$ that describe the amount of flow sent along the path $P \in \mathcal{P}$. This formulation maps the routing problem in Section 2.2 more directly than the flow formulation in Section 3.2.1. In particular, we do not have to formulate conditions ensuring cycle-free flows explicitly. The formulation can rather be seen as a packing formulation than a flow formulation, since we pack paths into the supply graph.

Unfortunately, the number of paths between a terminal pair $s_k$, $t_k$ can be very large, and hence the number of variables, too. That's why this formulation is usually used for unsplittable flows, see Section 2.3, or if the supply graph admits only few paths between the terminal pairs. We have:

$$\min \sum_{k=1}^{q} \Big( M a_k + \sum_{P \in \mathcal{P}_k} l_P f_P \Big), \tag{PFFa}$$

$$\sum_{k=1}^{q} \sum_{P : P \ni e} f_P \leq c_e, \qquad \forall e \in E, \tag{PFFb}$$

$$\sum_{k=1}^{q} \sum_{P : P \ni v} f_P \leq c_v, \qquad \forall v \in V, \tag{PFFc}$$

$$\sum_{P \in \mathcal{P}_k} f_P = d_k - a_k, \qquad \forall k \in \mathcal{Q}, \tag{PFFd}$$

$$\sum_{P \in \mathcal{P}_k : P \ni e} f_P \leq \lfloor \rho_k d_k \rfloor, \qquad \forall e \in E,\ k \in \mathcal{Q}, \tag{PFFe}$$

$$\sum_{P \in \mathcal{P}_k : P \ni v} f_P \leq \lfloor \rho_k d_k \rfloor, \qquad \forall v \in V \setminus \{s_k, t_k\},\ k \in \mathcal{Q}, \tag{PFFf}$$

$$f_P \in \mathbb{Z}_0^+, \qquad \forall P \in \mathcal{P}, \tag{PFFg}$$

$$a_k \in \mathbb{Z}_0^+, \qquad \forall k \in \mathcal{Q}. \tag{PFFh}$$

Note that $l_P$ describes the length of path $P$, that is the number of edges the path uses. (PFFb),(PFFc) formulate the mutual capacity constraints for edges and vertices in $G$, respectively. The desired number of paths connecting the various terminal pairs are described in (PFFd). The diversification restrictions are stated in (PFFe) for edges and in (PFFf) for vertices. For the integrality constraints (PFFh) on the $a_k$, the same remark holds equally as for the formulation (EFF) in the previous section: integrality is forced by the other constraints.

### 3.2.3 Resource-directed formulation

Next we want to give a resource-directed formulation of the problem. We already pointed out that the value of a maximum flow respecting edge capacities as well as vertex capacities is exactly the capacity of a minimum general cut for a $s_k$-$t_k$-flow. To fulfil a demand of size $d_k - a_k$ we just have to ensure that every general $(s_k, t_k)$-cut has capacity at least $d_k - a_k$. This formulation can be seen as a counterpart of the edge-flow based formulation (EFF), where the flow conservation constraints are substituted by the cut constraints. In fact, the minimum cut problem and the maximum flow problem are dual to each other. We define variables $r_e^k$ that determine how much capacity is reserved on edge $e$ for demand $k$. Furthermore, variables $r_v^k$ refer to capacity for commodity $k$ at vertex $v$. The formulation reads:

$$\min \sum_{k=1}^{q} \left( M a_k + \sum_{e \in E} r_e^k \right), \tag{RDFa}$$

$$\sum_{k=1}^{q} r_e^k \leq c_e, \qquad \forall e \in E, \tag{RDFb}$$

$$\sum_{k=1}^{q} r_v^k \leq c_v, \qquad \forall v \in V, \tag{RDFc}$$

$$\sum_{e \in L} r_e^k + \sum_{v \in B} r_v^k \geq d_k - a_k, \qquad \forall k \in \mathcal{Q}, \text{ general } s_k\text{-}t_k\text{-cuts } \delta(S,B) = (L,B), \tag{RDFd}$$

$$r_e^k \leq \lfloor \rho_k d_k \rfloor, \qquad \forall e \in E, k \in \mathcal{Q}, \tag{RDFe}$$

$$r_v^k \leq \lfloor \rho_k d_k \rfloor, \qquad \forall v \in V \setminus \{s_k, t_k\}, k \in \mathcal{Q}, \tag{RDFf}$$

$$r_e^k \in \mathbb{Z}_0^+, \qquad \forall e \in E, k \in \mathcal{Q}, \tag{RDFg}$$

$$r_v^k \in \mathbb{Z}_0^+, \qquad \forall v \in V, k \in \mathcal{Q}, \tag{RDFh}$$

$$a_k \in \mathbb{Z}_0^+, \qquad \forall k \in \mathcal{Q}. \tag{RDFi}$$

As every commodity has its own set of resources, the problem decomposes into $q$ max-flow min-cost flow problems, one for each commodity. All of these subproblems are formulated as maximum min-cut problems by the cut constraints (RDFd).

The maximum min-cut subproblems are coupled by the capacity constraints (RDFb), (RDFc) for edges and vertices, respectively. The diversification requirements are met by the constraints (RDFe), (RDFf) for edges and vertices, respectively.

Using general cuts to formulate conditions for a feasible flow of a certain amount has the advantage that we need only one variable for every edge and commodity. Recall that in the edge-flow formulation (EFF) we introduced two variables for each edge and commodity. On the other hand, in formulation (RDF) we have variables for every vertex and commodity, too, but not in formulation (EFF). However, the supply graph $G$ has at least as many edges as vertices, except when $G$ is a tree. Hence, the formulation (RDF) has less variables than formulation (EFF) whenever $G$ contains more than $|V|$ edges.

### 3.2.4 Convex cost formulation

In the formulation above, the large class of general cut inequalities is used in order to fulfil all demands. As an alternative we can put the tractable flow into the objective. As in the formulation in the last section, we introduce variables $r_e^k$ for edges $e \in E$ and commodity $k \in \mathcal{Q}$, and variables $r_v^k$ for vertices $v \in V$. Additionally, variables $a_k$ describe the amount of commodity $k$ that can not be sent.

This leads to the following nonlinear program:

$$\min g(r), \tag{CPa}$$

$$\sum_{k=1}^{q} r_e^k \leq c_e, \qquad \forall e \in E, \tag{CPb}$$

$$\sum_{k=1}^{q} r_v^k \leq c_v, \qquad \forall v \in V, \tag{CPc}$$

$$r_e^k \leq \lfloor \rho_k d_k \rfloor, \qquad \forall e \in E, k \in \mathcal{Q}, \tag{CPd}$$

$$r_v^k \leq \lfloor \rho_k d_k \rfloor, \qquad \forall v \in V, k \in \mathcal{Q}, \tag{CPe}$$

$$r_e^k \in \mathbb{Z}_0^+, \qquad \forall e \in E, k \in \mathcal{Q}. \tag{CPf}$$

where the objective in (CPa) is:

$$g(r) := \sum_{k=1}^{q} g_k(r^k).$$

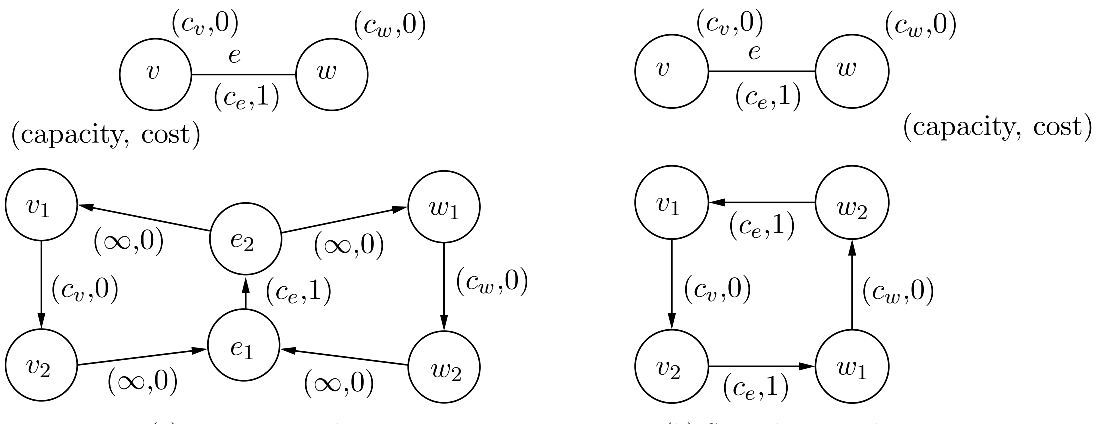
For each demand $k$, $g_k(r^k)$ describes the value of a min-cost flow with respect to edge capacities $r_e^k$ for all edges $e \in E$ and vertex capacities $r_v^k$ for all vertices $v \in V$.

To obtain a standard min-cost flow problem, that usually deals with undirected graphs and does not assume node capacities, we derive an extended graph $\vec{G} = (\vec{V}, \vec{E})$ from $G = (V, E)$. Two approaches are proposed in literature, see e.g., [1]. The first approach (see Figure 3.1(a)) places two vertices $e_1$ and $e_2$ for each edge $e \in E$, connected by an arc $(e_1, e_2)$. For each vertex $v \in V$, two vertices $v_1$ and $v_2$ are introduced. Let $e = vw \in E$ be some edge, then four more auxiliary arcs are put in: $(e_2, v_1)$, $(v_2, e_1)$, $(e_2, w_1)$, and $(w_2, e_1)$. The edge capacity $c_e$ is attached to the arc $(e_1, e_2)$, the vertex capacities $c_v$ are attached to the arc $(v_1, v_2)$. If costs are present, they are attached similarly. The auxiliary arcs usually have infinite capacity and no costs.

The second approach "expands" the vertices in the same way, i.e., some vertex $v$ refers to an arc $(v_1, v_2)$, with the vertex capacity as arc capacity attached. For some edge $e = vw \in E$, two arcs $(v_2, w_1)$ and $(w_2, v_1)$ are included, holding the same capacity as $e$, see Figure 3.1(b).

Both approaches model an undirected graph with node capacities as a directed graph with only edge capacities, so min-cost flow problems can be solved on these models by standard procedures. We will choose the latter approach, as it contains less arcs for a given graph $G$, compared to the first approach. Thus, we set $\vec{V} = \bigcup_{v \in V}\{v_1, v_2\}$, and $\vec{E} = \bigcup_{vw \in E}\{(v_2 w_1), (w_2 v_1)\} \cup \bigcup_{v \in V}\{(v_1, v_2)\}$.

To ease notation, we denote by $U(\vec{G}) \in \mathbb{R}^{\vec{V} \times \vec{E}}$ the vertex-edge incidence matrix of the graph $\vec{G}$. For $v \in \vec{V}$ and $(ij) \in \vec{E}$, the matrix contains at position $[v, (ij)]$ 1 if $v = i$, $-1$ if $v = j$, and 0 otherwise. The vector $D^k \in \mathbb{R}^{\vec{V}}$ contains the value $d_k$ at position $[s_{k_1}]$, the value $-d_k$ at position $[t_{k_2}]$, and 0 otherwise. The vector $A^k \in \mathbb{R}^{\vec{V}}$ contains a 1 at position $[t_{k_2}]$, $-1$ at position $[s_{k_1}]$, and 0 otherwise. Note, that by placing the source of commodity $k$ at $s_{k_1}$ in the extended graph $\vec{G}$, and the target at $t_{k_2}k$, all flow of commodity $k$ has to utilize the arcs $(s_{k_1} s_{k_2})$ and $(t_{k_1} t_{k_2})$. These two arcs refer to the terminals of commodity $k$ in the original graph $G$, and we assure that vertex utilization at the commodity's terminals in $G$ is modelled correctly in $\vec{G}$. The system $Uf + A^k a_k = D^k$ is equivalent to the system described by (EFFd) and (EFFe) for a fixed $k$, but with vertex capacities modelled on arcs.



(a) First approach (b) Second approach

Figure 3.1: Deriving a directed graph with only edge capacities from an undirected edge- and vertex-capacitated graph

We can state the definition of the min-cost flow values, i.e., the subproblems of (CP), using this notational simplification, as

$$
\begin{aligned}
g_k(r^k) = \quad & \min \sum_{e \in E} f_e + M a_k, && && (CP_k\text{a})\\
U(\vec{G})f + A^k a_k &= D^k, && && (CP_k\text{b})\\
f_{(v_2 w_1)}, f_{(w_2 v_1)} &\leq r^k_{vw}, && \forall\, vw \in E, && (CP_k\text{c})\\
f_{(v_1 v_2)} &\leq r^k_v, && \forall\, v \in V, && (CP_k\text{d})\\
f_{(v_2 w_1)}, f_{(w_2 v_1)} &\geq 0, && \forall\, vw \in E, && (CP_k\text{e})\\
f_{v_1 v_2} &\geq 0, && \forall\, v \in V && (CP_k\text{f})\\
a_k &\geq 0. && && (CP_k\text{g})
\end{aligned}
$$

The objective function $g(r)$ is a convex function, as we show in chapter 4. Thus, it can be solved with algorithms developed for nonlinear optimization. Optimization of convex functions is a special case of nonlinear optimization, and we describe a solution approach for (CP) in chapter 4.

## 3.3 Polyhedral investigation

We next investigate how the formulations in section 3.2 are related. For that, we introduce the polyhedra that underlie the formulations (EFF) and (RDF). We show how feasible points in these formulations are related to each other, and examine the dimension of the involved polyhedra.

Let us denote by $\tilde{E}$ the set of arcs used in formulation (EFF), i.e., $\tilde{E} = \bigcup_{vw \in E}\{(vw), (wv)\}$. First, we define the polyhedra

$$P_{EFF} = \text{conv}\{(f, a) \in \mathbb{R}^{\tilde{E}\mathcal{Q}} \times \mathbb{R}^{\mathcal{Q}} : (f, a) \text{ fulfils } (\text{EFF}b - \text{EFF}i)\} \tag{3.6}$$

and

$$P_{RDF} = \text{conv}\{(r, a) \in \mathbb{R}^{(E+V)\mathcal{Q}} \times \mathbb{R}^{\mathcal{Q}} : (r, a) \text{ fulfils } (\text{RDF}b - \text{RDF}i)\}. \tag{3.7}$$

We already mentioned that the number of variables in the description of $P_{EFF}$ is larger than the number of variables for $P_{RDF}$ if $|V| < |E|$, i.e., when the graph $G$ is not too thin. This fact is clearly an advantage of formulation (RDF). We next determine the dimensions of the two polyhedra to motivate further why the formulation (RDF) might be favourable.

**Lemma 3.1** *The dimension of $P_{EFF}$ is $q(2|E| - |V| + 1)$.*

**Proof.**

- The description of $P_{EFF}$ contains $q(|V| - 1)$ linearly independent equalities.

It is a well known fact that the edge-node incidence matrix $U(\vec{G})$ has row rank $|V|-1$, see [1] Theorem 11.10. Thus, for each $k \in \mathcal{Q}$, the constraints (EFFd), (EFFe) establish $|V|-1$ linearly independent equalities. Obviously, these sets of equalities are altogether linearly independent, since each set incorporates only variables of a single demand.

- There are $q(2|E| - |V| + 1) + 1$ affinely independent feasible points in $P_{EFF}$.

  Consider, for each $k \in \mathcal{Q}$, the setting $a_k = d_k - 1$. Furthermore, let $a_i = d_k$, and $f^i_{vw} = f^i_{wv} = 0$, for all $i \in \mathcal{Q} \setminus \{k\}$. Under these assumptions, to obtain a feasible point of $P_{EFF}$, we have to establish a single path for demand $k$. As already pointed out the matrix $U(\vec{G})$ has row rank of $|V| - 1$. As all other constraints like node constraints (EFFc) and diversification constraints (EFFf),(EFFg) are redundant, we yield $2|E| - |V| + 1$ linearly independent flow vectors for demand $k$, establishing the desired path. Note, that $U(\vec{G})$ is totally unimodular, see e.g., [69, Proposition 3.4]. Thus, the integrality constraints can be satisfied by the linearly independent flow vectors.

  As for all $k \in \mathcal{Q}$ such a setting can be obtained, and as the sets of linearly independent flow vectors are altogether linearly independent, this yields $q(2|E| - |V| + 1)$ linearly independent points in $P_{EFF}$. Together with the point $f^k_{vw} = f^k_{wv} = 0$ for all $vw \in E$, $k \in \mathcal{Q}$, $a_k = d_k$ for all $k \in \mathcal{Q}$, we get the desired $q(2|E| - |V| + 1) + 1$ affinely independent feasible points in $P_{EFF}$.

As we have $q(2|E| + 1)$ variables, the lemma follows. ∎

We next show that $P_{RDF}$ is full-dimensional.

**Lemma 3.2** *The dimension of $P_{RDF}$ is $q(|E|+|V|+1)$, that is, $P_{RDF}$ is full-dimensional.*

**Proof.** We proof the lemma straight forward by constructing $q(|E| + |V| + 1) + 1$ affinely independent feasible points of $P_{RDF}$.

For all demands $k \in \mathcal{Q}$ and all $\nu \in E \cup V$ we define a feasible point $p^{k,\nu}$ as follows: first we set $p^{k,\nu}[r^k_\nu] = 1$, and for all $\eta \in (E \cup V) \setminus \{\nu\}$ we set $p^{k,\nu}[r^k_\eta] = 0$. We denote by $p[r^i_\eta]$ the entry for the variable $r^i_\eta$ in the point $p$. For all other demands $i \neq k$ and all $\eta \in E \cup V$ we set $p^{k,\nu}[r^i_\eta] = 0$. We set $p^{k,\nu}[a_i] = d_i$ for all demands $i \in \mathcal{Q}$. Clearly, all points generated this way are feasible for (RDF), hence contained in $P_{RDF}$.

For every demand $k$ we define another feasible point $p^{k,\circ}$ quite similar: for all $\nu \in E \cup V set$ we set $p^{k,\circ}[r^k_\nu] = 1$, for all other demands $i \neq k$ we set $p^{k,\circ}[r^i_\nu] = 0$. The variables $a_i$ for not routed demands are set to $p^{k,\circ}[a_k] = d_k - 1$, and for all other demands $i \neq k$ we set $p^{k,\circ}[a_i] = d_i$.

Finally, we provide another feasible point $p^\bullet$. This point is given by $p^\bullet[r^i_\nu] = 0$ for all demands $i$ and all $\nu \in E \cup V$, and $p^\bullet[a_i] = d_i$.

All the feasible points described above are shown in the matrix (3.8) as row vectors.

$$
\begin{array}{|ccc|c|}
\hline
\begin{matrix}1 & & \\ & \ddots & \\ & & 1\end{matrix} & & & \begin{matrix} d_1 & \cdots & d_q\\ \vdots & & \vdots \\ d_1 & \cdots & d_q\end{matrix} \\
\hline
& \ddots & & \vdots \\
& & \begin{matrix}1 & & \\ & \ddots & \\ & & 1\end{matrix} & \begin{matrix} d_1 & \cdots & d_q\\ \vdots & & \vdots \\ d_1 & \cdots & d_q\end{matrix} \\
\hline
\mathbf{1}^T & & & \begin{matrix} d_1-1 & \cdots & d_q \\ & \ddots & \\ d_1 & \cdots & d_q-1\end{matrix}\\
& \ddots & & \\
& & \mathbf{1}^T & \\
\hline
& \mathbf{0}^T & & \begin{matrix} d_1 & \cdots & d_q\end{matrix}\\
\hline
\end{array}
\tag{3.8}
$$

To show that the points given above are affinely independent, we show that the $q(|E| + |V| + 1)$ points $p^{k,\nu} - p^{\bullet}$, $p^{k,\circ} - p^{\bullet}$ are linearly independent. These points are shown in the matrix (3.9) as row vectors.

$$
\begin{array}{|ccc|c|}
\hline
\begin{matrix}1 & & \\ & \ddots & \\ & & 1\end{matrix} & & & \\
& \ddots & & \\
& & \begin{matrix}1 & & \\ & \ddots & \\ & & 1\end{matrix} & \\
\hline
\mathbf{1}^T & & & \begin{matrix} -1 & & \\ & \ddots & \\ & & -1\end{matrix}\\
& \ddots & & \\
& & \mathbf{1}^T & \\
\hline
\end{array}
\tag{3.9}
$$

The row vectors are clearly linearly independent in matrix (3.9), since the matrix is quadratic and a lower triangle matrix with only nonzeros on the diagonal, hence not singular. ■

**Corollary 3.3** *From the last two lemmata it follows that if* $|E| \gtreqqless 2|V|$ $P_{EFF}$ *then* $\dim P_{EFF} \gtreqqless \dim P_{RDF}$.

