Multicriteria Approximation of Network Design and Network Upgrade Problems

Dissertation
zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Bayerischen Julius-Maximilians-Universität
Würzburg

vorgelegt
von **Hans-Christoph Wirth**
aus Naila

Würzburg im März 2001

Bei der Fakultät für Mathematik und Informatik
eingereicht am: 25. März 2001

Erster Gutachter: Prof. Dr. Hartmut Noltemeier, Würzburg
Zweiter Gutachter: Prof. Dr. Victor Chepoi, Marseille

Tag der mündlichen Prüfung: 27. Juli 2001

Contents

Preface

Network planning has come to great importance during the past decades. Today's telecommunication, traffic systems, and logistics would not have been evolved to the current state without careful analysis of the underlying network problems and precise implementation of the results obtained from those examinations. Graphs with node and arc attributes are a very useful tool to model realistic applications, while on the other hand they are well understood in theory.

We investigate network design problems which are motivated particularly from applications in communication networks and logistics. On the other hand, we use node and edge upgrade models to deal with the fact that in many cases one prefers to change existing networks rather than implementing a newly computed solution from scratch. All problems are examined within a framework of multi-criteria optimization.

Many of the problems can be shown to be NP-hard, with the consequence that, under the widely accepted assumption that P is not equal to NP, there cannot exist efficient algorithms for solving the problems. This motivates the development of approximation algorithms which compute near-optimal solutions with provable performance guarantee in polynomial time.

*

This thesis is organized as follows: In Chapter 1 we give a short overview of the notational framework used in the thesis. Moreover, we give a brief recapitulation of some important results this work relies on. Finally we refer the reader to a limited and exemplary list of monographs.

In Chapter 2 we start the part on network design problems with investigating the *minimum label spanning tree* problem: This problem is specified by a graph with edge labels, and the goal is to construct a spanning tree with minimum number of different labels. We also provide results on the analogous problem on paths, namely the *minimum label path* problem, and on further related problems.

Chapter 3 introduces the *reload cost model*: given an edge labeled graph, costs arise at nodes and depend on the pair of labels used by the traversal through that node. We investigate the problems of finding a *shortest path* and a spanning tree of *minimum radius* and *minimum diameter* with respect to reload costs.

In Chapter 4 we close the part on network design problems with providing results on the *dial a ride* problem. This problem is defined on a graph with an additional set of arcs specifying transportation requests. The goal is to find a shortest tour which traverses all arcs. We investigate the complexity for several graph classes and provide approximation algorithms.

Chapter 5 starts with the introduction of the node upgrade model used in the sequel for defining the *upgrading bottleneck constrained forest* problem. The problem consists in finding an optimal (budget constrained) node improvement strategy such that the resulting bottleneck graph admits a constrained forest. Constrained forests generalize well known design problems including the search for paths, spanning trees and Steiner trees.

The part on network upgrade problems is concluded with Chapter 6 where we investigate several *flow problems under an edge based upgrade model*. All problems are related to the well known maximum flow and minimum cost flow problems. The upgrade model admits to modify the edge capacity, the unit flow cost, or both. The goal is to optimize the total flow cost, the total flow value, and the upgrade budget.

Acknowledgements

At this point I want to thank all the persons who have contributed to this work. First of all, I would like to thank my advisor, Prof. Dr. H. Noltemeier, who has patiently guided me during all phases of this work and made inspiring suggestions for orienting the line of research. My thanks also go to all of my colleagues, in particular S. O. Krumke, O. Karch, I. Demgensky, and J. Steffan, who have spent numerous hours of fruitful discussions on all parts of this work and far beyond. Special thanks go also to M. V. Marathe and S. S. Ravi for giving me the excellent chance to do research together. Further I am thankful to the German research foundation "Deutsche Forschungsgemeinschaft" (DFG) for supporting large parts of my work.

Credits

The work on the minimum label spanning tree in Chapter 2 is a joint work with S. O. Krumke [KW98]. The results on the edge labeled paths reported in

that chapter were obtained jointly with R. Jacob, G. Konjevod, S. O. Krumke, M. V. Marathe, R. Ravi, and T. Samstag [JK+99]. The results from Chapter 3 on the minimum reload cost diameter spanning tree problem are based on joint work with J. Steffan [Ste99, WS99, WS01]. The work on the dial a ride problem in Chapter 4 was done together with D. Hauptmeier, S. O. Krumke, and J. Rambau [HK+99, HK+01]. The results from Chapter 5 on the node upgrade constrained forest problem are obtained jointly with S. O. Krumke, M. V. Marathe, H. Noltemeier, and S. S. Ravi [KM+01]. The work in Chapter 6 on arc upgrade flow problems is based on co-operative research with I. Demgensky, S. O. Krumke, H. Noltemeier, R. Ravi, and S. Schwarz [KN+99, Dem00, DNW00].

I am greatly thankful to all of the co-authors for many hours of inspiring discussions and for allowing me to include the work in this thesis.

Chapter 1

Introduction

This chapter is intended mainly as a reference for the notations used in this thesis and the foundations this work relies on. We assume that the reader is familiar with elementary graph theory, graph algorithmic concepts, and combinatorial optimization as well as with basic results from complexity theory. For detailed reviews we refer the reader to monographs and textbooks which are listed at the end of this chapter.

1.1 Definitions

Basics

By $\mathbb{N} := \{1, 2, \ldots\}$ we denote the set of **natural numbers**, and we define $\mathbb{N}_0 := \{0\} \cup \mathbb{N}$. Set $\mathbb{N}_0$ is also called the set of **integer numbers**. $\mathbb{Q}$ and $\mathbb{R}$ denote the set of **rational numbers** and **real numbers**, respectively. We write $\mathbb{Q}^+ := \{\, q \in \mathbb{Q} \mid q > 0 \,\}$ and $\mathbb{Q}_0^+ := \{0\} \cup \mathbb{Q}^+$. Sets $\mathbb{R}^+$ and $\mathbb{R}_0^+$ are defined analogously.

The rounding of real numbers $x \in \mathbb{R}_0^+$ is denoted by the common notation $\lfloor x \rfloor := \max\{\, n \in \mathbb{N}_0 \mid n \leq x \,\}$ and $\lceil x \rceil := \min\{\, n \in \mathbb{N}_0 \mid n \geq x \,\}$.

By 2^S we denote the **power set** of a set S, which is the set of all subsets of set S (including the empty set $\varnothing$ and S itself).

Sets and Multi-sets

A **multi-set** X over a ground set U, denoted by $X \sqsubset U$, is defined as a mapping $X\colon U \to \mathbb{N}_0$, where for $u \in U$ the number $X(u)$ denotes the multiplicity of u in X. We write $u \in X$ if $X(u) \geq 1$. If $Y \sqsubset U$ then $X \sqsubset Y$ denotes a multi-set over the ground set $\{\, u \in U : Y(u) > 0 \,\}$.

If $X \sqsubset U$ and $Y \sqsubset U$ are multi-sets over the same ground set U, then we denote by $X + Y$ their **multi-set union**, by $X - Y$ their **multi-set difference** and by $X \cap Y$ their **multi-set intersection**, defined for $u \in U$ by

$$\begin{aligned}(X + Y)(u) &:= X(u) + Y(u)\\ (X - Y)(u) &:= \max\{X(u) - Y(u), 0\}\\ (X \cap Y)(u) &:= \min\{X(u), Y(u)\}.\end{aligned}$$

The multi-set $X \sqsubset U$ is a **subset** of the multi-set $Y \sqsubset U$, denoted by $X \subseteq Y$, if $X(u) \leq Y(u)$ for all $u \in U$. We denote the **cardinality** of a multi-set $X \sqsubset U$ by $|X| := \sum_{u \in U} X(u)$. For a weight function $c\colon U \to \mathbb{R}$ the **weight** of a multi-set $X \sqsubset U$ is defined by $c(X) := \sum_{u \in U} c(u)X(u)$.

Any standard set can be viewed as a multi-set with elements of multiplicity 0 and 1. If X and Y are two standard sets with $X \subseteq Y$ and $X \neq Y$, then X is a **proper subset** of Y, denoted by $X \subset Y$. Two subsets $X_1 \subseteq Y$, $X_2 \subseteq Y$ of a standard set Y form a **partition** of Y, if $Y = X_1 \cup X_2$ and $X_1 \cap X_2 = \varnothing$.