Comparing the two polyhedra, we observe:

- Let $(f, a)$ be a feasible point of $P_{EFF}$. We construct a feasible point of $P_{RDF}$ from that as follows: first, let $r^k_{vw} = f^k_{v_2w_1} + f^k_{w_2v_1}$ for all $k \in \mathcal{Q}$ and all $vw \in E$. Let $r^k_v = \sum_{vw\in\delta(v)} f^k_{v_2w_1} + f^k_{w_2v_1}$. Then $(r, a)$ is feasible for $P_{RDF}$: since $(f, a)$ fulfils the edge capacity constraint (EFFb), $(r, a)$ fulfils the edge capacity constraint (RDFb).

By construction, the vertex capacity constraints (EFFc) for $(f, a)$ imply that $(r, a)$ fulfils the vertex capacity constraints (RDFc). The min-cut max-flow theorem 1.1 in its general form with vertex capacities applies to the flow $f^k$ for each commodity $k$: if such a flow $f^k$ established by (EFFd), (EFFe) admits sending $d_k - a_k$ units of flow from $s_k$ to $t_k$, every general $s_k$-$t_k$ cut has a capacity of at least $d_k - a_k$. Hence, $(r, a)$ satisfies the general cut restrictions (RDFd). Finally, $(r, a)$ meets the diversification constraints (RDFe), (RDFf), since $(f, a)$ fulfils the diversification constraints (EFFf), (EFFg).

- A reverse relation does not hold. Of course, given a feasible point $(r, a)$ of $P_{RDF}$, we can compute a feasible point $(f, a)$ by solving $q$ max flow problems with edge capacities $r^k_{vw}$ and node capacities $r^k_v$ on the graph $G$. Especially, we can yield the same values for the $a_k$. However, increasing some variable $r^k_{vw}$ or $r^k_v$ not in a minimal general cut for demand $k$ may lead to another feasible point of $P_{RDF}$, but the point $(f, a)$ obtained from that will stay the same. On the other hand, given edge and node resources $r^k_{vw}$ and $r^k_v$ as total flow values on edges and through nodes, possibly more than one directed flow can be obtained. E.g., circular (partial) flows can be reversed yielding the same total flow on all edges and nodes.

# Chapter 4

# Subgradient method

In this chapter we introduce some basic ideas of the subgradient method. We will state some known results, not in general as such but fitted to our aim. We outline several variants which are all suitable for a relaxation of our problem. Then we show how this method can be applied to our problem. Finally, we extend the method to produce heuristically integral solutions.

## 4.1 Theoretical framework

The subgradient method was first proposed by Shor [62]. It generalizes the idea of gradients of differentiable functions. The subgradient method aims to find optimal solutions of minimization problems with convex objective function, or maximization problems with concave objective function. It can be used only when the objective function is continuous and defined on a simply connected subset of a finite dimensional vector space. As an approximation approach, it only provides convergence to optimal solutions under certain conditions. Nevertheless, it has been shown to be valuable for various problems, often being superior to other approaches. Additionally, it is easy to implement in comparison to other solution techniques, for example a modern simplex algorithm. The subgradient method's first successful application was to solve the Lagrangian Dual of the Traveling Salesman Problem by Held and Karp [32, 33]. As a consequence, non-differentiable convex programming became a field of theoretical and practical interest on its own. In our context, the concepts presented here are only a starting point for this kind of algorithms.

In this section, we first mention some general ideas of subgradient methods, as developed for convex programs without side constraints. Afterwards we give some ideas from literature on how this can be extended to programs with additional constraints, namely linear inequalities, as this is the case in our model.

### 4.1.1 General scheme

The general idea of the subgradient method can be described as follows: suppose we are given a function defined on a proper ground set, that shall be minimized. Furthermore, let a starting point be given in the ground set. We aim to find a direction of decreasing function value. Furthermore, we have to determine a step length to move along the chosen direction. Iterating this process should end in a point that is (at least approximately) optimal for the problem. In the simplest case where the objective is a convex function to minimize, and this objective is defined on a whole vector space, every local minimum is a global minimum.

From analysis of differentiable function it is known that the gradient of a differentiable function points in the direction of local steepest ascent at every point where the gradient is evaluated. So it is a natural choice to move in the direction of the negative gradient. In the case of non-differentiability, where no gradient is available, we have to generalize the idea to subgradients, that still keeps the main properties needed for the optimization process.

A subgradient is defined as follows:

**Definition 4.1** *A* subgradient *at some point $x$ for a convex function $f : \mathbb{R}^n \to \mathbb{R}$ is a vector $\gamma(x)$ such that $f(\nu) \geq f(x) + \gamma(x)^T(\nu - x)$ for all $\nu \in \mathbb{R}^n$.*

Let $f$ be a convex function defined on $\mathbb{R}^n$. Then a *convex program* is to find a point in $\mathbb{R}^n$, if such one exists, that minimizes $f$. The subgradient method works as follows: provided a starting point $x_1 \in \mathbb{R}^n$, a sequence $\{x_k\}$ is computed according to

$$x_{k+1} = x_k - h_k \frac{\gamma(x_k)}{\|\gamma(x_k)\|} \quad \forall\, k \in \mathbb{N} \tag{4.1}$$

where $\gamma(\nu)$ is a subgradient of $f$ at the point $\nu$ and $\{h_k\}$ is a sequence of positive step lengths. Note that this is exactly the idea mentioned above: we choose the direction of the negative subgradient and move along this direction to get a new point. To obtain convergence of the sequence $\{x_k\}$ to a global minimum, the following theorem is essential, see [58]:

**Theorem 4.2 (Polyak)** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Let there be an arbitrary starting point $x_1 \in \mathbb{R}^m$, and a sequence $\{x_k\}$ defined by* (4.1)*. Let the sequence of step lengths be such*

$$\lim_{k\to\infty} h_k = 0,$$

$$\sum_{j=1}^{k} h_j \overset{k\to\infty}{\longrightarrow} \infty$$

*hold. Then*

$$\Phi_k := \min_{1\leq j\leq k} f(x_j) \overset{k\to\infty}{\longrightarrow} \min_{x\in\mathbb{R}^n} f(x).$$

For the step lengths many approaches are used, e.g.:

$$h_k = \frac{h}{k \ln k}, \qquad h_k = \frac{h}{k^\beta},\ 0 < \beta \leq 1, \qquad h_k = \frac{h}{k + C}, \tag{4.2}$$

where $h$ is a starting step length, while $\beta$ and $C$ are constants. These approaches differ in the speed of convergence of the $h_k$ to zero, and in the way the sum of the step lengths increases. The right choice of the step lengths is crucial in the convergence behaviour of a subgradient method, but the best choice heavily depends on the problem, the instance characteristics, the starting step length, and the initial point $x_1$. Although the step lengths in (4.2) assure convergence due to Theorem 4.2, they did not satisfy in practice. Usually the speed of convergence is too low. That's why in practice the step length $h$ is divided by a constant factor $\beta_1$ (often 2 is recommended) due to some heuristic. For example, if the best objective value reached so far has not been improved for $\beta_2$ iterations, $\beta_2$ being another positive constant, $h$ is lowered this way. Note that this setting can not guarantee convergence in general.

Finally, the step length can be chosen dependent only on the current point and subgradient. For that, a line search along $\gamma(x_k)$ is carried out to find (approximately) the step length leading to a new point that minimizes the objective function on the ray from $x_k$ along $\gamma(x_k)$. However, although this procedure assures convergence and can decrease the number of iterations needed, there is a trade-off between the decreased number of iterations and the increased expense in each iteration due to the multiple objective evaluation.

Another extension widely used is not to take only the subgradient at the current point to compute the next point, but to take the predecessor, too. To use the information of subgradients of former iterations is part of the wide field of bundle methods. These methods though reach beyond the scope of this work. For a detailed introduction to bundle methods see e.g., [35, 46]. However, a straight forward approach is to use a direction $g_k$ instead of $\gamma(x_k)$ in (4.1), defined as:

$$g_k = \gamma(x_k) + \sigma_k g_{k-1}, \qquad \sigma_k \in \mathbb{R}. \tag{4.3}$$

Here, $\sigma_k$ describes the influence of the preceding direction onto the new direction in the $k$th iteration. For that, $\sigma_k \geq 0$ should be set. Note that (4.1) describes the special case when $\sigma_k = 0$ for all $k \in \mathbb{N}$. There are several such subgradient adjusting rules that differ in the way $\sigma_k$ is set:

- *Crowder rule*

  Here, $\sigma_k$ is fixed to some constant value $< 1$. The rule was proposed by Crowder in [15].

- *Camerini-Fratta-Maffioli rule*

  This is a more sophisticated rule, motivated by the idea, that a vastly "jumping" direction should be smoothened. It was proposed in [12], and was supported by experimental experience in [14]. The rule states:

$$\sigma_k = \begin{cases} \tau \gamma(x_k)^T g_{k-1} / \|g_{k-1}\|^2, & \text{if } \gamma(x_k)^T g_{k-1} < 0, \\ 0, & \text{otherwise,} \end{cases}$$

where $\tau \in \mathbb{R}$ is a parameter. If $\tau$ is properly chosen in the interval $[0, 2]$, the direction $g_k$ is at least as good as $\gamma(x_k)$ in the following sense: in the important case if $\gamma(x_k)^T g_{k-1} < 0$, the preceding direction $g_{k-1}$ and the new subgradient $\gamma(x_k)$ form an angle $\varphi$ of more than $\pi/2$, since $\gamma(x_k)^T g_{k-1} = \|\gamma(x_k)\| \|g_{k-1}\| \cos \varphi$. Consider again (4.3), and applying the Camerini-Fratta-Maffioli rule for the case when $\gamma(x_k)^T g_{k-1} < 0$ yields:

$$g_k = \|\gamma(x_k)\| \Big( \frac{\gamma(x_k)}{\|\gamma(x_k)\|} - \tau \cos \varphi \frac{g_{k-1}}{\|g_{k-1}\|} \Big). \tag{4.4}$$

In that case, we observe that the direction of $g_k$ is chosen as an affine combination of the directions of $\gamma(x_k)$ and $g_{k-1}$. We further observe that:

$$\varphi \longrightarrow \frac{\pi}{2} \Rightarrow g_k \longrightarrow \gamma(x_k),$$

$$\varphi \longrightarrow \pi \Rightarrow g_k \longrightarrow \|\gamma(x_k)\| \Big( \frac{\gamma(x_k)}{\|\gamma(x_k)\|} + \tau \frac{g_{k-1}}{\|g_{k-1}\|} \Big).$$

This observation means that $\tau$ controls the influence of the direction of $g_{k-1}$ on the direction of $g_k$ if $\varphi$ is large. In any case the length of $g_k$ is mainly determined by the length of $\gamma(x_k)$. Hence, the Camerini-Fratta-Maffioli rule aims to prevent the subgradient method from "zigzagging" when succeeding subgradients form large angles. The authors proposed $\tau = 1.5$ as a reasonable choice.

- *Modified Camerini-Fratta-Maffioli rule*

  Here, the parameter $\tau$ is dynamically adjusted to $-\|\gamma(x_k)\| \|g_{k-1}\| / \gamma(x_k)^T g_{k-1}$, which arises from geometrical reasons. Thus, the influence parameter becomes

$$\sigma_k = \begin{cases} \|\gamma(x_k)\| / \|g_{k-1}\|, & \text{if } \gamma(x_k)^T g_{k-1} < 0, \\ 0, & \text{otherwise.} \end{cases}$$

In Chapter 6 we apply the rules on our problem IMCF-N and examine the impact of the different rules on the obtained solutions.

In the light of the preceding outline we mention two more step length schemes. Suppose we have a lower bound $\underline{f}$ of the optimal solution of the convex program. Crainic, Frangioni, and Gendron [14] proposed the following step lengths:

$$h_k = \lambda_k (f(x_k) - \underline{f}) / \gamma(x_k)^T g_k, \tag{4.5}$$

which can be seen as an extension of the step lengths proposed by Kennington and Shalaby [42], who used the formula

$$h_k = \lambda_k (f(x_k) - \underline{f}) / \gamma(x_k)^T \gamma(x_k), \tag{4.6}$$

and the simple setting $g_k = \gamma(x_k)$. In both step length schemes (4.5), (4.6) the sequence $\{\lambda_k\}$ assures convergence of the step lengths to zero. For that, the sequence $\{\lambda_k\}$ is chosen as a nonincreasing sequence of positive real numbers converging to zero. The authors of [14, 42] propose that, beginning with a given start value $\lambda_1$, $\lambda_k$ is divided by a constant factor $\beta_1$ every time the objective value $f(x_k)$ has not been improved for $\beta_2$ consecutive iterations. The authors of [34] divided $\lambda_k$ by $\beta_1 = 2$ every $\beta_2$ iterations, independently from the behaviour of the objective value. We apply the step length scheme (4.5) in our implementation of the subgradient method. The concrete setting of the constants $\beta_1$, $\beta_2$, and the initial prefactor $\lambda_1$ is described in Section 4.2.1.

### 4.1.2 Constrained programs

We extend this general subgradient schemes to problems with linear inequalities as side constraints, as this happens to be the case for our formulation (CP). First we formally state the problem, then we sketch the methods to deal with the constraints. These methods can roughly be classified as penalizing methods and projecting methods. As above, we restrict ourselves to methods that will be used in the run up.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. We define a *constrained convex program* as:

$$\begin{aligned} &\min f(x) \\ &g_i(x) \geq 0, \qquad\qquad \forall i \in [m] \end{aligned} \tag{CCONV}$$

Let the feasible region of (CCONV) be denoted by

$$X = \{x | x \in \mathbb{R}^n, g_i(x) \geq 0, \forall i \in [m]\}. \tag{4.8}$$

To ease the following considerations and to stay close to our problem, in the sequel we will assume that the constraint functions $g_i$ are affine functions, i.e. $g_i(x) = a_i^T x - b_i$, with $a_i \in \mathbb{R}^n, b_i \in \mathbb{R}$. We investigate how the subgradient method can be extended to deal with the additional conditions. We focus on penalty function, barrier functions, and exact penalty functions. The application of these methods within a subgradient method for our problem IMCF-N is described in Section 4.2. It should be mentioned that a variety of additional methods were suggested, see [5] for a survey.

#### Penalty functions

The general idea of penalty functions for constrained programs is to relax the constraints, but to "penalize" points outside the feasible region of the program. The penalty given to all infeasible points should be large enough to assure that the unconstrained penalized program has only optimal solutions that are feasible.

Given a problem (CCONV) with feasible region $X$, we define a *loss function* for this problem by

$$s(x) \quad \begin{cases} = 0, x \in X \\ > 0, x \notin X. \end{cases} \tag{4.9}$$

That means, that a loss function indicates the infeasible region where a penalty has to be provided. A typical setting for a loss function is

$$s(x) = \sum_{i=1}^{m} |\min\{0, g_i(x)\}|^\beta,$$

where $\beta$ is usually 1 or 2. The idea behind this example is that the loss function $s$ increases if the violation of some constraint increases. To choose a polynomial of low degree in $s$ is inspired by the fact that it is easy to compute and it grows not too fast (e.g., as an exponential function) with the violation of constraints. The latter fact is important to avoid numerical difficulties.

Let $\{\rho^k\}$ be a strictly decreasing sequence of positive numbers, called *guidance parameters*, and let $t$ be a real-valued *guidance function* such that $\rho^1 > \rho^2 > 0$ implies $t(\rho^2) > t(\rho^1) > 0$. That means that $t$ is positive and strictly decreasing for positive guidance parameters. Furthermore, let

$$\lim_{k\to\infty} \rho^k = 0 \tag{4.10}$$

$$\lim_{k\to\infty} t(\rho^k) = +\infty \tag{4.11}$$

Then $t(\rho^k)s(x)$ is called an *(exterior) penalty function*, and the *augmented objective function* is

$$F(x, p^k) = f(x) + t(\rho^k)s(x). \tag{4.12}$$

The penalty function is zero on the feasible set $X$ by (4.9), and positive on the infeasible set $\mathbb{R}^n \setminus X$ due to (4.9) and (4.10). We get a sequence of unconstrained problems, denoted by (CONV$_k$), that consist of finding a minimum of $F(x, \rho^k)$. The desired behaviour of penalty functions, i.e., leading to optimal solutions that are feasible and optimal for (CCONV), is provided by the following theorem, see [5], Theorem 12.2.:

**Theorem 4.3 (Avriel)** *Let the feasible set $X$ of* (CCONV) *as defined in* (4.8) *be non-empty. Suppose there exists an $\epsilon > 0$ such that the set*

$$X^\epsilon = \{x | x \in \mathbb{R}^n, g_i(x) \geq -\epsilon, \forall i \in [m]\}$$

*is compact. Also suppose that the $F(x, \rho^k)$ as given in* (4.12) *attain their unconstrained minima on $\mathbb{R}^n$ for all $k$. Then there exists a convergent subsequence $\{x^{k_i *}\}$ of the optimal solutions to* (CONV$_k$)*, and the limit of any such convergent subsequence is optimal for* (CCONV).

It should be noted that the assumption in Theorem 4.3, that the $F(x, \rho^k)$ attain their minima is a rather strong condition. Among other conditions to fulfil this requirement it is sufficient that all the $F(x, \rho^k)$ are convex and $X^\epsilon$ is compact for some $\epsilon > 0$, as proven by Zangwill [70]. Convexity of $F(x, \rho^k)$ can be established by convexity of the penalty functions $t(\rho^k)s(x)$, since the objective function $f$ is convex. In addition, as we deal with affine functions $g_i$ only, boundedness of $X^\epsilon$ implies compactness of $X^\epsilon$. This boundedness can not be established a priori, but only dependent on the specific problem. For example, if $X^\epsilon$ is not bounded for any $\epsilon > 0$, one could modify the program (CCONV) by introducing new constraints that do not exclude all optimal solutions.

**Barrier functions**

Penalty functions penalize leaving the feasible region $X$ as described in the section above. Another approach is to keep the sequence of generated points away from the boundary of $X$. For that, feasible points are increasingly penalized when approaching the boundary. This idea is provided by so-called *barrier functions*.