Graphs

An **undirected graph** is denoted by $G = (V, E)$. Here, $V(G) := V$ specifies the set of **nodes** or **vertices** and $E(G) := E$ specifies the set of **edges**. Throughout the thesis we assume that both V and E are finite, and we write $n := |V|$ and $m := |E|$ to denote the cardinalities. An **edge** $e \in E$ with two **endpoints** $v, w \in V$ is denoted by the unordered pair $e = (v, w)$ of nodes. Two edges with identical pairs of endpoints are called **parallel**. An edge with two coinciding endpoints is a **loop**. A graph without parallels and without loops is called **simple**.

For $e = (v, w)$, we say that edge e is **incident** with nodes v and w, while nodes v, w are **adjacent** to each other. For $v \in V$, the set $N(v)$, called the **neighborhood** of v, contains all adjacent nodes including node v itself. The number of edges incident with a node v is called the **degree** of node v and denoted by $d(v)$.

A **directed graph** is denoted by $G = (V, R)$, where R is the set of **arcs**. An arc with **source** $v \in V$ and **target** $w \in V$ is denoted by the ordered pair (v, w) of nodes. This arc **emanates** from v and **reaches** w. We write $(v, w)^{-1} := (w, v)$ to denote an **anti-parallel** arc. The **neighborhood** $N^+(v)$ of node v contains all nodes which are reached by an arc emanating from v. The **out-degree** $d^+(v)$ of node v is the number of arcs emanating from v, the **in-degree** $d^-(v)$ is the number of arcs reaching v. The **degree** of v is defined by $d(v) := d^+(v) + d^-(v)$. A graph is called **degree balanced** if $d^+(v) = d^-(v)$ for all of its vertices v. The remaining notions are defined analogously to the case of undirected graphs.

A **mixed graph** is a special type of graph which has undirected edges and directed arcs at the same time. A mixed graph $G = (V, E, A)$ consists of a set V of vertices, a set E of undirected edges without parallels, and a multi-set A of directed arcs (parallel arcs allowed). In order to be able to distinguish notationally between edges and arcs, we use $[u, v]$ to denote an (undirected) **edge** with endpoints u and v, and we use (u, v) to denote a (directed) **arc** from u to v. (The notational conventions for edges in mixed graphs are in contrast to the case of undirected graphs. In this thesis they apply exclusively to the work in Chapter 4.) We denote by $n := |V|$, $m_E := |E|$ and $m_A := |A|$ the number of vertices, edges and arcs, respectively. For $v \in V$ we let $A_v \subseteq A$ be the **set of arcs emanating from** v. For edge set E, denote by

$$E^{\rightleftarrows} := \{ (u, v), (v, u) : [u, v] \in E \} \tag{1.1}$$

the set of arcs which contains for each undirected edge $e \in E$ a pair of anti-parallel arcs between the endpoints of e.

Where not stated otherwise we assume throughout the thesis that graphs are undirected and simple.

Subgraphs

A graph $H = (V_H, E_H)$ is a **subgraph** of graph $G = (V, E)$, if $V_H \subseteq V$ and $E_H \subseteq E$. In this case we write $H \sqsubset G$.

Let $G = (V, E)$ be a graph, assume $V' \subseteq V$ and $E' \subseteq E$. By $G[E'] := (V, E')$ we denote the subgraph **induced by edge set** E'. The graph $G[V']$ **induced by node set** V' consists of node set V' and contains all those edges from E with both endpoints in V'.

The degree of a node v in a subgraph H is denoted by $d_H(v)$. For shorter notation we use $d_{E'}$ and $d_{V'}$ instead of $d_{G[E']}$ and $d_{G[V']}$ for degrees in induced subgraphs.

A graph $G = (V, E)$ is called **complete** if the set of its edges consists of all pairs of different nodes. A complete subgraph is called a **clique**.

A **path** p in a graph $G = (V, E)$ is defined to be an alternating sequence $p = (v_1, e_1, v_2, \ldots, e_k, v_{k+1})$ of nodes $v_i \in V$ and edges $e_i \in E$, where for each triple (v_i, e_i, v_{i+1}) we have $e_i = (v_i, v_{i+1})$. (For directed graphs $G = (V, R)$, edges are replaced by arcs, and we require $r_i = (v_i, v_{i+1})$ and $r_i \in R \cup R^{-1}$ for each triple. If the stronger condition $r_i \in R$ holds, the path is called **directed**. For mixed graphs, we define a **walk** which traverses arbitrarily edges and directed arcs.) We use equivalently the alternative notations $p = (v_1, v_2, \ldots, v_{k+1})$ and $p = (e_1, e_2, \ldots, e_k)$ when the meaning is clear. Nodes v_1 and v_{k+1} are the **start**

point and **end point** of p. If all nodes of the path or walk are pairwise different (without considering the pair v_1, v_{k+1}), the path or walk is called **simple**. A path or walk with coincident start and endpoint is **closed**. A closed and simple path is a **cycle**.

A (directed or undirected) graph is called **connected** if it contains for each pair v, w of nodes a path between v and w. A directed graph is called **strongly connected** if it contains for each pair v, w of nodes a directed path from v to w and vice versa. A subgraph is called maximal [strongly] connected, if it cannot be enlarged by edges or nodes without losing the [strong] connectivity. A maximal [strongly] connected subgraph is also called [**strongly**] **connected component**.

A **series-parallel graph** is a directed graph $G = (V, R)$ with two **terminals**, the **source** and the **target** (both terminals are distinct nodes from V) which admits to be defined recursively as follows.

A basic series-parallel graph is the graph with vertex set $\{s, t\}$ and arc set $\{(s, t)\}$. Node s is the source, node t the target. Assume that $G_1 = (V_1, R_1)$ and $G_2 = (V_2, R_2)$ are two series-parallel graphs (where $V_1 \cap V_2 = \varnothing$), with sources s_1, s_2 and targets t_1, t_2, respectively. Then

- the graph obtained by identifying t_1 and s_2 is series-parallel with source s_1 and target t_2. This graph is the **series composition** of G_1 and G_2.
- the graph obtained by identifying s_1 with s_2 and t_1 with t_2 is series-parallel with source s_1 and target t_1. This graph is the **parallel composition** of G_1 and G_2.

Series-parallel graphs are a special case of *decomposable graphs* [BLW87]. Decomposable graphs are specified by a finite set of primitive graphs, where each graph has an ordered set of terminal nodes. Moreover, there is a finite set of binary composition rules operating only on the terminals of the two graphs being composed. This definition can be applied to both undirected and directed graphs.

A **tree** is a connected subgraph with no cycles. A node in a tree is called a **leaf** if its degree equals 1, and an **inner node** otherwise. A **forest** is a set of pairwise node-disjoint trees. A subgraph $H \sqsubset G$ is called **spanning subgraph** if H is connected and $V(H) = V(G)$. A **directed spanning tree rooted towards** a node $o \in V$ is a spanning tree of a directed graph which contains for each node $v \in V$ a directed path from v to o.

A **star** is a tree with at most one inner node, the **center** of the star. A **spider** is a tree with at most one node of degree greater than 2.

A **caterpillar graph** is a special case of a tree where the graph induced by the set of inner nodes is a path. This path is called the **backbone** of the caterpillar, the remaining nodes (i. e., the leaves) are also called the **feet** of the caterpillar. The edges between the backbone and the feet are called **hairs**. We restrict the class of caterpillars further to those graphs where no two hairs are incident, i. e., the nodes on the backbone are of maximum degree 3.

Graphs with Attributes

Graphs can be attributed with functions defined on the set of edges or nodes of various meaning. Those functions are introduced and described in this thesis at the point where they come into need.

A commonly used attribution of a graph $G = (V, E)$ is a **cost** or **weight** function $l\colon E \to \mathbb{R}$ defined on the set of edges. Usually the total weight of a subgraph $H \sqsubset G$ is calculated by $l(H) := \sum_{e \in E(H)} l(e)$. If p is a path, then $l(p)$ denotes the **length** of the path.