First, we assume some regularity conditions on the program (CCONV), that read:

1. Let the interior of the $X$, denoted by $X^0$, be nonempty.
2. There is a point $x^0 \in X$ with $f(x^0) = \alpha^0$ such that the *level set* $S(f, \alpha^0) = \{x | x \in \mathbb{R}^n, f(x) = \alpha^0\}$ fulfils that $S(f, \alpha^0) \cap X$ is a compact set. (4.13)

Let $s$ be a real-valued *interior loss function* of $x \in \mathbb{R}^n$ such that $s$ is continuous on $X^0$. Furthermore, for every sequence $\{x^k\}$ in $X^0$ converging to some boundary point of $X$, an interior loss function $s$ has to satisfy $\lim_{k\to\infty} s(x^k) = +\infty$. Also, let $t$ be a real-valued *guidance function* of $\rho \in \mathbb{R}$ such that

$$\begin{aligned} \rho^1 > \rho^2 > 0 \quad &\Longrightarrow \quad t(\rho^1) > t(\rho^2) > 0, \\ \lim_{k\to\infty} \rho^k = 0 \quad &\Longrightarrow \quad \lim_{k\to\infty} t(\rho^k) = 0. \end{aligned}$$

Similar to the penalty function approach, $\rho_k \in \mathbb{R}$ are *guidance parameters*. The function $t(\rho^k)s(x)$ is a *barrier function*, sometimes also called interior penalty function. The augmented objective function obtained is

$$G(x, \rho^k) = f(x) + t(\rho^k)s(x). \tag{4.14}$$

Thus, we again obtain a sequence of unconstrained programs, denoted by (CONV$_k$), that consist of finding a minimum of $G(x, \rho^k)$, without side constraints. For this barrier-type programs a similar convergence result holds as for the penalty-type programs, see [5], Theorem 12.3.:

**Theorem 4.4 (Avriel)** *Let a problem* (CCONV) *be given with feasible set $X$ as defined in* (4.8). *Assume the regularity conditions* (4.13) *are satisfied and the loss function $s$ used in the augmented objective function are positive in $X^0$. Suppose that the $G(x, \rho^k)$ attain their unconstrained minima in $X^0$ for all $k$. If $\{\rho^k\}$ is a strictly decreasing sequence of positive numbers converging to zero, then there exists a convergent subsequence $\{x^{k_i *}\}$ of the optimal solutions to* (CONV$_k$)*, and the limit of any such convergent subsequence is optimal for* (CCONV).

Due to [5], a sufficient condition that all the $G(x, \rho^k)$ attain their minima in $X^0$ is that $X$ is bounded and $X^0$ is not empty. Note that this holds only in the special case of linear inequalities as side constraints.

### Exact penalty functions

Given a problem (CCONV), the solution approaches introduced in the last two sections construct sequences of unconstrained programs. The convergence Theorems 4.3, 4.4 provide convergence of the optimal solutions of the unconstrained programs (CONV$_k$) to an optimal solution of (CCONV). So we have to solve a sequence of convex problems, at least approximately. Hence, it might be desirable that the minimum of just one unconstrained augmented objective function yields a minimum of the original problem (CCONV). Actually, such an approach is known by means of special penalty functions. We define the loss function by $s(x) = \sum_{i=1}^{m} |\min\{0, g_i(x)\}|$, and we choose $r(\rho) = \frac{1}{\rho}$. This results in the unconstrained program

$$\min F(x, \rho) = \min f(x) - \frac{1}{\rho} \sum_{i=1}^{m} \min\{0, g_i(x)\}. \tag{4.15}$$

For the special augmented objective function in (4.15) we can state the following theorem, see [5], Theorem 12.5:

**Theorem 4.5 (Avriel)** *Let a constrained convex program* (CCONV) *be given with feasible set $X$ as defined in* (4.8). *Assume that the interior $X^0$ of the feasible set is nonempty, and that there is an optimal solution to* (CCONV), *attained at some point $x^* \in X$. Then there exists a $\rho^* > 0$ such that, for all $\rho^* \geq \rho > 0$, the unconstrained minimum of $F(x, \rho)$ as defined in* (4.15) *coincides with an optimal solution $x^*$ of* (CCONV).

As $\rho^*$ usually depends on the optimal solution of the original problem (CCONV), we need to estimate $\rho^*$. The proof of Theorem 4.5 yields the following construction for a $\rho^*$: let $\hat{x} \in X^0$; define

$$\alpha_1 = \min_{i \in [m]} g_i(\hat{x}) > 0,$$
$$\alpha_2 = f(\hat{x}) - f(x^*) \geq 0.$$

Then for arbitrary $\epsilon > 0$ it is proven in [5] that

$$\rho^* = \frac{\alpha_1}{\alpha_2 + \epsilon} > 0$$

is such a $\rho^*$ whose existence is proven in Theorem 4.5. To estimate $\rho^*$ suppose we have a lower bound $\underline{f}$ of (CCONV). Then we are able to compute

$$\hat{\alpha}_2 = f(\hat{x}) - \underline{f} \geq \alpha_2,$$
$$\hat{\rho} = \frac{\alpha_1}{\hat{\alpha}_2 + \epsilon} \leq \rho^*.$$

The Theorem 4.5 establishes that every $\rho \leq \rho^*$ yields an augmented objective function with an optimum at some point optimal for (CCONV), too. Hence, we use $\hat{\rho}$ instead of $\rho^*$, because $\hat{\rho}$ can be computed without knowledge of the optimal value of (CCONV).

**Projecting methods**

In the preceding approaches the constraints of (CCONV) are dualized, i.e. their violation is penalized. As a variant, we can consider constraints directly in the generation of directions and points. For that, an infeasible point generated by the subgradient method could be projected onto the feasible region. That means, we construct a feasible point with least distance to our current point. Taking Euclidean metric as the distance measure, this leads to a quadratic program. Hence, a projection onto the feasible set can be a difficult convex program itself. Whether this projection can be computed easily depends very much on the structure of the feasible set. In Section 4.2.2 we show how projection is applied for our formulation (RDF).

Additionally, the direction itself can be projected. Suppose a point $x^k$ generated during the iteration fulfils a constraint with equality, i.e. $a_i^T x^k = b_i$. Such a constraint will be called *active constraint*. Further, let $g_k$ be the direction generated at this point. If $a_i^T g_k < 0$, then the next point will be infeasible, as the direction points to the halfspace described by $a_i^T x \leq b_i$. Thus, the direction can be projected onto the affine space defined by the hyperplanes generated by the active constraints. However, this can not ensure that the next point will be feasible, as the step length will be chosen independently. In that case, we can either search along the chosen direction for feasible points, then the subgradient method becomes similar to the simplex method. As an alternative, we can apply any of the other rules dealing with side constraints, i.e., projecting and penalizing methods.

## 4.2 Application to IMCF-N

The subgradient procedure is suitable to approximate our problem using a relaxation of formulation (CP), see Chapter 3. In this section, we first introduce this relaxation. Next, we show that the objective $g(r)$ in (CPa) is convex. Then we determine appropriate subgradients, following the proofs given in [1, 34, 42] for directed graphs.

Afterwards we outline, how the different techniques to deal with constrained programs can be applied to our problem. As we focused on linear inequalities as constraints, we finally derive some heuristic methods to obtain integral solutions.

As outlined in Section 4.1, the subgradient is capable of minimizing convex functions over connected sets, and many results presented in Section 4.1.2 hold only for convex sets. So we can apply the subgradient method on (CP) only when the integrality constraints in (CPf) are relaxed. Let the relaxed feasible region of the master problem (CP) be

$$R = \{r \in \mathbb{R}^{EV} : r \text{ fulfils } (\text{CP}b - \text{CP}e)\}. \tag{4.16}$$

Then, the relaxation of (CP) considered in the sequel reads:

$$\min g(r), \tag{4.17a}$$

$$r \in R. \tag{4.17b}$$

First of all, consider the subproblems $(CP_k)$ as linear programs. For all $v \in \vec{V}$, let there be dual variables $\tau_v^k$ associated with the flow conservation constraints $(CP_k\text{b})$, $\pi_{(vw)}^k$ for the arc bounds ($CP_k$c), and $\pi_v^k$ for the "vertex" bounds ($CP_k$d), for each individual $k \in \mathcal{Q}$. Actually, the latter ones are arc bounds, too, but these arcs refer to vertices in the original graph $G$. Additionally, we define $\gamma_{vw}^k = \pi_{(vw)}^k + \pi_{(wv)}^k$, $\gamma_v^k = \pi_v^k$ as the *condensed duals* to translate the extended graph setting in $(CP_k)$ to the master problem (4.17), where undirected edges are considered.

By duality theory, the dual programs for $(CP_k)$ attain the same optimal value as the primal programs, for each $k \in \mathcal{Q}$, respectively. These dual programs read:

$$g_k(r^k) = \max {\tau^k}^T D^k + \sum_{vw \in E} \left( \left( \pi_{(vw)}^k + \pi_{(wv)}^k \right) r_{vw}^k \right) + \sum_{v \in V} \pi_v^k r_v^k \tag{4.18a}$$

$${\tau^k}^T U(\vec{G}) + {\pi^k}^T \leq (\mathbf{1}^T, \mathbf{0}^T) \tag{4.18b}$$

$${\tau^k}^T \leq M {A^k}^T \tag{4.18c}$$

$${\pi^k}^T \leq \mathbf{0}^T \tag{4.18d}$$

For short notation, we write $(\tau, \pi) \in P_{CP_k}$ if $(\tau^k, \pi^k)$ satisfies (4.18b)–(4.18d) for some demand $k$. An essential basis to apply the subgradient method is the fact that the objective function is convex. For the objective function $g$ of formulation (4.17) the next lemma establishes this fact. We follow the proofs given in [34], [42] for directed graphs.

**Lemma 4.6** *$g(r)$ is convex over $R$.*

**Proof.** Let $r, \hat{r} \in R$ be two resource vectors and a scalar $\alpha$ such that $0 \leq \alpha \leq 1$. Then

$$\begin{aligned}
g(\alpha r + (1-\alpha)\hat{r}) = \sum_{k=1}^{q} \Bigg( \max \Bigg\{ & {\tau^k}^T D^k + \sum_{vw \in E} \left( \left( \pi_{(vw)}^k + \pi_{(wv)}^k \right) \left( \alpha r_{vw}^k + (1-\alpha) \hat{r}_{vw}^k \right) \right) \\
& + \sum_{v \in V} \pi_v^k (\alpha r_v^k + (1-\alpha)\hat{r}_v^k) : (\tau^k, \pi^k) \in P_{CP_k} \Bigg\} \Bigg), \\
= \sum_{k=1}^{q} \Bigg( \max \Bigg\{ & \alpha \left( {\tau^k}^T D^k + \sum_{vw \in E} \left( \pi_{(vw)}^k + \pi_{(wv)}^k \right) r_{vw}^k + \sum_{v \in V} \pi_v^k r_v^k \right) \\
& + (1-\alpha) \left( {\tau^k}^T D^k + \sum_{vw \in E} \left( \pi_{(vw)}^k + \pi_{(wv)}^k \right) \hat{r}_{vw}^k + \sum_{v \in V} \pi_v^k \hat{r}_v^k \right) \\
& : (\tau^k, \pi^k) \in P_{CP_k} \Bigg\} \Bigg).
\end{aligned}$$

Decomposing the objectives and maximizing each addendum separately yields

$$\begin{aligned}
g(\alpha r + (1-\alpha)\hat{r}) \leq \alpha \sum_{k=1}^{q} \Bigg( & \max\Bigg\{ {\tau^k}^T D^k + \sum_{vw \in E} \left( \pi^k_{(vw)} + \pi^k_{(wv)} \right) r^k_{vw} + \sum_{v \in V} \pi^k_v r^k_v \\
& : (\tau^k, \pi^k) \in P_{CP_k} \Bigg\} \Bigg) \\
+ (1-\alpha) \sum_{k=1}^{q} \Bigg( & \max\Bigg\{ {\tau^k}^T D^k + \sum_{vw \in E} \left( \pi^k_{(vw)} + \pi^k_{(wv)} \right) \hat{r}^k_{vw} + \sum_{v \in V} \pi^k_v \hat{r}^k_v \\
& : (\tau^k, \pi^k) \in P_{CP_k} \Bigg\} \Bigg), \\
= \alpha g(r) + (1-\alpha) g(\hat{r}).
\end{aligned}$$

Since $r$, $\hat{r}$ are chosen arbitrary as well as $\alpha$, the definition of a convex function is satisfied. ■

The proof of the lemma already contains the main idea for a subgradient of $g$, as the solution of the dual programs yield lower bounds on the objective value of the intermediate point $\alpha r + (1-\alpha)\hat{r}$. So we can derive a subgradient for $g$ straight forward.

**Lemma 4.7** *Let $\hat{r} \geq 0$ be a resource setting, and let $\hat{\tau}^k$, $\hat{\pi}^k$ be optimal dual variables for the subproblems $(CP_k)$, for each $k \in \mathcal{Q}$. Then $(\hat{\gamma}^1, \ldots, \hat{\gamma}^q)$ is a subgradient of $g(r)$ at $\hat{r}$.*

**Proof.** Let $r \geq 0$ be any other resource setting with optimal dual variables $\tau^k$, $\pi^k$ for each subproblem. Then we get

$$\begin{aligned}
g(r) - g(\hat{r}) &= \sum_{k=1}^{q} g_k(r^k) - g_k(\hat{r}^k) \\
&= \sum_{k=1}^{q} \Bigg( {\tau^k}^T D^k + \sum_{vw \in E} \left( \left( \pi^k_{(vw)} + \pi^k_{(wv)} \right) r^k_{vw} \right) + \sum_{v \in V} \pi^k_v r^k_v \\
&\qquad - {\hat{\tau}^k}{}^T D^k - \sum_{vw \in E} \left( \left( \hat{\pi}^k_{(vw)} + \hat{\pi}^k_{(wv)} \right) \hat{r}^k_{vw} \right) - \sum_{v \in V} \hat{\pi}^k_v \hat{r}^k_v \Bigg)
\end{aligned}$$

Provided $r^k$, note that $(\tau^k, \pi^k)$ is optimal for the $k$th subproblem's dual. Moreover, these dual programs are maximization problems. We conclude

$$\begin{aligned}
&\geq \sum_{k=1}^{q}\Big(\hat{\tau}^{k^T} D^k + \sum_{vw\in E}\left(\left(\hat{\pi}^k_{(vw)} + \hat{\pi}^k_{(wv)}\right) r^k_{vw}\right) + \sum_{v\in V}\hat{\pi}^k_v r^k_v \\
&\qquad - \hat{\tau}^{k^T} D^k - \sum_{vw\in E}\left(\left(\hat{\pi}^k_{(vw)} + \hat{\pi}^k_{(wv)}\right) \hat{r}^k_{vw}\right) - \sum_{v\in V}\hat{\pi}^k_v \hat{r}^k_v\Big) \\
&= \sum_{k=1}^{q}\left(\sum_{vw\in E}\left(\hat{\pi}^k_{(vw)} + \hat{\pi}^k_{(wv)}\right)\left(r^k_{vw} - \hat{r}^k_{vw}\right) + \sum_{v\in V}\hat{\pi}^k_v\left(r^k_v - \hat{r}^k_v\right)\right) \\
&= \sum_{k=1}^{q}\left(\sum_{vw\in E}\hat{\gamma}^k_{vw}\left(r^k_{vw} - \hat{r}^k_{vw}\right) + \sum_{v\in V}\hat{\gamma}^k_v\left(r^k_v - \hat{r}^k_v\right)\right) \\
&= \sum_{k=1}^{q}\left(\hat{\gamma}^T_k\left(r^k - \hat{r}^k\right)\right) \\
&= \hat{\gamma}^T\left(r - \hat{r}\right).
\end{aligned}$$

Thus $\hat{\gamma}$ is a subgradient at $\hat{r}$ by Definition 4.1. ■

As the essential foundations are given by lemma 4.6, 4.7, we focus in the following on how the solution techniques outlined in Section 4.1 can be applied.

### 4.2.1 Step length selection

We use in our subgradient method the stepsize scheme (4.5). For that, we have to determine the sequence $\{\lambda_k\}$ used therein. The initial prefactor $\lambda_1$ is chosen experimentally, since no rule suitable for all instances is known.

First, we determine a lower bound on the objective function $g$. This is needed to apply the step length scheme proposed in (4.6). The simplest way to do this is to relax the condition that the demands are distinguishable, i.e., that the commodities have their own associated terminals. By this, the multicommodity flow problem reduces to a min-cost flow problem with $c_e$ as edge capacity, for all edges $e$, and $c_v$ as vertex capacity, for all vertices $v$. We set

$$\delta_v = \sum_{i:s_i=v} d_i - \sum_{i:t_i=v} d_i, \quad \forall v \in V,$$

and use $\delta$ as a demand vector. The solution value of this flow will be a lower bound $\underline{g}$ on the objective $g$ of (4.17). This bound can be very poor, it can even be zero (even $\delta = \mathbf{0}$ is possible), but a better one would require additional computational effort. Moreover, the quality of the lower bound is not important for the effectiveness of the stepsize scheme.

In the following we describe how to handle the constraints in (4.17). We outline a projection method as described next, and penalizing methods, that are outlined afterwards.

### 4.2.2 Projection method

To deal with the capacity constraints (CPb), (CPc), the diversification constraints (CPd), (CPe), and the nonnegativity constraints (CPf), Held, Crowder, and Wolfe [34] proposed a method that projects a possibly infeasible point generated by the subgradient procedure onto the feasible region. Suppose the point generated is $\hat{r}$. We want to find the feasible point $r$ with least Euclidean distance to $R$. This can be done by solving the following program:

$$\min \frac{1}{2}\sum_{i=1}^{q}\left(\sum_{e\in E}(r_e^i-\hat{r}_e^i)^2+\sum_{v\in V}(r_v^i-\hat{r}_v^i)^2\right), \tag{4.19a}$$

$$\sum_{i=1}^{q} r_e^i \le c_e, \qquad \forall e\in E, \tag{4.19b}$$

$$\sum_{i=1}^{q} r_v^i \le c_v, \qquad \forall v\in V, \tag{4.19c}$$

$$r_e^i,\ r_v^i \le \lfloor \rho_i d_i \rfloor, \qquad \forall e\in E,\ v\in V\setminus\{s_i,t_i\},\ i\in\mathcal{Q}, \tag{4.19d}$$

$$r_v^i \le d_i, \qquad \forall v\in\{s_i,t_i\},\ i\in\mathcal{Q}, \tag{4.19e}$$

$$r \ge \mathbf{0}, \tag{4.19f}$$

where the restrictions (4.19b)–(4.19e) are just a restatement of $r\in R$. The factor $\frac{1}{2}$ in the objective is just to ease the following consideration, but does not change the problem's solution. We observe that this quadratic programming problem decomposes into $|E|+|V|$ subproblems, one for each edge and each vertex. Each of these subproblems is again a quadratic programming problem, all having the same structure.

We next consider such a subproblem for some $\nu\in E\cup V$. To ease notation, let $z_i=\hat{r}_\nu^i$ for all $i\in\mathcal{Q}$ the partial given point, and $x_i=r_\nu^i$ the partial solution of (4.19) we look for. Let $u_i$ be the upper bound of $r_i$ as stated in (4.19d) or (4.19e), and $c$ the capacity of $\nu$ as stated in (4.19b) or (4.19c). Then all the subproblems take the form

$$\min\sum_{i=1}^{q}\frac{1}{2}(z_i-x_i)^2, \tag{4.20a}$$

$$\sum_{i=1}^{q} x_i \le c, \tag{4.20b}$$

$$x_i \le u_i, \qquad \forall i\in\mathcal{Q}. \tag{4.20c}$$

$$x \ge \mathbf{0}, \tag{4.20d}$$

In [34], the multicommodity flow problem was considered without individual bounds on the edges for each commodity. Hence, without constraints (4.20c), the constraints in (4.20) describe a simplex, a rather simple structure. In our problem, the bounds (4.20c) arise due to diversification restrictions. So, we have to extend the solution approach for (4.20), taking these upper bounds into account. However, the main idea is the same, and is based on Kuhn-Tucker theory for general programming problems. In the next paragraphs we give a construction for a solution of (4.20).