Let $G = (V, R)$ be a directed graph with two distinguished nodes $s, t \in V$. A function $x\colon R \to \mathbb{R}_0^+$ is called a **flow**, if for each node $v \in V \setminus \{s, t\}$ the inflow $x^-(v) := \sum_{r \in R \text{ with target } v} x(r)$ equals the outflow $x^+(v) := \sum_{r \in R \text{ with source } v} x(r)$. The net outflow $x^+(s) - x^-(s)$ of the source is called the **value** of flow x and denoted by $F(x)$. Assume that there is an arbitrary function $u\colon R \to \mathbb{R}_0^+$, called the **capacity** function. A flow x is called **feasible** if $0 \le x(r) \le u(r)$ for each arc $r \in R$.

Some models introduce also a lower capacity function l on the set of arcs and define feasibility by $l(r) \le x(r) \le u(r)$ for each arc r. Since there are standard techniques to reduce to the case $l \equiv 0$ (see [AMO93]) we silently assume $l \equiv 0$ throughout the thesis.

1.2 Complexity of Computation

When working with computational problems there arise two main goals: The first goal is to find a solution. The second (and maybe the even more important) goal is to do this efficiently. This implies the need for a classification of algorithms with respect to their efficiency and a classification of problems according to their difficulty and complexity.

Measuring the Running Time

A natural way to determine the complexity of a problem is to measure the time needed by an algorithm which computes a solution. This measure is dependent on many factors including the applied computational model and, as a consequence thereof, the representation of problem instances.

In complexity theory the *Turing machine* [GJ79] is widely used as a model of computation. This plain model is unwieldy for the aims of algorithmic graph theory. There are models which are more applicable from this practical point of view, e. g. the model of a *random access machine* or an abstract *Pascal machine* [Pap94, AC+99]. These models admit to use a high-level programming language for formulating algorithms. Since we are more interested in the complexity of the problems itself rather than in the efficiency of a particular implementation, we decided not to specify and use a concrete programming language, but we formulate our algorithms using a pseudo-code which can easily be compiled into a programming language if desired. The main reason for this decision is that the presentation of algorithms can be cleared and simplified. Of course one must be careful that complex tasks are not hidden by the notation.

We briefly reflect the arguments of [AC+99]. The running time of an algorithm is basically the number of instructions the abstract processor performs until the calculation is complete. The set of instructions includes assignments and basic numerical operations on the numbers stored in the machine variables. From a strict point of view, the duration of execution of a single statement depends on the size of the operands, i. e., the number of bits which is logarithmic in the size of the numbers. This suggests to employ the *log-cost model*. Albeit, it turns out that as long as numbers keep "sufficiently small" (to be more concrete, the size of numbers must be bounded by a polynomial in the size of the input instance, see below) during the execution, the extra time does not affect the measured effort substantially. Since the above condition is true for most applications (and in particular, for the problems investigated in this thesis), it is acceptable to use the easier to analyze *unit-cost model* where the execution cost of a numerical operation does not depend on the size of the operands.

In order to characterize the complexity of a particular problem one must consider all instances to this problem. A convenient way to do the characterization is to specify the *running time function* of an algorithm. Such a function measures the running time of an algorithm as a function of the size of the instance. This size is measured in the number of bits which are needed for storing the instance.

This leads immediately to the question how to represent an instance of a particular problem. One must assure that the representation cannot be padded arbitrarily with redundant information, because in this case the specification of

a running time function would become useless. For graphs, reasonable encoding schemes include the representation as an incidence matrix or as adjacency lists. Here all graph items are represented by a unique number which is binary encoded. (See e. g. [CLR90] for a further discussion of graph representations.) As a result, the size of a reasonable representation of a graph $G = (V, E)$ is linear in $|V|$ and $|E|$.

A key observation [AC+99] is, that all "reasonable" encoding schemes are *polynomially equivalent*, i. e., for arbitrary reasonable encodings there exists a polynomial such that the size of an instance coded according to one scheme is bounded by the polynomial in the size of the other encoding and vice versa. A similar observation holds for the machine models and languages used for describing algorithms. Coming back to the discussion of the cost model from above, this observation is the reason why numbers of polynomially bounded size can be treated by a unit cost model.

For graph algorithms it is conventional to specify the running time as a function depending on $|V|$, i. e., the number of nodes of the graph, rather than to use a function of the length of the encoding. This is admissible due to the above considerations: As long as we decide for reasonable encodings, the length of the encoding is bounded by a polynomial in $|V|$. Hence if an algorithm is said to have *polynomial running time*, is does not matter whether to use $|V|$ or the length of the encoding as the argument of the polynomial. As a consequence, if the running time function is given in terms of $|V|$, the results are independent of the particular machine model and graph encoding scheme. This enables to compare algorithms with respect to their running time.