First we choose Karush-Kuhn-Tucker(KKT)–multipliers $\mu$ for (4.20b), $\lambda$ for (4.20d) and $\kappa$ for (4.20c). We know (see e.g., [5]) that is necessary and sufficient for $\hat{x}$ to be a solution to (4.20), if $(\hat{x}, \lambda, \mu, \kappa)$ fulfil:

$$\hat{x}_i - z_i + \kappa_i + \mu - \lambda_i = 0, \qquad \forall i \in [q] \tag{4.21a}$$

$$\mu\Big(\sum_{i=1}^{q} \hat{x}_i - c\Big) = 0, \tag{4.21b}$$

$$\kappa_i(\hat{x}_i - u_i) = 0, \qquad \forall i \in [q] \tag{4.21c}$$

$$\lambda_i \hat{x}_i = 0, \qquad \forall i \in [q] \tag{4.21d}$$

$$\kappa, \lambda \geq 0, \tag{4.21e}$$

$$\mu \geq 0 \tag{4.21f}$$

$$\sum_{i=1}^{q} \hat{x}_i \leq c, \tag{4.21g}$$

$$\hat{x} \geq \mathbf{0}, \tag{4.21h}$$

$$\hat{x} \leq u. \tag{4.21i}$$

As (4.20) surely has an optimal solution, the above system (4.21) has a solution, too. We propose a method, how such a solution can be constructed. For that, we set:

$$\hat{x}_i(\mu) = \min\{u_i, \max\{z_i - \mu, 0\}\}, \tag{4.22a}$$

$$\kappa_i(\mu) = \max\{z_i - \mu - u_i, 0\}, \tag{4.22b}$$

$$\lambda_i(\mu) = \max\{-(z_i - \mu), 0\}. \tag{4.22c}$$

We can express (4.22) in more detail if we distinguish between the three cases how $z_i - \mu$ is related to 0 and $u_i$:

$$\begin{array}{llll} z_i - \mu \geq u_i \Rightarrow & \hat{x}_i(\mu) = u_i, & \kappa_i(\mu) = z_i - \mu - u_i, & \lambda_i(\mu) = 0 \\ u_i \geq z_i - \mu \geq 0 \Rightarrow & \hat{x}_i(\mu) = z_i - \mu, & \kappa_i(\mu) = 0, & \lambda_i(\mu) = 0 \\ 0 \geq z_i - \mu \Rightarrow & \hat{x}_i(\mu) = 0, & \kappa_i(\mu) = 0, & \lambda_i(\mu) = \mu - z_i \end{array}$$

immediately satisfying (4.21a), (4.21c), (4.21d), (4.21e), (4.21h), (4.21i). Now, the crucial conditions (4.21b) and (4.21g) will be realized as follows. (4.21b) requires that either

$$\sum_{i=1}^{q} \hat{x}_i(\mu) - c = 0 \tag{4.23a}$$

$$\text{or } \mu = 0 \tag{4.23b}$$

We have to check whether one of the cases in (4.23) can be established, and whether (4.21g) holds.

- The case (4.23a)

  We immediately satisfy (4.21g), and all that is left for finding a solution of (4.21) is finding a $\mu$ such that (4.23a). Examining the sum in (4.21g) we get:

$$\sum_{i=1}^{q} \hat{x}_i(\mu) = \sum_{i:z_i \geq \mu > z_i - u_i} (z_i - \mu) + \sum_{i:z_i - u_i \geq \mu} u_i. \tag{4.24}$$

  We define how often $\mu$ occurs in the right hand side of (4.24) by

$$r(\mu) := |\{i : z_i - u_i < \mu \leq z_i\}|. \tag{4.25}$$

  Using (4.24), the condition (4.23a) can be written as

$$\mu \cdot r(\mu) = \sum_{i:z_i \geq \mu > z_i - u_i} z_i + \sum_{i:z_i - u_i \geq \mu} u_i - c \tag{4.26}$$

  By (4.26) we have reduced (4.21b) to an equation for $\mu$ only. The crucial breakpoint values $b_i$, $i \in [2q]$, for $\mu$ are the components of $z$ and $z - u$. We sort these $2q$ values non-increasingly, and call the values $z_k$ *entry points* (as we enter the interval $(z_k - u_k, z_k]$ from above) and the values $z_k - u_k$ *exit points* (as we leave the interval). While scanning through the breakpoints, we check whether we fulfil (4.26), that is, whether we can fulfil for some $k \in [2q - 1]$:

$$\mu \cdot r(b_k) = \sum_{i:z_i \geq b_k > z_i - u_i} z_i + \sum_{i:z_i - u_i \geq b_k} u_i - c \tag{4.27a}$$

$$b_k \geq \mu > b_{k+1} \tag{4.27b}$$

$$\mu \geq 0 \tag{4.27c}$$

  We consider the case when none of the breakpoints $b_k$, $k \in [2q - 1]$, will fulfil the conditions (4.27). If the last breakpoint $\mu = b_{2q}$ fulfils (4.27a), (4.27c), we have reached our aim. Since $b_{2q} = z_j - u_j$ for some $j$, we have $r(b_{2q}) = 0$. In that case, (4.27a) reduces to the condition $\sum_{i=1}^{q} u_i = c$. For $\mu = b_{2q}$ being a solution of (4.26), we have to check whether $b_{2q} \geq 0$ and $\sum_{i=1}^{q} u_i = c$. If so, $\mu = b_{2q}$ is a solution of (4.26). By application of (4.22), we have a solution of (4.21).

- The case (4.23b)

  If none of the breakpoints $b_k$ fulfils the requirements for the case(4.23a), $\mu$ must be zero. We immediately satisfy (4.21b). The remaining question is, whether (4.21g) holds in this case. Rewriting (4.22a), we have that

$$\hat{x}_i(\mu) = \begin{cases} u_i & \mu \leq z_i - u_i, \\ z_i - \mu & z_i - u_i < \mu \leq z_i, \\ 0 & z_i < \mu, \end{cases} \tag{4.28}$$

  are continuous functions. Thus, $\sum_{i=1}^{q} \hat{x}(\mu)$ is continuous as well. Moreover, $\sum_{i=1}^{q} \hat{x}_i(b_1) - c = -c$ and $\sum_{i=1}^{q} \hat{x}_i(b_{2q}) - c = \sum_{i=1}^{q} u_i - c$. Hence, if $\sum_{i=1}^{q} u_i \geq c$, there will be a solution to (4.27) with $b_1 \leq \mu \leq b_q$ by the mean-value theorem. This solution is found in the case (4.23a). Otherwise, if $\sum_{i=1}^{q} u_i < c$, we get $\sum_{i=1}^{q} \hat{x}(0) \leq \sum_{i=1}^{q} u_i < c$, fulfilling (4.21g).

Finally, this leads to the Algorithm 1. In the sequel we will refer to it as PROJ.

**Algorithm 1** Projection with diversification

```
$S \leftarrow -c + b_1$ (the right hand side of (4.26))
$r \leftarrow 1$ (the number of intervals hit)
$i \leftarrow 1$ (the index of the current breakpoint)
while $\neg(i = 2q \wedge b_i > b_{i+1} \wedge ((r = 0 \wedge S = 0 \wedge b_i \geq 0) \vee (b_i \geq \frac{S}{r} > b_{i+1} \wedge \frac{S}{r} \geq 0)))$ do
  if $b_{i+1}$ is an entry point (say $b_{i+1} = z_k$) then
    $r \leftarrow r + 1$
    $S \leftarrow S + b_{i+1}$
  else {$b_{i+1}$ is an exit point, say $b_{i+1} = z_k - u_k$}
    $r \leftarrow r - 1$
    $S \leftarrow S - b_{i+1}$
  $i \leftarrow i + 1$
if $r = 0 \wedge S = 0 \wedge b_i \geq 0$ then
  $\mu \leftarrow b_i$
else if $r > 0 \wedge b_i \geq \frac{S}{r} > b_{i+1} \wedge \frac{S}{r} \geq 0$ then
  $\mu \leftarrow \frac{S}{r}$
else if $S + b_{2q} = 0 \wedge b_{2q} \geq 0$ then
  $\mu \leftarrow b_{2q}$
else
  $\mu \leftarrow 0$
$\hat{x}_i(\mu) \leftarrow \min\{u_i, \max\{z_i - \mu, 0\}\}$ (Application of (4.22a))
```

We will consider another variant of the above projection method. If $r, \hat{r} \in R$ are two feasible resource vectors, with $r \leq \hat{r}$, then $g(r) \geq g(\hat{r})$. This results from the fact that each subproblem $i$ yields for $\hat{r}^i$ a flow value at least as good as for $r^i$, since every flow established for $r^i$ is feasible for $\hat{r}^i$, but not the vice versa.

So there is an optimal solution, where all capacity constraints (CPb) and (CPc) are fulfilled with equality, that can hold with equaltity at all. These capacity constraints are represented by (4.20b) in the projetion programs. They can be ineffective due to the upper bound constraints (4.20c): if $\sum_{i=1}^{q} u_i < c$, then (4.20b) can not hold with equality. Otherwise, we can restrict (4.20b) to hold equality.

In fact, in [34, 42] the authors restrict the projection to resource vectors that hold these equalities. In that case, we can drop the restriction (4.21f) in the KKT optimality program. So, we can drop (4.27c) when scanning through the breakpoints as well. Finally, in the Algorithm 1 we can drop the condition $\frac{S}{r} \geq 0$ in all places where it occurs. This variant has the advantage that all resources are distributed among the commodities, even when the subgradient method produced a distribution with spare capacity. On the other hand, if some commodity $k$ needs more resources on some edge or vertex $\nu$, the according capacity constraint is easily violated, as all capacity of $\nu$ is already distributed. Hence, on the succeeding projection the increase for $k$ on $\nu$ will be reduced. Possibly this leads to a slower convergence of this variant of the projection method. We will refer to this variant as PROJ$^=$.

### 4.2.3 Penalty functions

Another way to deal with the mutual capacity constraints instead of projecting onto the feasible region is using penalty functions, as described in Section 4.1.2. As already mentioned there, we will use the loss function

$$s(r) = \sum_{e\in E}\left(\min\left\{c_e - \sum_{i=1}^{q} r_e^i, 0\right\}\right)^2 + \sum_{v\in V}\left(\min\left\{c_v - \sum_{i=1}^{q} r_v^i, 0\right\}\right)^2.$$

As the guidance function, we choose $t(\frac{1}{k}) = k + 10$, where $k$ counts the iterations. Here, $\frac{1}{k}$ is the guidance parameter in the terminology of Section 4.1. The conditions for a barrier function are clearly satisfied. Since $R$ is nonempty and there is a closed neighbourhood of $R$ that is compact, Theorem 4.3 is applicable. That means, convergence of the optima of the augmented objectives (4.12) to an optimal solution of(4.17) is guaranteed. In our implementation we iterate the subgradient method for each augmented objective only 10 times. The reasons for that are twofold: firstly, consecutive optimization of the augmented objective functions results in long running times; secondly, the step length scheme (4.5) in our implementation can not guarantee convergence at all. Hence, we can not expect to obtain optima of the augmented objective functions within proven tolerance.

### 4.2.4 Barrier functions

To obtain a barrier function for model (4.17), we first define an inner loss function

$$s(r) = \sum_{e\in E}(c_e - \sum_{i=1}^{q} r_e^i)^{-p} + \sum_{v\in V}(c_v - \sum_{i=1}^{q} r_v^i)^{-p},$$

where we set $p = 2$. Clearly, this function turns to infinity when $r$ approaches the boundary of $R$. The guidance function is set to $t(\frac{1}{k}) = \frac{1}{k}$, where $k$ is the iteration count again. This rather complicated notation is chosen to be consistent with Section 4.1, so the guidance parameters here are $\{\frac{1}{k}\}$. Moreover, the regularity conditions (4.13) are fulfilled by (4.17), so Theorem 4.4 guarantees convergence of the optima of the augmented objective functions (4.14) to an optimal solution of(4.17).

For our barrier approach, similar statements concerning convergence as for penalty functions hold. We choose to iterate the subgradient method on each objective function 10 times. Applying step length schemes that ensure convergence and iterating each augmented objective to approximate optimality is far too impractical. In fact, convergence theorems of practical use are subject of future research.

### 4.2.5 Using barrier-penalty functions

Gersht and Shulman [27] proposed an approach combining barrier and penalty functions. The main goal is to overcome the weaknesses of both the single approaches: barrier functions prevent to satisfy an inequality constraint with equality, although possibly all optimal points have to do so. On the other hand, such an inequality is unlikely to be satisfied by a penalty function approach, as the sequence of points generated by the subgradient method frequently violates this constraint. This is due to the fact that feasible points close to the hyperplane determined by this inequality are not evaluated with respect to this closeness.

The penalty-terms proposed in [27] are

$$\Phi_{\nu,\epsilon}(r_\nu) = \begin{cases} \left(\frac{c_\nu - \sum_{i=1}^q r_\nu^i}{c_\nu}\right)^{-p}, & \frac{c_\nu - \sum_{i=1}^q r_\nu^i}{c_\nu} > \epsilon, \\ a\left(\frac{c_\nu - \sum_{i=1}^q r_\nu^i}{c_\nu}\right)^2 + b\left(\frac{c_\nu - \sum_{i=1}^q r_\nu^i}{c_\nu}\right) + c, & \text{otherwise,} \end{cases}$$

with

$$\begin{aligned} a &= \frac{p(p+1)}{2\epsilon^{p+2}}, \\ b &= \frac{p}{\epsilon^{p+1}} - \frac{(1-\epsilon)p(p+1)}{\epsilon^{p+2}}, \\ c &= \frac{1}{\epsilon^p} + \frac{(1-\epsilon)^2 p(p+1)}{2\epsilon^{p+2}} - \frac{p(1-\epsilon)}{\epsilon^{p+1}}, \end{aligned}$$

for all $\nu \in E \cup V$. So the extra terms are twice continuous partial differentiable, and thus twice differentiable. The barrier-penalty function in the augmented objective function is

$$\Phi_{\kappa,\epsilon}(r) = \kappa\,\Phi_\epsilon(r) = \kappa\Big(\sum_{e\in E}\Phi_{e,\epsilon}(r_e) + \sum_{v\in V}\Phi_{v,\epsilon}(r_v)\Big).$$

If we choose $\epsilon(\kappa) = \kappa^{2/p}$, we fulfil that

$$\lim_{\kappa\to 0}\epsilon(\kappa) \to 0, \qquad \lim_{\kappa\to 0}\frac{\kappa}{\epsilon^p(\kappa)} \to \infty.$$

When $\kappa \to 0$, the barrier-penalty function converges to a barrier function in the sense of Section 4.1. Moreover, $\kappa$ can be seen as the guidance function $t(k)$ introduced in the theoretical outline. Thus, Theorem 4.4 can be applied, ensuring that the subgradient function converges to an optimal point.

The barrier-penalty function behaves like a barrier function with the required conditions to converge to the solution to the original problem. We only have to assure that $\kappa \to 0$ during the iterations. For that, we choose the prefactor $\kappa = \frac{1}{k}$, where $k$ is the iteration count.

### 4.2.6 Exact penalty approach

As described in Section 4.1, we can determine a fixed penalty function that still yields convergence to an optimum under certain conditions. If we are able to compute a lower bound on the optimal solution of (4.17) and to find an inner point of the feasible set $R$, we can construct such an exact penalty function.

This is easy in our case: we already described how to obtain a lower bound in Section 4.2.1, and an inner point is computed as follows. We set

$$\hat{r}_\nu^i = \frac{c_\nu}{q+1}, \text{for all } i \in [q], \text{for all } \nu \in E \cup V.$$

Thus, none of the inequalities (CPb)–CPe is fulfilled with equality, so we have an inner point. Let $\underline{c} = \min_{\nu \in E \cup V} c_\nu$ be the least capacity. Following the description in Section 4.1, we can conclude that

$$\hat{t} = \frac{q(g(\hat{r}) - \underline{g})}{\underline{c}}$$

is an exact penalty, so the augmented objective function

$$g(r) + \hat{t}\Big(\sum_{e \in E} \min\Big\{c_e - \sum_{i=1}^{q} r_e^i, 0\Big\} + \sum_{v \in V} \min\Big\{c_v - \sum_{i=1}^{q} r_v^i, 0\Big\}\Big)$$

will have the same optimal solutions as (4.17). The exact penalty approach overcomes the disadvantage of the barrier and penalty approaches described in the last two sections: we now do not have to solve a sequence of unconstrained programs, but only one.

### 4.2.7 Obtaining integral solutions

In the preceding part we focused on obtaining solutions of a relaxed version (4.17) of program (CP), when the integrality constraints were dropped. So far, we are just able to obtain a solution similar to a LP-relaxation. In the next part, we give three heuristic techniques producing integral solutions, exploring the structure of the subgradient method.

#### Convex cost functions

Here we want to introduce piecewise linear cost functions for every edge and every demand. This can be handled by splitting each edge into several edges with linear costs. The subproblems $(CP_k)$ are expanded min cost flow problems again, so the dual variables determine a subgradient again. The motivation for this is the following observation:

**Theorem 4.8** *There are convex cost functions for every edge and demand so that the optimal fractional solution to this modification of* (4.17) *is integral.*

**Proof.** Let $f$ be an optimal integral solution to (CP). Now define cost functions $w_e^i$ for every edge $e \in E$ and every demand $d$ as follows:

$$w_e^i(x) = \begin{cases} 1, & \text{if } \le x \le f_e^i, \\ N, & \text{otherwise,} \end{cases} \tag{4.29}$$

where $N$ is a sufficiently large number, e.g., $N > q|E|$. Consider for every commodity $i$ the subproblems $(CP_k)$ as min-cost flow problems on networks $\vec{G}_i$. Let the convex cost functions (4.29) be modelled by splitting each edge $e \in E$ into two edges $e_1$ and $e_2$. That means that in the extended graph $\vec{G}$ we obtain four arcs: two arcs each for $e_1$ and $e_2$. To the edge $e_1$ we assign a cost coefficient 1 and a capacity $f_e^i$, whereas with $e_2$ we associate a cost coefficient $N$ and capacity $r_e^i$, dependent on the current resource distribution vector $r$. The edge $e_1$ is called a cheap edge, and the edge $e_2$ an expensive edge. Obviously, $f$ is still a solution to the modified problem with split edges, since we only change costs.

If there would be a better solution $r$, it can be obtained from routing circular flows in each single-commodity network $\vec{G}_i$, such that the capacity constraints still hold. Suppose we have such a collection of cycles for various demands, and we can send certain amounts flow along these cycles, obtaining a better solution. If there were a circular flow for any demand that is in opposite direction to flow leading arcs on all circle arcs, then we could send at least one unit of flow along the circle as all resources of the cheap edges are integral. Hence, the new flow would be integral, and $f$ was not an optimal integral solution. Note that all cheap edges for all demands are fully utilized. So any circular flow contains at least one expensive edge. Thus, the additional cost for any circular flow will be positive, and $f$ must be optimal to the (fractional) problem. ■

The observation that convex cost functions tend to produce solutions with less non-integral variables was made by Ozdaglar and Bertsekas [56], who solved the routing and wavelength assignment problem with such an approach of convexified objective functions. The construction of convex cost functions given in the proof of Theorem 4.8 depends on a known optimal integral solution. So it is not of practical use within an algorithm. Hence, we derive convex cost functions heuristically. Suppose we are given a fractional solution $r$ of(4.17). We define the cost functions used within the subproblems (4.17) for all edges $e \in E$ and all demands $i \in \mathcal{Q}$ as:

$$w_e^i(x) = \begin{cases} 1, & \text{if } \leq x \leq \lfloor r_e^i \rfloor, \\ 1 + \frac{(i-1)\cdot|E|+k(e)}{q|E|+1}, & \text{otherwise.} \end{cases} \tag{4.30}$$

In (4.30) $k(e)$ is a numbering of the edges in $G$ from 1 to $|E|$. Note that the cost functions (4.30) are mutually different for different edges $e$ and/or different commodities $i$. Finally we have to increase the cost coefficient $M$ for not routed commodities, see Section 3.1. Since the cost for sending a unit of flow can be at most $1 + \frac{q|E|}{q|E|+1} < 2$, we set $M = 2|V|$. As we already mentioned in the original definition of $M$ in Section 3.1, the longest, not self-crossing path between two terminals contains at most $|V|-1$ edges. Hence, the cost for any path of any commodity with respect to the cost functions (4.30) is at most $2(|V|-1)$. By setting $M = 2|V|$ we can ensure that sending flow of any commodity is always preferable to not sending flow.

### Duals from integer flows

Instead of using the dual variables from the min cost flow problems determined by the resource vector $r$, we can use the duals from the integral flow for the subgradient. The optimal integral flow subject to $r$ can be obtained from rounding down the resources to the nearest integers. The new point is generated from $r$, that is, the resources are just rounded as input of the subproblems, not in the sequence of points for the subgradient method. Instead of using the update formula (4.1), we set

$$x_{k+1} = x_k - h_k \frac{\gamma(\lfloor x_k \rfloor)}{\|\gamma(\lfloor x_k \rfloor)\|}. \tag{4.31}$$

Note, that the duals of the integer flow do not longer form a subgradient of $g$, but the real subgradient is perturbed to the integral lattice of $R$.

### Rounding Heuristic

The most simple idea is clearly to round the resource vector $r$ in order to obtain integral solutions. To fulfil the capacity constraints, we round the given fractional solution for every edge and every node in such a way that single components $r_\nu^i$ are rounded up only when enough spare capacity is given, either from capacity unused by $r$ or from preceding down rounding of components of $r$ for this $\nu \in E \cup E$. This is done via Algorithm 2.

**Algorithm 2** Rounding heuristic

```
for all $e \in E$ do
  $\rho \leftarrow c_e - \sum_{i=1}^{q} r_e^i$ ($\rho$ stands for the remaining capacity on this edge)
  for $i = 1$ to $q$ do
    if $\lceil r_e^i \rceil - r_e^i \leq \rho$ then
      $r_e^i \leftarrow \lceil r_e^i \rceil$
      $\rho \leftarrow \rho - (\lceil r_e^i \rceil - r_e^i)$
    else
      $r_e^i \leftarrow \lfloor r_e^i \rfloor$
      $\rho \leftarrow \rho + (r_e^i - \lfloor r_e^i \rfloor)$
```

# Chapter 5

# Branch-and-cut method

In this chapter we focus on a solution technique for integer linear programs, known as the branch-and-cut method. We use it to solve our problem via the formulation (RDF), presented in Section 3.2.3. As in the previous chapter, we first outline the general idea of the method, and then describe its application to our problem.

## 5.1 Theoretical framework

Branch-and-cut is a combination of two solution approaches: cutting plane method and branch-and-bound method. We first describe these two approaches separately, and then their combination.

### 5.1.1 Cutting plane method

The cutting plane method is known since the late 1950's, when Gomory [28, 29] used it to strengthen LP-relaxations of integer and mixed integer programs. It has become a powerful tool in solving many combinatorial optimization problems and, more general, mixed integer programs.

Suppose we are given a linear program with $n$ variables and $m$ linear inequalities:

$$\min c^T x, \tag{5.1a}$$

$$Ax \leq b, \qquad \text{with } A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n. \tag{5.1b}$$

Note that every linear program can be transformed to a representation (5.1). The feasible set of (5.1) is a polyhedron denoted by

$$P_{[m]} = \{x \in \mathbb{R}^n : Ax \leq b\}. \tag{5.2}$$

Assume, that $m$ is very large, i.e., passing all inequalities into a LP-solver is not possible, or not desirable due to the expected solution time. From polyhedral theory, we know that each vertex of the polyhedron $P_{[m]}$ can be described by the intersection of $n$ hyperplanes, arising from inequalities $A_i^T x \leq b_i$ satisfied at equality. Here, $A_i$ denotes the $i$th row of the constraint matrix $A$ in (5.1). Thus, we would need only $n$ inequalities for the description of an optimal vertex of the polyhedron, too. So, the basic idea of the cutting plane method is to solve a reduced formulation of the linear program (5.1) first. For that, let $I \subset [m]$ be a row index set of matrix $A$, we start by solving the reduced formulation

$$\min c^T x, \tag{5.3a}$$

$$A_I x \leq b_I. \tag{5.3b}$$

Then we check whether the optimal solution of (5.3), $x_I^*$, violates inequalities $A_k^T x_I^* \leq b_k$ not yet included in the formulation (5.3), i.e., $k \in [m] \setminus I$. The process of recognizing such violated inequalities is called *separation.* If we find violated inequalities, they (or only some of them) are included in the reduced formulation, i.e.,

$$I' = I \cup \{k \in [m] : A_k^T x_I^* > b_k\}. \tag{5.4}$$

The reduced formulation (5.3) is reoptimized with the new extended row index set $I'$. The whole process

1. Optimizing the reduced formulation
2. Separation of violated inequalities
3. Extending the reduced formulation to violated inequalities

is iterated until the separation fails. Clearly, we will iterate only a finite number of times: in each iteration, at least one inequality is added to the reduced formulation. So at the latest we stop if all inequalities of the original formulation (5.1) are included in the reduced formulation. By adding violated inequalities step by step we hope to consider only inequalities that are more likely to be used for an overall optimal solution of (5.1). Hence, we possibly finish the iteration with far less inequalities than in (5.1).

From a polyhedral point of view, the added inequalities serve as *cutting planes.* To see this, consider a polyhedron $P_I$ arising in the iteration, and the optimal solution $x_I^*$ of the associated reduced program (5.3) with respect to row index set $I$. By adding violated inequalities by (5.4) we obtain a new polyhedron $P_{I'}$, that does not contain $x_I^*$. Hence, the hyperplanes associated with the added inequalities "cut off" the point $x_I^*$ from the polyhedron.

A typical reason for linear programs containing a large number of inequalities is that the linear program arises as a LP-relaxation of a mixed integer program. Consider the mixed integer program

$$\min c^T x + d^T y, \tag{5.5a}$$

$$Ax + By \leq b, \tag{5.5b}$$

$$x \in \mathbb{R}^{n_1}, \tag{5.5c}$$

$$y \in \mathbb{Z}^{n_2}. \tag{5.5d}$$

Then (5.5) can be solved as a linear program with feasible set

$$P_{MIP} = \operatorname{conv}\{(x,y) \in \mathbb{R}^{n_1} \times \mathbb{Z}^{n_2} : Ax + By \le b\}. \tag{5.6}$$

Clearly, $P_{MIP}$ is a polyhedron and can be described by a set of inequalities. These inequalities are usually not all known in advance. So (5.5) can be solved with a cutting plane algorithm, starting with the LP-relaxation of (5.5). If the optimal solution $(x^*, y^*)$ of the LP-relaxation yields non-integral variables $y_i^*$, additional inequalities that are valid for (5.6) must be provided. There are several known methods to generate such valid inequalities from solutions that do not satisfy all integrality constraints in (5.5). Among such methods are *Gomory-cuts*, *General upper bound-cuts*, *Flow cover-cuts*, and *Disjunctive cuts*, to name just a few. For a survey on these methods see, e.g., [69].

Note, that the cutting plane method relies on the solver's ability to fast reoptimize a linear program after new inequalities are added. If the sequence of relaxed linear programs (5.3) had to be solved from scratch in each iteration, the cutting plane method would loose much of its advantages. Fortunately, the ability of easy reoptimizing is given, e.g., when using the dual simplex method, see [59].

### 5.1.2 Branch-and-bound

Here, we turn to the branch-and-bound method. We first state the basic principle, and then turn to its application to mixed integer programs. This method was first described by Land and Doig [45] and was refined in the following by Little et. al. [47] (who founded the name of the method), Dakin [16], and Balas [6], to name just a few. For a detailed survey see [53].

The main idea can be seen as a divide-and-conquer principle: suppose we are given a minimization problem with feasible set $S$ and objective function $g$, and

$$z^* = \min_{x \in S} g(x) \tag{5.7}$$

is its optimal solution value. Furthermore, let the feasible set be decomposed in a finite number $K$ of smaller sets, $S = S_1 \cup \cdots \cup S_K$, with optima

$$z_k = \min_{x \in S_k} g(x), \qquad \forall k \in [K]. \tag{5.8}$$

Then clearly $z^* = \min_{k \in [K]} z_k$. If the subproblems (5.8) are easier to solve than the original problem(5.7), the decomposition can be advantageous. The decomposition is the "branching" part of the branch-and-bound method. It is improved by the "bounding" part, we describe next. Let $\underline{z}_k$, $\overline{z}_k$ be (local) lower and upper bounds on $z_k$, for each subproblem (5.8). Then $\underline{z} = \min_{k \in [K]} \underline{z}_k$ is a (global) lower bound on $z^*$, and $\overline{z} = \min_{k \in [K]} \overline{z}_k$ is a (global) upper bound on $z^*$. If for some subproblem (5.8) we get $\underline{z}_k \ge \overline{z}$, we do not need consider this subproblem any further, as it could not yield a better solution than the solutions found in the remaining subproblems. Neglecting a subproblem because of its lower bound and the overall upper bound is called *pruning by bound*. It might be possible that some of the subproblems (5.8) are infeasible, i.e., if some $S_k$ are empty. Such a subproblem with proven infeasibility can also be dismissed, this operation is called *pruning by infeasibility*. Finally, *pruning by optimality* takes place if some subproblem is solved to an optimal solution, and can be dismissed, too. The bounding feature improves the decomposition (branching) principle because lower and upper bounds for the subproblems are usually much easier to obtain than the according solutions. Additionally, if we are able to improve local lower bounds gradually, we can fine-tune the bounding process so that pruning takes place as often as possible.

If the problem is a mixed integer program (5.5), then branch-and-bound can be applied to it. Actually, branch-and-bound is one of the favourite methods to solve mixed integer programs, especially in extensions such as branch-and-cut or branch-and-price. Usually branch-and-bound is applied recursively. The feasible set $P_1^0 = P_{MIP}$ is subdivided by adding linear constraints, partitioning $P_1^{(0)}$ into subsets $P_1^{(1)}, \dots, P_{K_1}^{(1)}$. The obtained subproblems are mixed integer problems as well, so branch-and-bound can be applied to them. More formally, if we are given a problem with feasible set $P_j^{(i)}$, we derive the following $K_l$ subproblems:

$$z_k^l = \min\ c^T x + d^T y, \tag{$MIP_k^l$a}$$

$$(x, y) \in P_j^{(i)}, \tag{$MIP_k^l$b}$$

$$A_k^l x + B_k^l y \leq b_k^l, \tag{$MIP_k^l$c}$$

$$(x, y) \in \mathbb{R}^{n_1} \times \mathbb{Z}^{n_2}, \tag{$MIP_k^l$d}$$

$$\text{with } P_k^{(l)} = \{(x, y) \in \mathbb{R}^{n_1} \times \mathbb{Z}^{n_2} : (x, y) \text{ fulfils } (MIP_k^l\text{b}), (MIP_k^l\text{c})\}, \tag{$MIP_k^l$e}$$

and $k$ is running from 1 to $K_l$.

Iterating this procedure, we obtain a tree-structure, usually referred to as a *branching tree*, see Figure 5.1. The subproblems are represented in the tree as nodes. All nodes whose subproblems have still to be solved are called *active*. Note that all active nodes are leaves of the branching tree.

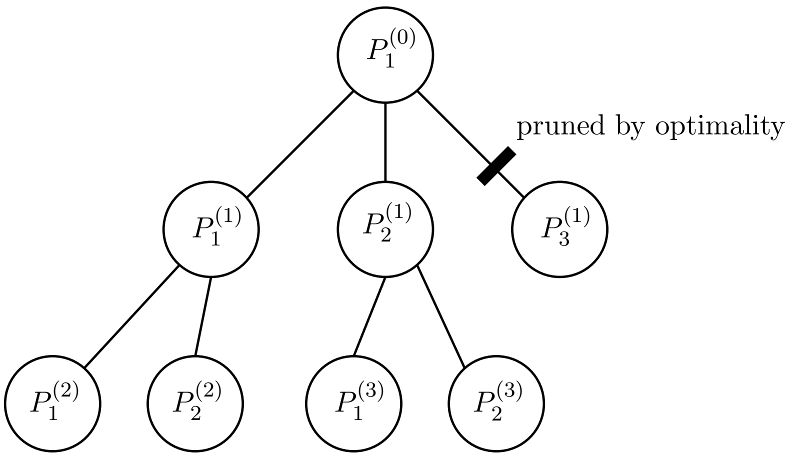


Figure 5.1: A branching tree

The process stops, if all subproblems generated in the branching tree are either decomposed into subproblems (the inner nodes of the branching tree) or pruned, that is, if no active nodes are left. The overall optimal solution of the original problem is then the best (minimal) solution among the solutions of the subproblems. If no such solution was found, the original problem is infeasible.

To solve the mixed integer problem this way becomes similar to a complete enumeration of all of the integral variables, if the branching tree is fully explored. Such a complete enumeration yield an exponential number of subproblems to solve. The bounding process is the key tool to prevent the number of subproblems to grow too fast. Note however, that bounding can not guarantee to obtain only a polynomial number of subproblems.

How can lower and upper bounds be derived for the subproblems? Clearly, the LP-relaxation of any subproblem yield a local lower bound $\underline{z}_k^l$ of a subproblem $MIP_k^l$. On the other hand, any feasible point of $P_{MIP}$ yields an upper bound for the subproblem's solution. Such feasible points are derived by a heuristic, or occur as optimal solutions of subproblems. Clearly, tight lower and upper bounds will shorten the branch-and-bound procedure, as more subproblems are likely to be pruned.

Several parts of the branch-and-bound procedure are still to be specified in more detail:

- Branching rule

  We already described the general branching in $(MIP_k^l)$. The open question is how the branching constraints$(MIP_k^l$c) are derived. For that, let $a^Tx$ be a linear form with integral coefficients, and let $\{b_1, \dots, b_L\}$ be the set of all integral values that $a^Ty$ can attain in $P_j^i$. Then we can derive $K_l$ subproblems $MIP_1^l, \dots, P_{L_l}^l$ by adding constraint $a^Ty = b_i$ to $P_j^i$, for all $i \in [L_l]$. Then the $P_k^l$ are clearly disjunct, and all integral points feasible for $P_j^i$ are feasible for some $P_k^l$.

  As the running time of most LP-solvers is very sensitive to the number of non-zero coefficients of the constraints, a good choice for $a$ is a unit vector. That means, that we actually branch on a single variable $y_t$, obtaining subproblems with additional constraints $y_t = b_i$, for $i \in [K_l]$. Another advantage of branching on single variables is that the admissible values $b_1, \dots, b_{K_l}$ are usually easy to compute. Moreover, fixing a variable $y_t$ with a constraint $y_t = b_i$ decreases the subproblem's size, as the variable is no longer present in the subproblem, but can be replaced by the constant $b_i$.

  If the number $L$ of admissible values $b_i$ is large, the scheme with adding equality constraints to the subproblems lead to a large number of subproblems. So a two way branching rule is possibly preferable. A two way branching rule produces only two subproblems $P_i$, $P_j$ by adding constraints $a^Tx \leq b$, $a^Tx \geq b+1$ to $P_i$, $P_j$, respectively. Here, $b \in \{b_1, \dots, b_L\}$ is an integer carefully chosen dependent on the fractional solution of $P_k$.

  The most widely used branching rule is a two way branching rule on single variables. That means, that in subproblem $MIP_j^i$ an integral variable $y_t$ is selected and an associated branching value $\beta \notin \mathbb{Z}$. Two subproblems $MIP_1^l$ and $MIP_2^l$ are created by adding the restrictions $y_t \leq \lfloor \beta \rfloor$ and $y_t \geq \lceil \beta \rceil$, respectively. A two way branching rule has the advantage that only two new problems arise in the branching tree. Moreover, the added constraints are simple in the sense of non-zero coefficients. Sometimes LP-solvers can handle bounds on variables in a special manner, decreasing the computational effort even more. Finally, branching on a fractional variable is likely to force this variable to an integer value.

When applying this two way branching rule, we still have to decide on which variable we want to branch. The best idea is usually to branch on an integral variable that has a fractional value in the optimal fractional solution of $P_j^i$. As the optimum $\underline{z}_k^l$ of the LP-relaxation of $MIP_j^i$ is neither feasible for $MIP_1^l$ nor for $MIP_2^l$, we can hope that the optima of $MIP_1^l$ and $MIP_2^l$ are strictly greater than $\underline{z}_k^l$, thus possibly improving the overall lower bound $\underline{z}$. Again, there might be several possibilities for branching variables in $P_k$. Taking the mentioned improvement of the lower bound into account, typical choices are either to choose branching variables with large cost coefficients, or variables with small reduced costs, or variables whose fractional value part is closest to 0.5.

Branching rules are shown to have a great impact on the branching tree's size, and thus on the running time of the solution process. The decision which branching rule should be used depends on the program to be solved, and no general preferable rule can be given.

- Node selection

  The sequence in which the subproblems should be examined is another important parameter of the branch-and-bound process. Again, a robust node selection rule working best for all problems is not known. So, we have to determine a good rule for each problem on its own.

  Choosing the "right" subproblem will yield tighter lower or upper bounds. Thus, a typical procedure is choosing a branching node with a least local lower bound, as the new arising subproblems are likely to provide better bounds. Note that the least local lower bound determines the global local bound. Another approach is to choose the node with best upper bound, as better or even optimal solutions are to find more probably in the neighbourhood of already known good solutions. Finally, we mention the *diving* rule. This rule selects a node that is deepest in the search tree. Such a node has underrun most branching decisions among all yet active nodes, so the selected node represents a most restricted subproblem. Hence, we can hope to either obtain a good lower bound from this node, or to prune the node.

- Heuristics

  Obtaining good upper bounds decreases the number of search tree nodes considerably. Unfortunately, a general scheme for deriving good feasible solutions is not given. General purpose solvers try to round fractional variables in some constraint dependent sequence. However, a problem specific heuristic is usually preferable.

### 5.1.3 Branch-and-cut

We next turn to the question how cutting plane method and branch-and-bound can be combined. The general scheme of branch-and-cut is similar to the branch-and-bound scheme described above. We consider a branching tree as well, but in contrast to simple branch-and-bound cutting planes can be generated at every node of the branching tree. So, in each node a cutting plane method takes place. The only difference is that a subproblem needs not to be solved to integer optimality with the cutting plane method, as we can choose to branch on this subproblem. In case of combinatorial cutting planes, we must be careful when deriving upper bounds. A heuristic producing such upper bounds has to consider that there can be valid inequalities that are not yet part of the description, but are necessary for a correct description. Thus, a point satisfying all constraints at some branching node might not be feasible for the original problem. The point rather has to be checked whether or not it fulfils all restrictions that are part of the problem. Only if it does so it can be used for an upper bound.

## 5.2 Application to IMCF-N

We next describe how branch-and-cut can be applied to our problem IMCF-N, based on the formulation (RDF), as given in Section 3.2.3. First we motivate why we use branch-and-cut, and specify the components of the method introduced in Section 5.1. As mentioned, we have to decide on a branching rule, a node selection scheme, a heuristic for upper bounds, and on a separation method for cutting planes.

The reason for applying branch-and-cut to formulation (RDF) is twofold. First, the formulation contains an exponential number of inequalities arising from the general cuts in (RDFd). In general, the number of general cuts in a graph can be exponential in the number of edges and nodes of the graph. So we can not include all general cut constraints, since the time needed to solve the linear programs occurring in a branch-and-bound tree would be too large. Moreover, since we deal with $q(|V| + |E| + 1)$ variables in formulation (RDF), only $q(|V| + |E| + 1)$ inequalities of the formulation can be linearly independent. Hence, most of the general cut constraints are redundant, and we consider them only as demanded, i.e., if they are violated during the solution process. The second reason for using branch-and-cut is that we deal with an integer program, and branch-and-cut is one of the most successful solution methods for integer programs.