We close this section with a remark on the encoding of numbers. Integer numbers are usually encoded binary (or k-ary for alphabets of cardinality k). As considered above, the length of the encoding of each number must be bounded by a polynomial $p(|V|)$. Consequently, the range of integer numbers which are specified by any input instance is restricted to the interval $[0, 2^{p(|V|)}]$ for some polynomial p. This restriction is tacitly assumed in most of the contributions to the complexity of graph algorithms in literature, and this is true also for the current thesis.

Non-integer numbers cannot be stored with arbitrary precision under the above restrictions. Usually, those numbers are approximated by a rational number. Rational numbers are stored as a pair of integer numbers. In most cases (in particular this holds for the problems considered in this thesis), one can transform an instance with rational numbers into an equivalent instance with integer numbers by multiplying all numbers by the lowest common denominator. One can observe that this operation retains the polynomial bound on the encoding

size of all numbers. Thus, whenever it is convenient, we can assume without loss of generality that all numbers of an input instance are integer. For a further discussion of some subtleties involved by division operations with rational numbers in several machine models we refer to [GLS88].

Asymptotic Running Time

A widely used tool for characterizing the running time complexity of an algorithm is the *worst case analysis*: The running time is specified by a function f, where the value $f(n)$ is an upper bound on the running time of the algorithm for each instance with size n. Alternative approaches include the *average case* analysis which is usually more complex to perform and has applications different from the aims of this thesis.

Running time functions are usually denoted using the O-*notation*. This makes the results independent of constant factors which may be solely a consequence of special capabilities of the particular machine model or programming language. We briefly introduce this notation: Let $F = \{f\colon \mathbb{N} \to \mathbb{R}\}$ be the set of all real valued functions on $\mathbb{N}$. For $f \in F$, define

$$
\begin{aligned}
O(f) &:= \{\, g \in F \mid \exists a \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : \forall n \in \mathbb{N}, n > n_0 : |g(n)| \leq a \cdot f(n) \,\} \\
\Omega(f) &:= \{\, g \in F \mid \exists a \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : \forall n \in \mathbb{N}, n > n_0 : g(n) \geq a \cdot |f(n)| \,\} \\
\Theta(f) &:= O(f) \cap \Omega(f) \\
o(f) &:= \{\, g \in F \mid \forall a \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : \forall n \in \mathbb{N}, n > n_0 : a \cdot |g(n)| < f(n) \,\}
\end{aligned}
$$

In literature there can be found an alternative definition of $g \in o(f)$, namely $\lim_{n\to\infty} g(n)/f(n) = 0$. It is easy to observe that this definition is equivalent (as long as the functions do not attain negative values). Moreover, for a subset $F' \subset F$ we define $O(F') := \cup_{f\in F'} O(f)$.

Notice that for arbitrary constants $a, b > 1$, we have $\Theta(\log_a) = \Theta(\log_b)$. Therefore we basically omit the base of the logarithm function in all arguments of the O-notation.

A very fast growing function is *Ackerman's function* $A\colon \mathbb{N}^2 \to \mathbb{N}$ (see [CLR90]), which is defined recursively by

$$
\begin{aligned}
A(1, j) &:= 2^j && \text{for } j \geq 1\,, \\
A(i, 1) &:= A(i-1, 2) && \text{for } i \geq 2\,, \\
A(i, j) &:= A(i-1, A(i, j-1)) && \text{for } i, j \geq 2\,.
\end{aligned}
$$

By defining

$$\alpha(m, n) := \min\{\, i \in \mathbb{N} \mid A(i, \lfloor m/n \rfloor) > \log_2 n \,\}$$

one introduces a function $\alpha\colon \mathbb{N}^2 \to \mathbb{N}$, called the *inverse of Ackerman's function*. Here the notion "inverse" must be taken with a grain of salt: The function α is not the inverse in the true mathematical sense, but it growths as slow as Ackerman's function grows fast.

Complexity classes

We briefly note some few complexity classes which are defined for decision problems. A *decision problem* is a problem where each instance has only one of two outcomes from the set $\{\text{yes}, \text{no}\}$. For a function $f\colon \mathbb{N} \to \mathbb{N}$, the class $\text{DTIME}(f(n))$ [$\text{NTIME}(f(n))$] contains all problems which can be decided on a deterministic [nondeterministic] Turing machine within time $O(f)$. There are two important classes known from complexity theory, defined by

$$P := \bigcup_{k=1}^{\infty} \text{DTIME}(n^k) \qquad \text{and} \qquad NP := \bigcup_{k=1}^{\infty} \text{NTIME}(n^k)\,.$$

A decision problem π is called *NP-complete*, if it is a member of NP and every other problem of NP can be reduced to π in polynomial time. This means, for each problem $\pi' \in \text{NP}$ there is a polynomial time computable function $f_{\pi'}$ mapping instances of π' to instances of π with the property that for each instance x' of π', x' is a yes-instance of π' if and only if $f(x')$ is a yes-instance for π. [GJ79] Many interesting decision problems are NP-complete. Under the widely beliefed assumption $P \neq NP$ this has the consequence that those problems are *intractable*, i. e., there is no deterministic algorithm deciding them in polynomial time.

1.3 Optimization Problems

1.3.1 Uni-criterion Optimization

Following [AC+99], an *optimization problem* π is characterized by a quadruple $\pi = (I, \text{SOL}, m, g)$ where the objects have the following meaning: I specifies the set of instances of problem π, and SOL is a function on I which yields the set of feasible solutions to the problem. Measure function m assigns a positive integer value to each feasible solution, and g specifies whether problem π is a maximization or a minimization problem. By $\text{SOL}^*(x)$ one denotes the set of *optimal solutions* for instance $x \in I$. The value $m(y)$ of all optimal solutions $y \in \text{SOL}^*(x)$ is also denoted by $m^*(x)$.

Each optimization problem π induces naturally the following family of three problems:

- *constructive problem* π_C*:* Given an instance $x \in I$, derive an optimal solution $y \in SOL(x)$ and its measure $m(y)$.
- *evaluation problem* π_E*:* Given an instance $x \in I$, derive its measure $m^*(x)$.
- *decision problem* π_D : Given an instance $x \in I$ and an integer $K \in \mathbb{N}$, decide whether $m^*(x) \geq K$ (for maximization problem) or whether $m^*(x) \leq K$ (for minimization problem).

Optimization problems in algorithmic graph theory are usually *constructive problems* in this sense: We are always interested both in optimizing the measure function (albeit the actual value of the measure function is usually not computed explicitly) and in constructing a solution which implements this value.

A first rough characterization of optimization problems can be performed similarly to the situation for decision problems, where classes P and NP have been introduced. For optimization problems, one uses another view at nondeterminism [AC+99]. The class NPO is defined to contain all optimization problems where the set of instances can be recognized in polynomial time, and a feasible solution can be guessed and verified (and the measure function can be evaluated) in polynomial time. The subclass PO of NPO contains all optimization problems where a deterministic algorithm exists which constructs for each instance an optimal solution in polynomial time.

One can show that for each optimization problem in NPO, the corresponding decision problem is in NP. Moreover, the notion of hardness can be extended to optimization problems: An optimization problem is called *NP-hard*, if the corresponding decision problem is NP-complete. Since the common assumption $P \neq NP$ implies also $PO \neq NPO$ (see [AC+99]), all optimization problems which can be shown to be NP-hard are considered as intractable.

Approximation Algorithms

In order to deal with the intractability of NP-hard optimization problems, it is useful to design *approximation algorithms* [AC+99]. In a general sense, an approximation algorithm is a polynomial time algorithm which produces for any instance $x \in I$ of an optimization problem an arbitrary feasible solution $y \in SOL(x)$.