The initial relaxation of formulation (RDF) is a LP-relaxation, where the capacity constraints (RDFb), (RDFc), and the diversification constraints (RDFe), (RDFf) are included. Additionally, some of the general cut constraints (RDFd) were included as well by the following construction. For each demand $k \in \mathcal{Q}$ independently, we start with all resources zero, i.e., $r^k_\nu = 0$, for all $\nu \in V \cup E$. With the separating routine described in Section 5.2.4 we generated minimal general $s_k$-$t_k$-cuts as long as no such cut could be found with capacity less than $d_k$. Once a cut was found, one edge or node in the cut was assigned a resource $d_k$. Thus, no cut can occur twice.

### 5.2.1 Branching rule

We decide to use a two way branching scheme on single variables as described in Section 5.1. The branching variables are chosen in order of decreasing costs. Thus, the artificial variables $a_k$ describing the amount of flow not routed for demand $k$ are chosen first, if they are not integral. As all variables $a_k$ have the same cost coefficient, we still have to decide which of these variables to choose for branching, if more than one of them is not integral in a solution of a intermediate relaxation. In that case, we choose the most fractional variable, i.e., the variable $a_k$ whose fractional part is closest to 0.5. If all variables $a_k$ are integral, the edge resource variables $r_e^k$ are chosen next as possible branching variables. Finally, the node resource variables $r_v^k$ are considered. For the resource variables $r_e^k$ and $r_v^k$ the same tie-breaking rule is applied as for the $a_k$. That is, if more than one has a non-integral value, we choose the most fractional. We choose this branching order to obtain rapidly increasing lower bounds, since a change of a variable with high associated costs has more impact on the objective value. By this, we aim to keep the branching tree's size as small as possible.

### 5.2.2 Node selection

Computational experiments showed that even in the case of small gaps between lower and upper bound a large number of nodes has to be considered. Thus, pruning nodes is very desirable. By testing some standard rules we have decided to choose to branch on a node with a local lower bound equal to the global local bound.

### 5.2.3 Heuristics

To obtain upper bounds we used the built-in heuristic of the MIP-solver CPLEX. This heuristic yields already good upper bounds. Thus, a problem specific heuristic can not improve the upper bound that much. Due to the complexity results given in Section 2.4.1 it would be difficult to develop a combinatorial heuristic anyway.

### 5.2.4 Cutting planes

We next describe how cutting planes were involved. As already mentioned, the number of general cuts in formulation (RDF) can be very large. So we decided to put only few general cuts into the initial formulation, and we added more of them only when they were recognized to be violated.

The separation procedure works as follows. Suppose we are given a point $(r^*, a^*)$, that is considered optimal in the current relaxation, at some node of the branching tree. So for each $k \in \mathcal{Q}$, we have resources $r_e^{*k}$ for all edges $e \in E$, and $r_v^{*k}$ for all vertices $v \in V$. Then we are able to compute a minimal general $s_k$-$t_k$-cut for demand $k$. This can be done via any standard max-flow or min-cut procedure. We decided to use a network simplex procedure, working on the extended graph $\vec{G}$ as described in Section 3.2.4. Once we have found a minimal general $s_k$-$t_k$-cut, we check whether this cut has capacity at least $d_k - a_k$. If not, we have found a violated inequality to be added as a cutting plane.

When given a point $(r, a)$, we decided to check for cutting planes for all demands, to fasten computation. So we can add at most $q$ cutting planes at once, avoiding intermediate reoptimization.

The general cuts present in the initial formulation are generated as follows.

The number of initial cuts derived like this was substantial, compared to the number of cuts generated at all. Although no guarantee can be given that the initial cuts are good in the sense of being inevitable in an optimal solution, they turned out to be useful in speeding up the solution process.

Details of the influence of the methods outlined on the solution process are described in Chapter 6.

# Chapter 6

# Computational results

In this chapter we describe the application of the methods outlined in the Chapters 4 and 5. First, we give some implementation details and mention the software incorporated in Section 6.1. In Section 6.2 we describe the test instances we used for the comparison of the methods. Finally, we report on the obtained results in Section 6.3, and analyse them.

## 6.1 Implementation

We implemented the methods described in the Sections 4.2 and 5.2 in C++ using the ANSI Standard Template Library. Our implementation was built on top of the Optical Network Design (OND) project software by Koster and Zymolka, see [71]. As the implementation of the min cost flow solver the software MCF by Löbel [49] was used through the MCFClass interface by Frangioni [20]. The branch-and-cut method was developed with CPLEX 7.5 and Concert Technology 1.2 [37, 36].

## 6.2 Test instances

Unfortunately, there are no instances for IMCF-N publicly available to compare our results with other researcher's results. A collection of instances of the fractional multicommodity flow problem is provided in [21], together with some computational results from various models and implementations. These instances are not suited for our purposes due to their characteristics: they contain large supply graphs, or very dense supply graphs, or very few demands. So we have chosen to build own test instances, inspired by the instance generator described in [11].

Our instance generator works as follows. First of all, we have to provide the following input data:

- the number of vertices $|V|$ in the supply graph,
- a density parameter $\delta$ for the supply graph,

- the number of demands $q$,
- the diversification parameter $\rho$.

Then the instance generator computes a random position in the plane for every vertex of the supply graph. These positions are uniformly distributed in the square $[0, 100] \times [0, 100]$. Afterwards a set $E_{pot}$ of $\delta(|V| - 1)$ of potential supply edges is generated by $\delta$ minimum spanning tree computations in the following way. We start with the complete graph $\hat{G} = (V, \hat{E})$. To each edge $vw \in \hat{E}$ we assign a weight that is the Euclidean distance between $v$ and $w$. In the beginning let $E_{pot} = \varnothing$. In each of $\delta$ iterations, we compute a minimum spanning tree of $\hat{G}$ with respect to the edge weights. Let $E_t$ be the edge set of the spanning tree. We set $\hat{E} \leftarrow \hat{E} \setminus E_t$, $E_{pot} \leftarrow E_{pot} \cup E_t$, and the next iteration follows. This construction yields a set of potential supply edges similar to real world instances. Since costs of optical fibers are length dependent, supply graphs are rather sparse, and short edges are preferred to long edges.

After constructing the potential supply edges, the demands are computed. For that, $q$ terminal pairs are randomly chosen, uniformly distributed among all vertex pairs $vw \in V \times V$. The according demand values are randomly chosen integers from the interval $[1, 15]$. We provide the same diversification parameter $\rho$ for all demands.

To obtain instances that are close to real world instances, the dimensioning of the network is carried out by the tool DISCNET by Wessäly [68] and the preprocessing features of the OND software. The input of this tool is a set of potential vertices and edges that can be established as supply vertices and supply edges. Furthermore, demands must be specified, i.e., the demand's terminals, its value, and diversification parameter. Finally, the installable hardware must be given, as described in Chapter 2. For that, optical fibers, WDM systems, and OXCs are specified with their routing capacity (in terms of lightpaths) and costs. To the WDM systems we assigned a capacity of 10 channels, and we provided 2 types of OXCs with capacity for 2 and 5 lightpaths, respectively.

DISCNET computes from this input a dimensioning of the optical network, minimizing the overall costs of installed hardware. It is guaranteed that all demands can be fully routed with respect to its diversification parameter, but the flow can be fractional. However, the dimensioning is provided in integral units of the routing capacity of the specified hardware.

By applying DISCNET we obtain a supply graph that is tight dimensioned in the sense that most demands can be fully satisfied by integral routings, while many supply edges and nodes are fully utilized. The instances obtained this way are described as having tight capacities in Table 1. To explore the behaviour of our solution approaches in more detail, we also derive high and low dimensioned instances from the tight dimensioned ones. The high dimensioned instances have twice the capacity on every edge and vertex as the tight dimensioned instances, while in the low dimensioned instances it is half the capacity. It is likely that the high dimensioned instances have enough edge and vertex capacities to satisfy all demands, and that the according capacity bounds are not the main influence of the optimal solution value. So we can observe the performance of the solution methods if less interference between the commodities takes place. On the other hand, the low dimensioned instances shall give an insight into how the solution methods perform if not enough capacity is given to route all commodities.

The instances created this way differ in four characteristics:

- The supply graph size ranges from 10 vertices to 76 vertices, while the supply graph density $p_G = \frac{2|E|}{|V|(|V|-1)}$ ranges from 0.04 to 0.58. The graph density depends on the parameter $\delta$ used in the initial construction of the instances. By the instance construction the relation $p_G \leq \frac{2\delta}{|V|}$ holds. Due to the dimensioning process of DISCNET that determines which potential supply edges are realized, the same $\delta$ parameter can lead to a different number of actual supply edges and graph densities $p_G$.
- The number of demands ranges from 15 to 150, or as expressed with the demand graph density $p_H = \frac{2q}{|T|(|T|-1)}$, $p_H$ ranges from 0.01 to 0.89. Recall that $T$ is the set of demand terminals, i.e., the set of vertices of the demand graph.
- The diversification parameter used are 0.25, 0.5, and 0.75. Note that a diversification parameter 0.25 requires that at least four vertex-disjoint paths between the terminal vertices of each demand exist.
- The provided capacities range roughly from half of the capacities needed to route all commodities to twice the capacity needed to route all commodities.

The details of the single instances we used can be seen in Table 1. The names of the instances contain the parameters that are used when constructing the instances. For example, N20_G3_H100_D75_h stands for 10 vertices, a density parameter $\delta = 3$, 100 commodities, a diversification parameter $\rho = 0.75$ for all demands, and finally a high dimensioned instance.

## 6.3 Results

In this section we first describe the computational results of the subgradient method applied to formulation (CP). We characterize the best parameters of the subgradient method, for each variant of the method. By choosing the best parameters and the best variants, we compare the results for the three heuristics described in Section 4.2.7. Then we give the results obtained by the branch-and-cut method for formulation (RDF). Finally, we compare the results of both methods to the benchmark results obtained with formulation (EFF).

### 6.3.1 Results of the subgradient method

In Section 4.2 we outlined that the subgradient method is suitable to be applied to the formulation (CP). We presented six variants of the subgradient methods that differ in the way the constraints (CPb)–(CPe) are treated. In a first step, we compare these variants by applying only the rounding heuristic described in Section 4.2.7. For all test runs we set a running time of 60 seconds. Experiments have shown that the quality of the solutions does not improve significantly if more running time is given to the methods. Comparing all variants by the same running time, i.e., instead of comparing them by the number of iterations, results from the heuristic character of the subgradient method. Since a heuristic is meant to be applied several times during an exact solution approach, the main issue is its running time.

We use the Crowder adaption scheme, see Section 4.1, to extract the best variant of the subgradient method. We extensively tested all variants to find an appropriate initial value for $\lambda_1$ used in the step length computation, see Section 4.2.1. It turns out that a careful adjustment of $\lambda_1$ is essential. This can also be seen in Table 3, where the best values for $\lambda_1$ for the different subgradient adjustment rules are listed. These results depict that the most successful values for $\lambda_1$ differ substantially among the instances. The integral solutions are produced by the rounding heuristic described in Section 4.2.7.

The results for the different constraint handling variants are given in Table 2. The quality of the solution is measured in per cent, where the optimal solution or best lower bound is used as a reference. This reference value is taken from the branch-and-cut method for either formulation (EFF) or formulation (RDF), cf. Section 6.3.2. It is more reliable than the lower bound computed by the subgradient method itself, since the quality of the lower bound of the subgradient method varies strongly among the instances.

We make the following observations for the variants of the subgradient method:

- First of all, the projection method PROJ works best on all instances, followed by the variation PROJ$^=$. With dramatical gap the penalty method, the barrier-penalty method, the exact penalty method, and the barrier method follow in this order. The reason for that is that the four penalizing methods yield large subgradients due to the penalty terms in the augmented objective function.

- As expected, the solution quality decreases with increasing instance size. Especially the number of commodities has a large impact.

- The instances with high and low capacities are approximated best by all variants, while the tightly dimensioned instances are rather hard to approximate. This hardness results from the fact that in the tightly dimensioned instances usually all demands can be fully satisfied in an optimal solution. By the subgradient method's approximation usually a fraction of the demands can not be routed, and in that case the high cost coefficient for not routed demands increases the solution value fast.

Taking into account the above results we decide to use the projection variant PROJ throughout the following considerations. We next examine which subgradient adjustment rule should be used. In Chapter 4 we introduced the Crowder rule, the Camerini-Fratta-Maffioli (CFM) rule, and the modified Camerini-Fratta-Maffioli (mCFM) rule. We apply these rules together with the projection method PROJ, and again test for the best $\lambda_1$ value. The results can be seen in Table 3. In this table the results of the Crowder rule as applied to obtain the results of Table 2 are restated for comparison. We observe:

- The modified Camerini-Fratta-Maffioli rule and the Crowder rule work best. The simple Camerini-Fratta-Maffioli rule never yields the best result.

- The initial values $\lambda_1$ differ substantially for different instances and different subgradient adaption rules. Choosing other values for $\lambda_1$ as presented in the table can lead to a dramatic loss of solution quality, even a divergent behaviour of the method occurs.

Finally we turn to the two heuristics for obtaining integral solutions, the convex cost heuristic and the integral flow subgradient heuristic as outlined in Section 4.2.7. Both heuristics are combined with the rounding heuristic, since the first two mentioned heuristics do not ensure to obtain an integral solution.

We decide to use the modified Camerini-Fratta-Maffioli rule for the comparison of the heuristics, since this rule proved to be successful in previous experiments. The results are given in Table 4, with a restatement of the results obtained without the heuristics from Table 3. We observe:

- Both heuristics improve the solution quality compared to the results without the heuristic. However, the improvement is not substantial.
- Comparing the two heuristics, none of them is superior to the other. The convex cost heuristic does less iterations due to the more complicated subproblems $(CP_k)$. Recall from Section 4.2.7 that for the convex cost heuristic each of the subproblems has twice the arcs as in the integral flow subgradient heuristic, since the convex cost functions are modelled by split edges. On the other hand, the heuristic using subgradient directions derived by the integral flows lacks to reach integral points due to the step lengths used. Experiments with longer computation time does not yield better results.

By the results presented in this section we conclude that the subgradient method works well for instances with high and low capacity. The most interesting instances however, i.e., the tight dimensioned instances that refer to the application presented in Chapter 2, yield unsatisfying results.

### 6.3.2 Results of the branch-and-cut method

We apply the branch-and-cut method to formulation (RDF) as described in Section 5.2. The running time for each instance was bounded to one hour. The results of the method are presented in Table 5. The table lists the following details:

- Row GAP shows the gap between the best solution found and the lower bound of the branch-and-cut method in per cent. A gap of 0.00 means that the according instance was solved to optimality.
- Row TIME lists the time needed by the branch-and-cut method, rounded to seconds. This includes the time for the separation of the general cut inequalities, the time for the MIP-solver, and the time for intermediate output, but not the time for reading the data and constructing the initial data structures.
- Row INIT CUTS gives the number of initial cuts included in the first relaxation.
- Row CUTS shows the number of general cut inequalities generated in the whole process, including the initial cuts.
- Row NODES lists the number of branching nodes that were generated in the branching tree.

Examining the obtained results we observe that 24 of the 33 instances are solved to optimality, while for nine instances solutions within a gap of less than 0.42% are found. In contrast to the results of the subgradient method, the low dimensioned instances turn out to be most difficult for the branch-and-cut approach. The number of branching nodes shows an interesting behaviour. The instances are either solved within few branching nodes or require a very large number of them. Note that the instances with most branching nodes are usually not solved to optimality, so even more nodes would be needed to reach optimality.

### 6.3.3 Comparison of the results

As mentioned in Chapter 3 we compare the results of the subgradient procedure for formulation (CP) and the results of the branch-and-cut method for formulation (RDF) with benchmark results. Here, a branch-and-cut method applied to the standard formulation (EFF) serves as the benchmark. No parameter settings are made, i.e., node selection, branching rule, and generation of cuts are left to the default behaviour of the MIP-solver. Again, a time bound of 3600 seconds is stipulated. In Table 6 the details of the obtained results are given. It contains the gap between the best solution found and the lower bound, measured in per cent, the time needed for the computation, measured in seconds, and finally the number of rows in the formulation. The MIP-solver tries to aggregate formulation (EFF), and the number of rows given in Table 6 refers to after aggregation.

By the given results we conclude that the subgradient method together with the heuristics to find integral solutions does not yield satisfying results. Only in special cases it provides solutions for formulation (CP) within reasonable tolerance. As already mentioned in Section 6.3.1, the most interesting instances are the tight dimensioned ones. Especially for these instances the subgradient method is not competitive with the two branch-and-cut methods. Additionally, the lack of a good lower bound in the subgradient method requires the use of other solution techniques to prove the quality of the obtained solutions.

Both branch-and-cut methods provide satisfying results for all instances. Especially the tight dimensioned instances, as of major interest, are solved quickly. The benchmark of formulation (EFF) is superior to the branch-and-cut method based on formulation (RDF). Comparing the number of generated cuts for (RDF) with the number of rows in (EFF) shows that both formulations contain a similar number of constraints, except for low dimensioned instances, where considerable more general cut inequalities are separated for (RDF). The difference in the running times is mainly due to the repeated resolving of the relaxations of formulation (RDF).

We also tested our implementation with real world instances provided to us by our project partners T-Nova. Due to a disclosure agreement we are not able to describe these instances in detail. However, the instances given in Section 6.2 cover the main characteristics of the real world instances. Hence the results for them are similar to those results for the “artificial” instances.

# Chapter 7

# Conclusions

In this chapter we draw a conclusion on the results we obtained in this work. We first judge the capabilities of the subgradient approach. Then we summarize the results of the branch-and-cut method. Both the heuristic and the exact approach are compared to the benchmark formulation and its results. Finally, we give an outlook in Section 7.3.

## 7.1 Evaluation of the subgradient approach

The subgradient method produces feasible solutions for the test instances described in Section 6.2 within a gap of 1.2% to 157.3% to the best lower bound obtained by any method. This gap is unsatisfying large. Compared to the results of the two branch-and-cut methods we must conclude that the subgradient method and the heuristics applied within are unsuitable for our problem. The branch-and-cut methods produce better solutions, usually optimal solutions, in less time than the subgradient approach. Thus, there is no need for the heuristic we suggested providing upper bounds.

The formulation (CP) (or its relaxation (4.17)) is not suitable to apply the subgradient method to produce lower bounds for formulations (EFF), (RDF). Since (CP) provides primal solutions, primal in terms of formulation (RDF), the relaxation (4.17) must be solved to proven approximated optimality to provide such a lower bound. But good approximation guarantees are beyond the capabilities of the subgradient method. However, the subgradient method might still be useful to solve a Lagrangian dual program to obtain lower bounds.

The results of the branch-and-cut method suggest that the difficulty of the integer routing problem does not arises from bad upper and lower bounds, but from high degeneracy of the programs due to many equivalent solutions. Thus, our subgradient approach is not needed to improve the performance of the exact solution approaches.

## 7.2 Evaluation of the branch-and-cut approach

The branch-and-cut method described in Chapter 5 produces feasible solutions for all test instances described in Section 6.2. The found solutions are mostly proven to be optimal, for a few instances a gap of up to 0.42% occurred.

It turned out that good lower and upper bounds were found quickly, but for some instances it was difficult to obtain optimal solutions. The difficulties lead back to an enormous number of branching nodes. This is due to a large number of equivalent solutions. The formulation (RDF) aimed to restrict the number of equivalent solutions, but any progress toward this goal was nullified by the effort for separation of the general cut inequalities.

However, both branch-and-cut methods for the formulations (EFF), (RDF) offer a quick possibility for solving the integer multicommodity flow problem at least approximately. Considering the underlying optical network design problem it turns out that obtaining good solutions very fast is advantageous for the dimensioning task. If the overall design process is decomposed as described in Section 2.1.2 the dimensioning has to be repeated until a routing of all demands is possible. Both lower and upper bounds of the integer routing problem contribute to this iteration. Lower bounds give necessary conditions for the capacities to provide while upper bounds, i.e., feasible solutions prove sufficiency of the provided capacities.

## 7.3 Outlook

To our opinion, a promising approach for future research is to review the formulations (EFF), (RDF) to decrease the degeneracy of these formulations. One possibility could be a different objective function, where demands are rated differently. Another idea is to narrow the set of feasible solutions by introducing additional constraints to the formulations. By that feasible solutions might be excluded as long as at least one optimal solution is still contained in the feasible set.

Another approach to achieve better performance of the branch-and-cut method is to focus on polyhedral investigations. The dimensioning task of optical network design is often carried out under use of Theorem 2.2. However, this applies only for fractional multicommodity flows. An extension to integer multicommodity flow would improve the approaches to the dimensioning problem and would yield deeper insight to the integer multicommodity flow problem.

# Zusammenfassung

Optische Netzwerke sind Netzwerke aus dem Telekommunikationsbereich, deren Vorteil in ihren hohen Bandbreiten und Geschwindigkeiten liegt. Durch den technologischen Fortschritt und die Möglichkeiten, die neuartige technische Komponenten bieten, wurden mathematische Lösungsansätze vor neue Herausforderungen gestellt.

Diese Diplomarbeit entstand im Rahmen von Projekten des Konrad-Zuse-Instituts in Kooperation mit zwei Industriepartnern. Die Projektarbeit verdeutlichte, dass die Probleme, die sich im Rahmen von optischen Netzwerken ergeben, mit den bisherigen Lösungsmöglichkeiten nicht zufriedenstellend gelöst werden können. Dies resultiert daraus, dass herkömmliche Lösungsansätze an der Komplexität des Gesamtproblems scheitern.

Ziel dieser Arbeit ist es, Möglichkeiten und Lösungsvorschläge für ein spezielles Routingproblem zu entwickeln, dass im Rahmen der Gestaltung von optischen Netzwerken entsteht. Dieses Problem entsteht bei der Aufgabe, gegebene Kommunikationsanforderungen durch die Verbindung der entsprechenden Terminals mit Hilfe von Pfaden zu erfüllen.

Derartige Verbindungen werden im Rahmen von optischen Netzwerken mit Hilfe von Lichtwegen realisiert. Das Routingproblem besteht genau in dieser Realisierung ausreichender Lichtwege. Typischerweise können in solchen optischen Netzwerken die Lichtwege nicht beliebig geteilt werden, sondern müssen als ganze Einheit bestehen bleiben. Daraus folgen systemimmanente Beschränkungen für den mathematischen Lösungsansatz.

Aufgrund des hohen wirtschaftlichen Aufwands sind die Routingkapazitäten eine knappe Ressource, was das Problem zusätzlich erschwert. Es ist daher umso wichtiger, effektive Lösungswege für das Routingproblem zu entwickeln.

Das Routingproblem ist bereits in anderen Bereichen und Anwendungen behandelt worden, jedoch ist der dieser Arbeit zugrunde liegende Lösungsansatz neu. Wir haben uns für den so genannten ressourcenorientierten Zerlegungsansatz entschieden, da dieser besonders vielversprechend für die spezifischen Anforderungen von optischen Netzwerken ist.

Zunächst stellen wir die grundlegenden Aspekte der optischen Netzwerke und die im Rahmen des Designprozesses entstehenden Probleme dar, womit die Schwierigkeit des zugrunde liegenden Problems verdeutlicht wird. Im Anschluß daran wird ein Modell und eine spezifische Formulierung entwickelt. Diese Formulierung wurde durch vorangegangene Forschungsarbeiten und Untersuchungen in dieser Arbeit motiviert.

Ferner wurde ein Lösungsansatz basierend auf der Subgradientenmethode entwickelt und anschließend durch heuristische Ansätze zur Behandlung der Ganzzahligkeitsbedingungen

weiterentwickelt.

Es wurden verschiedene Varianten der einzelnen Aspekte der Subgradientenmethode verglichen. Weiterhin wurde ein exakter Lösungsansatz, basierend auf dem Branch-and-Cut Verfahren, entwickelt.

Mithilfe von Testinstanzen, die strukturell real existierenden optischen Netzwerken ähneln, werden die entwickelten Methoden schließlich getestet und bewertet.

Letztlich werden beide vorgestellten Lösungsansätze und ihre Ergebnisse mit dem durch eine Standardformulierung erzielten Vergleichsansatz verglichen. Dieser Vergleichsansatz basiert ebenfalls auf der Branch-and-Cut Methode.

Abschließend werden alle erzielten Ergebnisse verglichen und ausgewertet.

# Abstract

Optical networks are structures arising in telecommunication. These optical networks provide high speed and high bandwidth capabilities. Due to engineering progress and modern technical equipment new challenges appear to mathematical models and solution methods.

This thesis was inspired by two projects of the Konrad-Zuse-Zentrum with industrial partners. The project led to the realization that the problem of optical network design can not be sufficiently solved by ordinary solution approaches which tackle the whole design problem at once.

The aim of this work is to provide means and methods to solve a specific routing problem arising in the design process of optical networks. This problem evolves from the task to provide connections between pairs of terminals dependent on a given communication demand. Such connections are realized via lightpaths in the optical network. The problem of establishing such lightpaths is known as a routing problem. An inherent characteristic of optical networks is that paths can not be split arbitrary but have to be provided in basic units. This results in integrality constraints of the mathematical model.

Due to economic reasons capacities providing routing capabilities are scarce resources. This requires sophisticated solution approaches for the routing problem. The routing problem has arisen before in different areas and applications but was not yet approached from the point of our specification. This resource directed decomposition approach is chosen by us since it is especially promising for the specific requirements of optical networks.

After surveying the basic aspects of optical networks and the problems arising in the design process we state the hardness of the problem under consideration. We extract a model for the routing problem and give a new formulation not yet studied. This formulation is motivated by previous research and by considerations given in this work.

We develop an approach based on a subgradient method and extend it to handle the integrality constraints heuristically. We test different variants of the single aspects of the subgradient method. Furthermore, we tackle the problem by a branch-and-cut method which leads us to an exact solution method. By providing test instances similar to real world instances of optical networks we evaluate the developed methods. Both approaches are finally compared to a standard benchmark formulation, that is solved via a straightforward branch-and-cut algorithm. Finally, the obtained results are analyzed and the methods are evaluated.

# Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, New Jersey, 1993.

[2] Filipe Alvelos and J.M. Valerio de Carvalho. Solving the multicommodity flow problem by branch-and-price. Technical report, Departamento de Producao e Sistemas, Universidade do Minho, Braga, Portugal, 2000.

[3] Arjang A. Assad. Models for rail transportation. *Transportation Research*, 14A:205–220, 1980.

[4] Y. Aumann and Y. Rabani. Approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, 27:291–301, 1998.

[5] Mordecai Avriel. *Nonlinear Programming: Analysis and Methods.* Prentice Hall, Englewood Cliffs, N.J., 1976.

[6] E. Balas. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13:517–546, 1965.

[7] Cynthia Barnhart, Christopher A. Hane, and Pamela H. Vance. Integer multicommodity flow problems. In *Integer Programming and Combinatorial Optimization, Proceedings of the 5th International IPCO Conference, Vancouver, British Columbia, Canada*, Lecture Notes in Computer Science, pages 59–71. Springer Verlag, 1996.

[8] Alok Baveja and Aranvind Srinivasan. Approximation algorithms for disjoint paths and related routing and packing problems. *Mathematics of Operations Research*, 25(2):255–280, 2000.

[9] Dimitri P. Bertsekas. *Linear Network Optimization: Algorithms and Codes.* MIT Press, Cambridge, 1991.

[10] P. Bruckner, J.L. Hurink, and T. Rolfes. Routing of railway carriages: A case study. Technical report, University of Twente, 1999.

[11] Lorenzo Brunetta, Michele Conforti, and Matteo Fischetti. A polyhedral approach to an integer multicommodity flow problem. *Discrete Applied Mathematics*, 101(1–3):13–36, 2000.

[12] P.M. Camerini, L. Fratta, and F. Maffioli. On improving relaxation methods by modified gradient techniques. *Mathematical Programming Study*, 3:26–34, 1975.

[13] Teodor Gabriel Crainic, Antonio Frangioni, and Bernard Gendron. Multicommodity capacitated network design. In *Telecommunications Network Planning*, pages 1–19. Kluwer Academic Publisher, 1999.

[14] Teodor Gabriel Crainic, Antonio Frangioni, and Bernard Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design problems. *Discrete Applied Mathematics*, 112(1–3):73–99, 2001.

[15] H. Crowder. *Computational Improvements for Subgradient Optimization*, volume XIX of *Symposia Mathematica*. Academic Press, London, 1976.

[16] R.J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8:250–255, 1965.

[17] Nina K. Detlefsen and Stein W. Wallace. The simplex algorithm for multicommodity networks. *Networks*, 39(1):15–28, 2002.

[18] Lisa K. Fleischer. Approximationg fractional multicommodity flow independent of the number of commodities. *SIAM Journal Discret. Math.*, 13(4):505–520, 2000.

[19] L.R.jun. Ford and D.R. Fulkerson. *Flows in networks.* Princeton, N. J.: Princeton University Press, XII, 194 p. , 1962.

[20] Antonio Frangioni. The MCFClass project. See http://www.di.unipi.it/-di/groups/optimize/Software/MCF.html, 2001.

[21] Antonio Frangioni. The MCFClass project. See http://www.di.unipi.it/-di/groups/optimize/, 2001.

[22] Antonio Frangioni and Giorgio Gallo. A bundle type dual-ascent approach to linear multi-commodity min cost flow problems. Technical Report TR-96-01, Universita di Pisa-Genova-Udine, 1996.

[23] András Frank. Packing paths, circuits and cuts - a survey. In Bernhard Korte, László Lovász, Hans Jürgen Prömel, and Alexander Schrijver, editors, *Paths, flows, and VLSI-Layout*, number 9 in Algorithms and Combinatorics, pages 47–100. Springer-Verlag, 1990.

[24] András Frank, Alexander V. Karzanov, and András Sebö. On integer multiflow maximization. *SIAM J. Discrete Math.*, 10(1):158–170, 1997.

[25] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the Theory of $\mathcal{NP}$-Completeness.* Freeman and Company, N.Y., 1979.

[26] N. Garg, V.V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

[27] Alexander Gersht and Alexander Shulman. A new algorithm for the solution of the minimum cost multicommodity flow problem. In *Proceedings of the 26th Conference on Decision and Control*, pages 748–758, 1987.

[28] Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[29] Ralph E. Gomory. An algorithm for the mixed integer problem. Technical Report RM-2597, The RAND Corporation, 1960.

[30] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Number 2 in Algorithms and Combinatorics. Springer-Verlag, 1988.

[31] Oktay Günlük. A new min-cut max-flow ratio for multicommodity flows. In Andreas S. Schulz William J. Cook, editor, *Integer Porgramming and Combinatorial Optimization. Proceedings of the 9th IPCO conferecnce, MIT Cambridge MA*. Springer, May 2002.

[32] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

[33] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971.

[34] Michael Held, P. Wolfe, and H. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6:62–88, 1974.

[35] Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex Analysis and Minimization Algorithms II.* Number 306 in A Series of Comprehensive Studies in Mathematics. Springer, 1993.

[36] ILOG. ILOG concert technology, version 1.2, 2001.

[37] CPLEX division of ILOG. CPLEX callable library, version 7.5, 2001.

[38] Masao Iri. On an extension of the maximum-flow minimum-cut theorem to multicommodity flows. *J. Operations Res. Soc. Japan*, 13:129–135, 1971.

[39] O. Kakusho and K. Onaga. On feasibility conditions of multicommodity flows in networks. *Transactions on Circuit Theory*, 18:425–429, 1971.

[40] Alexander V. Karzanov. On multicommodity flow problems with integer-valued optimal solutions. *Dokl. Akad. Nauk SSSR*, 280(4), 1985. English translation: Soviet Math. Dokl. Vol. 31, pp. 151–154.

[41] Jeff L. Kennington. A survey of linear cost multicommodity network flows. *Operations Research*, 26(2):209–236, 1978.

[42] Jeff L. Kennington and Mohamed Shalaby. An effective subgradient procedure for minimal cost multicommodity flow problems. *Management Science*, 20(9):994–1004, May 1977.

[43] Ephraim Korach and Michal Penn. Tight integral duality gap in the Chinese postman problem. *Math. Program., Ser. A*, 55(2):183–191, 1992.

[44] Bernhard Korte, László Lovász, Hans Jürgen Prömel, and Alexander Schrijver, editors. *Paths, flows, and VLSI-Layout.* Number 9 in Algorithms and Combinatorics. Springer-Verlag, 1990.

[45] A.H. Land and A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[46] C. Lemaréchal. *Nondifferentiable Optimization*, volume 1, Optimization of *Handbooks in Operations Research and Management Science*, chapter 7, pages 529–572. North Holland, Amsterdam, 1989.

[47] J.D.C. Little, K.G. Murty, D.W. Sweeney, and C. Karel. An alogorithm for the travelling salesman problem. *Operations Research*, 11:972–989, 1963.

[48] Andreas Löbel. Solving large-scale multiple-depot vehicle scheduling problems. In *Wilson, Nigel H. M. (ed.), Computer-aided transit scheduling. Proceedings, Cambridge, MA, USA, August 1997. Lect. Notes Econ. Math. Syst. 471*, pages 193–220. Springer Verlag, Berlin, 1999.

[49] Andreas Löbel. MCF – a network simplex implementation, 2000.

[50] Bin Ma and Lusheng Wang. On the inapproximability of disjoint paths and minimum steiner forest with bandwidth constraints. *Journal of Computer and System Sciences*, 60(1):1–12, 2000.

[51] Richard D. McBride and John W. Mamer. Solving the undirected multicommodity flow problem using a shortest path-based pricing algorithm. *Networks*, 38(4):181–188, 2001.

[52] K. Menger. Zur Allgemeinen Kurventheorie. *Fundamenta Mathematicae*, 10:96–115, 1927.

[53] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley and Sons, N.Y., 1988.

[54] Haruko Okamura. Multicommodity flows in graphs. *Discrete Applied Mathematics*, 6:55–62, 1983.

[55] Haruko Okamura and P.D. Seymour. Multicommodity flows in planar graphs. *J. Comb. Theory, Ser. B*, 31:75–81, 1981.

[56] Asuman E. Ozdaglar and Dimitri P. Bertsekas. Routing and wavelength assignment in optical networks. Technical Report LIDS Report P-2535, Dept. of Electrical Engineering and Computer Schience, M.I.T., Cambridge, Mass., December 2001.

[57] Serge Plotkin and Éva Tardos. Improved bounds on the max-flow min-cut ratio for multicommodity flows. *Combinatorica*, 15(3):425–434, 1995.

[58] B.T. Polyak. Minimization of nonsmooth functionals. *USSR Computational Mathematics and Mathematical Physics*, 9(3):14–29, 1969.

[59] A. Schrijver. *Theory of linear and integer programming.* Wiley, New York, 1986.

[60] András Sebö. Integer plane multicommodity flows with a bounded number of demands. Technical Report 88543-OR, Institut für Operations Research Bonn, 1988.

[61] P.D. Seymour. On odd cuts and planar multicommodity flows. *Proc. Lond. Math. Soc., III. Ser.*, 42:178–192, 1981.

[62] Naum Z. Shor. *On the structure of algorithms for the numerical solution of optimal planning and design problems.* Dissertation, Cybernetics Institute, Academy of Sciences of the Ukrainian SSR, Kiev, 1964.

[63] Martin Skutella. Approximating the single source unsplittable min-cost flow problem. *Mathematical Programming*, 91(3):493–514, 2002.

[64] Aravind Srinivasan. Distributions on level-sets with applications to approximation algorithms. In *Proc. of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 588–597, Las Vegas, Nevada, 2001.

[65] Anand Srivastav and Peter Stangier. On complexity, representation and approximation of integral multicommodity flows. *Discrete Appl. Math.*, 99(1-3):183–208, 2000.

[66] Bram Verweij and Goos Kant Karen I. Aardal. On an integer multicommodity flow problem from the airplane industry. Technical Report UU-CS-1997-38, Department of Computer Science, Utrecht University, 1997.

[67] Dorothea Wagner and Karsten Weihe. A linear-time algorithm for edge-disjoint paths in planar graphs. *Combinatorica*, 15(1):135–150, 1995.

[68] R. Wessäly. *DImensioning Survivable Capacitated NETworks.* PhD thesis, Technische Universität Berlin, 2000.

[69] Laurence A. Wolsey. *Integer Programming.* Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc, N.Y., 1998.

[70] W.I. Zangwill. Non-linear programming via penalty functions. *Management Science*, 13:344–358, 1967.

[71] A. Zymolka, A. M. C. A. Koster, and R. Wessäly. Transparent optical network design with sparse wavelength conversion. ZIB-report 02–34, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Berlin, Germany, 2002.

# Computational results tables

| Instance | $\|V\|$ | $\|E\|$ | $p_G$ | $\|\mathcal{Q}\|$ | $\sum_{i=1}^{q} d_i$ | $\rho$ | Capacities |
|---|---|---|---|---|---|---|---|
| N10_G3_H15_D50_h | 10 | 20 | 0.44 | 15 | 103 | 0.5 | high |
| N10_G3_H15_D50_t | 10 | 20 | 0.44 | 15 | 103 | 0.5 | tight |
| N10_G3_H15_D50_l | 10 | 20 | 0.44 | 15 | 103 | 0.5 | low |
| N10_G3_H40_D50_h | 10 | 26 | 0.58 | 40 | 336 | 0.5 | high |
| N10_G3_H40_D50_t | 10 | 26 | 0.58 | 40 | 336 | 0.5 | tight |
| N10_G3_H40_D50_l | 10 | 26 | 0.58 | 40 | 336 | 0.5 | low |
| N10_G7_H15_D50_h | 10 | 26 | 0.58 | 15 | 146 | 0.5 | high |
| N10_G7_H15_D50_t | 10 | 26 | 0.58 | 15 | 146 | 0.5 | tight |
| N10_G7_H15_D50_l | 10 | 26 | 0.58 | 15 | 146 | 0.5 | low |
| N20_G3_H100_D50_h | 20 | 54 | 0.28 | 100 | 753 | 0.5 | high |
| N20_G3_H100_D50_t | 20 | 54 | 0.28 | 100 | 753 | 0.5 | tight |
| N20_G3_H100_D50_l | 20 | 54 | 0.28 | 100 | 753 | 0.5 | low |
| N20_G3_H100_D75_h | 20 | 55 | 0.29 | 100 | 753 | 0.75 | high |
| N20_G3_H100_D75_t | 20 | 55 | 0.29 | 100 | 753 | 0.75 | tight |
| N20_G3_H100_D75_l | 20 | 55 | 0.29 | 100 | 753 | 0.75 | low |
| N20_G3_H100_D25_h | 20 | 57 | 0.30 | 100 | 753 | 0.25 | high |
| N20_G3_H100_D25_t | 20 | 57 | 0.30 | 100 | 753 | 0.25 | tight |
| N20_G3_H100_D25_l | 20 | 57 | 0.30 | 100 | 753 | 0.25 | low |
| N30_G3_H50_D50_h | 30 | 79 | 0.18 | 50 | 475 | 0.5 | high |
| N30_G3_H50_D50_t | 30 | 79 | 0.18 | 50 | 475 | 0.5 | tight |
| N30_G3_H50_D50_l | 30 | 79 | 0.18 | 50 | 475 | 0.5 | low |
| N30_G3_H150_D50_h | 30 | 86 | 0.20 | 150 | 1216 | 0.5 | high |
| N30_G3_H150_D50_t | 30 | 86 | 0.20 | 150 | 1216 | 0.5 | tight |
| N30_G3_H150_D50_l | 30 | 86 | 0.20 | 150 | 1216 | 0.5 | low |
| N50_G2_H50_D50_h | 49 | 83 | 0.07 | 50 | 385 | 0.5 | high |
| N50_G2_H50_D50_t | 49 | 83 | 0.07 | 50 | 385 | 0.5 | tight |
| N50_G2_H50_D50_l | 49 | 83 | 0.07 | 50 | 385 | 0.5 | low |
| N50_G2_H150_D50_h | 50 | 95 | 0.08 | 150 | 1290 | 0.5 | high |
| N50_G2_H150_D50_t | 50 | 95 | 0.08 | 150 | 1290 | 0.5 | tight |
| N50_G2_H150_D50_l | 50 | 95 | 0.08 | 150 | 1290 | 0.5 | low |
| N76_G2_H20_D50_h | 76 | 113 | 0.04 | 20 | 181 | 0.5 | high |
| N76_G2_H20_D50_t | 76 | 113 | 0.04 | 20 | 181 | 0.5 | tight |
| N76_G2_H20_D50_l | 76 | 113 | 0.04 | 20 | 181 | 0.5 | low |

Table 1: Characteristics of the test instances

| Instance | PROJ | PROJ$^=$ | PEN | BAR | EPEN | BARPEN |
|---|---|---|---|---|---|---|
| N10_G3_H15_D50_h | 1.3 | 1.4 | 12.6 | 14.8 | 5.2 | 11.8 |
| N10_G3_H15_D50_t | 25.3 | 27.2 | 25.7 | 61.2 | 57.3 | 37.1 |
| N10_G3_H15_D50_l | 8.4 | 6.3 | 28.8 | 47.3 | 41.2 | 12.4 |
| N10_G3_H40_D50_h | 1.8 | 1.8 | 7.2 | 21.5 | 9.2 | 7.8 |
| N10_G3_H40_D50_t | 63.1 | 226.0 | 426.9 | 593.2 | 537.3 | 338.0 |
| N10_G3_H40_D50_l | 17.2 | 45.2 | 36.3 | 82.1 | 75.4 | 37.9 |
| N10_G7_H15_D50_h | 1.7 | 2.1 | 3.4 | 12.8 | 3.4 | 7.7 |
| N10_G7_H15_D50_t | 55.8 | 159.8 | 281.6 | 424.4 | 473.1 | 324.3 |
| N10_G7_H15_D50_l | 9.2 | 47.3 | 35.4 | 63.9 | 52.1 | 24.7 |
| N20_G3_H100_D50_h | 9.5 | 108.1 | 221.1 | 223.9 | 201.7 | 172.4 |
| N20_G3_H100_D50_t | 137.4 | 496.4 | 612.7 | 683.8 | 723.2 | 525.8 |
| N20_G3_H100_D50_l | 33.6 | 40.1 | 37.2 | 56.3 | 61.3 | 54.0 |
| N20_G3_H100_D75_h | 35.0 | 42.6 | 210.2 | 194.1 | 173.2 | 161.7 |
| N20_G3_H100_D75_t | 106.6 | 201.9 | 562.8 | 582.2 | 732.6 | 457.1 |
| N20_G3_H100_D75_l | 37.7 | 75.3 | 41.1 | 74.5 | 63.7 | 50.2 |
| N20_G3_H100_D25_h | 13.8 | 47.4 | 218.6 | 267.3 | 261.6 | 252.8 |
| N20_G3_H100_D25_t | 114.8 | 340.6 | 412.3 | 762.7 | 726.4 | 624.7 |
| N20_G3_H100_D25_l | 47.7 | 84.1 | 102.3 | 134.7 | 152.3 | 80.3 |
| N30_G3_H50_D50_h | 7.9 | 123.8 | 177.4 | 229.5 | 204.5 | 124.0 |
| N30_G3_H50_D50_t | 92.8 | 313.2 | 374.3 | 632.8 | 614.2 | 416.4 |
| N30_G3_H50_D50_l | 36.9 | 36.9 | 143.0 | 231.9 | 147.6 | 108.2 |
| N30_G3_H150_D50_h | 30.6 | 60.9 | 578.2 | 627.2 | 572.2 | 493.3 |
| N30_G3_H150_D50_t | 171.1 | 523.7 | 672.7 | 852.2 | 871.6 | 620.5 |
| N30_G3_H150_D50_l | 53.6 | 285.0 | 460.6 | 671.8 | 705.8 | 439.3 |
| N50_G2_H50_D50_h | 17.7 | 20.4 | 205.1 | 266.4 | 247.2 | 183.7 |
| N50_G2_H50_D50_t | 119.3 | 366.4 | 438.3 | 620.6 | 657.9 | 348.3 |
| N50_G2_H50_D50_l | 45.4 | 42.2 | 292.7 | 352.3 | 369.0 | 271.5 |
| N50_G2_H150_D50_h | 55.0 | 174.9 | 647.5 | 624.1 | 638.5 | 572.6 |
| N50_G2_H150_D50_t | 232.7 | 631.0 | 746.7 | 877.4 | 935.0 | 784.8 |
| N50_G2_H150_D50_l | 68.3 | 286.5 | 600.3 | 693.2 | 671.3 | 519.5 |
| N76_G2_H20_D50_h | 5.4 | 73.5 | 184.5 | 158.0 | 235.4 | 158.2 |
| N76_G2_H20_D50_t | 154.5 | 243.2 | 473.0 | 545.2 | 628.1 | 360.6 |
| N76_G2_H20_D50_l | 21.1 | 62.1 | 204.6 | 272.7 | 294.1 | 183.8 |

Table 2: Results of the variants of the subgradient method as percentage above the best solution computed by the branch-and-cut methods

PROJ=projection method, PROJ$^=$=variant of projection method, PEN=penalty method, BAR=barrier method, EPEN=exact penalty method, BARPEN=barrier-penalty method

| Instance | Crowder | $\lambda_1$ | CFM | $\lambda_1$ | mCFM | $\lambda_1$ |
|---|---|---|---|---|---|---|
| N10_G3_H15_D50_h | 1.3 | 3.0 | 1.3 | 3.0 | 1.2 | 2.5 |
| N10_G3_H15_D50_t | 25.3 | 0.5 | 22.7 | 0.5 | 20.8 | 0.5 |
| N10_G3_H15_D50_l | 8.4 | 0.3 | 7.1 | 0.3 | 6.2 | 0.1 |
| N10_G3_H40_D50_h | 1.8 | 3.0 | 3.2 | 2.5 | 1.8 | 2.5 |
| N10_G3_H40_D50_t | 63.1 | 0.5 | 79.6 | 0.5 | 57.3 | 0.3 |
| N10_G3_H40_D50_l | 17.2 | 0.2 | 22.8 | 0.3 | 15.6 | 0.1 |
| N10_G7_H15_D50_h | 1.7 | 3.0 | 4.9 | 3.0 | 1.7 | 2.5 |
| N10_G7_H15_D50_t | 55.8 | 0.5 | 46.4 | 0.5 | 42.2 | 0.3 |
| N10_G7_H15_D50_l | 9.2 | 0.3 | 8.6 | 0.2 | 5.5 | 0.1 |
| N20_G3_H100_D50_h | 9.5 | 2.5 | 10.7 | 3.0 | 9.6 | 3.0 |
| N20_G3_H100_D50_t | 137.4 | 0.3 | 121.3 | 0.3 | 104.9 | 0.1 |
| N20_G3_H100_D50_l | 33.6 | 0.3 | 42.2 | 0.1 | 31.2 | 0.1 |
| N20_G3_H100_D75_h | 35.0 | 4.0 | 25.8 | 3.5 | 23.3 | 3.0 |
| N20_G3_H100_D75_t | 106.6 | 0.3 | 94.4 | 0.3 | 83.5 | 0.1 |
| N20_G3_H100_D75_l | 37.7 | 0.4 | 48.7 | 0.3 | 38.0 | 0.1 |
| N20_G3_H100_D25_h | 13.8 | 2.5 | 24.1 | 3.5 | 11.8 | 3.5 |
| N20_G3_H100_D25_t | 114.8 | 0.5 | 117.0 | 0.3 | 96.4 | 0.1 |
| N20_G3_H100_D25_l | 47.7 | 0.3 | 68.9 | 0.1 | 42.3 | 0.1 |
| N30_G3_H50_D50_h | 7.9 | 1.5 | 15.5 | 2.0 | 13.9 | 2.5 |
| N30_G3_H50_D50_t | 92.8 | 0.5 | 85.8 | 0.1 | 76.2 | 0.1 |
| N30_G3_H50_D50_l | 36.9 | 0.7 | 47.8 | 0.1 | 30.6 | 0.1 |
| N30_G3_H150_D50_h | 30.6 | 3.5 | 49.2 | 4.0 | 47.4 | 3.0 |
| N30_G3_H150_D50_t | 171.1 | 0.3 | 159.9 | 0.3 | 152.8 | 0.1 |
| N30_G3_H150_D50_l | 53.6 | 0.2 | 66.7 | 0.1 | 61.3 | 0.1 |
| N50_G2_H50_D50_h | 17.7 | 1.0 | 19.6 | 3.0 | 18.3 | 1.0 |
| N50_G2_H50_D50_t | 119.3 | 1.0 | 96.3 | 0.3 | 94.2 | 0.1 |
| N50_G2_H50_D50_l | 45.4 | 0.5 | 52.8 | 0.1 | 43.8 | 0.1 |
| N50_G2_H150_D50_h | 55.0 | 1.5 | 63.1 | 2.0 | 53.2 | 4.0 |
| N50_G2_H150_D50_t | 232.7 | 0.5 | 195.8 | 0.3 | 180.4 | 0.1 |
| N50_G2_H150_D50_l | 68.3 | 0.7 | 64.7 | 0.3 | 63.5 | 0.1 |
| N76_G2_H20_D50_h | 5.4 | 4.0 | 6.0 | 3.0 | 8.6 | 3.0 |
| N76_G2_H20_D50_t | 154.5 | 0.5 | 142.6 | 0.5 | 141.4 | 0.1 |
| N76_G2_H20_D50_l | 21.1 | 0.4 | 25.2 | 0.7 | 20.3 | 0.1 |

Table 3: The results of the different subgradient adjustment rules and the initial $\lambda_1$ values used

Crowder=Crowder rule, CFM=Camerini-Fratta-Maffioli rule, mCFM=modified Camerini-Fratta-Maffioli rule

| Instance | Rounding | Convex cost | Integral flow duals |
|---|---|---|---|
| N10_G3_H15_D50_h | 1.2 | 1.2 | 1.2 |
| N10_G3_H15_D50_t | 20.8 | 19.4 | 20.3 |
| N10_G3_H15_D50_l | 6.2 | 6.2 | 6.2 |
| N10_G3_H40_D50_h | 1.8 | 1.8 | 1.8 |
| N10_G3_H40_D50_t | 57.3 | 54.1 | 55.7 |
| N10_G3_H40_D50_l | 15.6 | 15.6 | 15.2 |
| N10_G7_H15_D50_h | 1.7 | 1.7 | 1.7 |
| N10_G7_H15_D50_t | 42.2 | 39.8 | 38.6 |
| N10_G7_H15_D50_l | 5.5 | 5.5 | 5.5 |
| N20_G3_H100_D50_h | 9.6 | 8.4 | 8.4 |
| N20_G3_H100_D50_t | 104.9 | 92.7 | 94.6 |
| N20_G3_H100_D50_l | 31.2 | 30.0 | 28.8 |
| N20_G3_H100_D75_h | 23.3 | 19.3 | 20.4 |
| N20_G3_H100_D75_t | 83.5 | 75.2 | 76.9 |
| N20_G3_H100_D75_l | 38.0 | 35.1 | 34.5 |
| N20_G3_H100_D25_h | 11.8 | 11.8 | 11.8 |
| N20_G3_H100_D25_t | 96.4 | 91.9 | 90.2 |
| N20_G3_H100_D25_l | 42.3 | 39.6 | 37.0 |
| N30_G3_H50_D50_h | 13.9 | 13.9 | 13.9 |
| N30_G3_H50_D50_t | 76.2 | 71.1 | 72.4 |
| N30_G3_H50_D50_l | 30.6 | 29.3 | 27.8 |
| N30_G3_H150_D50_h | 47.4 | 45.7 | 46.0 |
| N30_G3_H150_D50_t | 152.8 | 142.1 | 137.5 |
| N30_G3_H150_D50_l | 61.3 | 59.2 | 57.6 |
| N50_G2_H50_D50_h | 18.3 | 17.1 | 17.6 |
| N50_G2_H50_D50_t | 94.2 | 88.5 | 84.7 |
| N50_G2_H50_D50_l | 43.8 | 38.0 | 39.2 |
| N50_G2_H150_D50_h | 53.2 | 49.9 | 48.8 |
| N50_G2_H150_D50_t | 180.4 | 157.3 | 162.7 |
| N50_G2_H150_D50_l | 63.5 | 56.3 | 57.2 |
| N76_G2_H20_D50_h | 8.6 | 8.6 | 8.6 |
| N76_G2_H20_D50_t | 141.4 | 125.7 | 119.2 |
| N76_G2_H20_D50_l | 20.3 | 18.4 | 17.5 |

Table 4: Comparison of the heuristics for integral solutions

| Instance | GAP | TIME | INIT CUTS | CUTS | NODES |
|---|---|---|---|---|---|
| N10_G3_H15_D50_h | 0.00 | 0 | 113 | 248 | 1 |
| N10_G3_H15_D50_t | 0.00 | 0 | 113 | 321 | 1 |
| N10_G3_H15_D50_l | 0.00 | 2343 | 113 | 2771 | 10274 |
| N10_G3_H40_D50_h | 0.00 | 0 | 319 | 702 | 1 |
| N10_G3_H40_D50_t | 0.00 | 5 | 319 | 1020 | 1 |
| N10_G3_H40_D50_l | 0.42 | 3600 | 319 | 10270 | 1454 |
| N10_G7_H15_D50_h | 0.00 | 0 | 98 | 250 | 1 |
| N10_G7_H15_D50_t | 0.00 | 0 | 98 | 378 | 1 |
| N10_G7_H15_D50_l | 0.00 | 2 | 98 | 766 | 1 |
| N20_G3_H100_D50_h | 0.00 | 51 | 1518 | 3630 | 1 |
| N20_G3_H100_D50_t | 0.00 | 165 | 1518 | 5472 | 1 |
| N20_G3_H100_D50_l | 0.00 | 2139 | 1518 | 19972 | 4 |
| N20_G3_H100_D75_h | 0.00 | 73 | 1501 | 3588 | 1 |
| N20_G3_H100_D75_t | 0.00 | 260 | 1501 | 4173 | 10 |
| N20_G3_H100_D75_l | 0.00 | 1079 | 1501 | 16917 | 1 |
| N20_G3_H100_D25_h | 0.00 | 8 | 1675 | 4145 | 1 |
| N20_G3_H100_D25_t | 0.00 | 221 | 1675 | 5681 | 1 |
| N20_G3_H100_D25_l | 0.00 | 1763 | 1675 | 15237 | 4 |
| N30_G3_H50_D50_h | 0.00 | 5 | 1167 | 2073 | 1 |
| N30_G3_H50_D50_t | 0.01 | 3600 | 1167 | 4357 | 21293 |
| N30_G3_H50_D50_l | 0.01 | 3600 | 1167 | 9138 | 1254 |
| N30_G3_H150_D50_h | 0.00 | 63 | 3363 | 9408 | 1 |
| N30_G3_H150_D50_t | 0.04 | 3600 | 3363 | 14586 | 1027 |
| N30_G3_H150_D50_l | 0.02 | 3600 | 3363 | 10584 | 720 |
| N50_G2_H50_D50_h | 0.00 | 33 | 2435 | 6464 | 1 |
| N50_G2_H50_D50_t | 0.00 | 2027 | 2435 | 10920 | 1 |
| N50_G2_H50_D50_l | 0.02 | 3600 | 2435 | 7140 | 2051 |
| N50_G2_H150_D50_h | 0.00 | 151 | 5738 | 12652 | 1 |
| N50_G2_H150_D50_t | 0.05 | 3600 | 5738 | 15482 | 509 |
| N50_G2_H150_D50_l | 0.02 | 3600 | 5738 | 13580 | 677 |
| N76_G2_H20_D50_h | 0.00 | 23 | 1502 | 4246 | 1 |
| N76_G2_H20_D50_t | 0.00 | 1554 | 1502 | 7569 | 1 |
| N76_G2_H20_D50_l | 0.01 | 3600 | 1502 | 6963 | 2786 |

Table 5: The results of the branch-and-cut method for formulation (RDF)

| Instance | GAP | TIME | ROWS | NODES |
|---|---|---|---|---|
| N10_G3_H15_D50_h | 0.00 | 0 | 300 | 1 |
| N10_G3_H15_D50_t | 0.00 | 0 | 290 | 1 |
| N10_G3_H15_D50_l | 0.00 | 0 | 300 | 1 |
| N10_G3_H40_D50_h | 0.00 | 0 | 756 | 1 |
| N10_G3_H40_D50_t | 0.00 | 0 | 756 | 1 |
| N10_G3_H40_D50_l | 0.24 | 3600 | 756 | 519276 |
| N10_G7_H15_D50_h | 0.00 | 0 | 306 | 1 |
| N10_G7_H15_D50_t | 0.00 | 0 | 306 | 1 |
| N10_G7_H15_D50_l | 0.00 | 0 | 306 | 1 |
| N20_G3_H100_D50_h | 0.00 | 1 | 3874 | 1 |
| N20_G3_H100_D50_t | 0.00 | 5 | 3874 | 1 |
| N20_G3_H100_D50_l | 0.00 | 12 | 3874 | 1 |
| N20_G3_H100_D75_h | 0.00 | 0 | 3877 | 1 |
| N20_G3_H100_D75_t | 0.00 | 5 | 3877 | 2 |
| N20_G3_H100_D75_l | 0.00 | 5 | 3877 | 1 |
| N20_G3_H100_D25_h | 0.00 | 0 | 3875 | 1 |
| N20_G3_H100_D25_t | 0.00 | 2 | 3875 | 2 |
| N20_G3_H100_D25_l | 0.00 | 2 | 3875 | 1 |
| N30_G3_H50_D50_h | 0.00 | 0 | 3009 | 1 |
| N30_G3_H50_D50_t | 0.00 | 2 | 3009 | 1 |
| N30_G3_H50_D50_l | 0.24 | 3600 | 3009 | 48262 |
| N30_G3_H150_D50_h | 0.00 | 2 | 8816 | 1 |
| N30_G3_H150_D50_t | 0.00 | 10 | 8816 | 1 |
| N30_G3_H150_D50_l | 0.10 | 3600 | 8816 | 11258 |
| N50_G2_H50_D50_h | 0.00 | 0 | 4932 | 1 |
| N50_G2_H50_D50_t | 0.00 | 1 | 4794 | 1 |
| N50_G2_H50_D50_l | 0.00 | 35 | 4928 | 4 |
| N50_G2_H150_D50_h | 0.00 | 2 | 14845 | 1 |
| N50_G2_H150_D50_t | 0.00 | 8 | 14784 | 1 |
| N50_G2_H150_D50_l | 0.09 | 3600 | 14845 | 2471 |
| N76_G2_H20_D50_h | 0.00 | 0 | 3035 | 1 |
| N76_G2_H20_D50_t | 0.00 | 1 | 2676 | 1 |
| N76_G2_H20_D50_l | 0.00 | 21 | 2976 | 3 |

Table 6: The results of the branch-and-cut method for formulation (EFF)