Let $x \in I$ be an instance of an optimization problem and $y \in SOL(x)$ be an approximate solution. We use the ***performance*** *ratio* $R(x, y)$

$$R(x, y) := \max \left\{ \frac{m(y)}{m^*(x)}, \frac{m^*(x)}{m(y)} \right\}$$

to estimate the quality of y. Other approaches include measuring the *absolute error* $D(x, y)$ and the *relative error* $E(x, y)$, defined by

$$D(x, y) := |m(y) - m^*(x)| \qquad \text{and} \qquad E(x, y) := \frac{|m(y) - m^*(x)|}{\max\{m^*(x), m(y)\}}.$$

Notice that the performance ratio satisfies always $R(x, y) \geq 1$. Moreover, $R(x, y) = 1$ if and only if y is an optimal solution. At this point one defines:

Definition 1.1 (Approximation Algorithm, Performance Guarantee)
Let π be an optimization problem and A be a (deterministic) polynomial time algorithm for π. Algorithm A is called ***approximation algorithm*** *for π* ***with performance*** *α (or α-**approximation algorithm**), if for each instance x, the algorithm outputs a solution $A(x) \in \mathrm{SOL}(x)$ with*

$$R(x, A(x)) \leq \alpha.$$

In this definition, α may be a constant or a function depending on the input instance. Observe that this definition relies on a worst case analysis.

Complexity Classes

There are some important subclasses in NPO according to the benevolence of NPO problems with respect to their approximability [AC+99]. Class APX contains all NPO problems where for some constant $r > 1$ there exists an r-approximation algorithm. Class PAS contains all problems for which there is a *polynomial approximation scheme*, i. e., there is an algorithm which for each $\varepsilon > 0$ is an approximation algorithm with performance $1 + \varepsilon$. If the running time of this algorithm is also polynomial in $1/\varepsilon$, the scheme is called *fully polynomial approximation scheme*, and the resulting class is denoted by FPAS. The relation between the problem classes is reflected by the chain of inclusions

$$\mathrm{PO} \subseteq \mathrm{FPAS} \subseteq \mathrm{PAS} \subseteq \mathrm{APX} \subseteq \mathrm{NPO}$$

which can be proven to be strict if and only if $\mathrm{P} \neq \mathrm{NP}$. Observe that showing the existence of an FPAS for an optimization problem is the best approximation result one can achieve if this problem is NP-hard.

1.3.2 Multi-criteria Optimization Problems

In this thesis we are also interested in optimization problems π which are characterized by more than one measure function. In this case we have a tuple $m = (m_1, \ldots, m_k)$ of $k > 1$ measure functions (also called *objective* functions)

and a corresponding tuple $g = (g_1, \ldots, g_k)$ of goals. There are different ways to define optimality for multi-criteria problems, since it is not clear in general how to solve the conflict between concurrent objective functions.

We use the following approach in our work: We constrain all but one of the objective functions by a numerical bound, and optimize the remaining objective subject to those constraints. Other approaches include the search for *lexicographic optimal* solutions and for the set of *pareto optimal* solutions: a solution is pareto optimal (also called *efficient*), if there is no other solution which dominates it in *all* objectives at the same time.

In the context of the definitions from [AC+99], the approach of bounding the objectives can be modeled as follows: The set I of problem instances has the form $I = I' \times (\mathbb{R}_0^+)^{k-1}$. As before, there is a function which maps each instance $x \in I'$ to a set of solutions, we call this function Γ. This function is augmented to I by defining $\Gamma(x, B) := \{ (y, B) \mid y \in \Gamma(x) \}$. A function $FS_{m',g'}$ on the set of solutions selects *feasible solutions*. This function is designed to reflect measure vector $m' = (m_2, \ldots, m_k)$ and goal vector $g' = (g_2, \ldots, g_k)$, and is defined by

$$FS_{m',g'}(Y, B) = \left\{ y \in Y \;\middle|\; \begin{array}{ll} m_i(y) \leq B_i, & \text{for } 2 \leq i \leq k \text{ where } g_i = \min, \\ m_i(y) \geq B_i, & \text{for } 2 \leq i \leq k \text{ where } g_i = \max. \end{array} \right\}$$

where $B = (B_2, \ldots, B_k)$ is the tuple of $k - 1$ numerical constraints specified by the instance. With the composition $SOL := FS_{m',g'} \circ \Gamma$, the overall problem $\pi = (I, SOL, m_1, g_1)$ is now a uni-criterion optimization problem with measure function m_1 and goal g_1.

Since the definitions for SOL^*, m^*, and the performance ratio $R(x, y)$ can easily be extended from I' to I, also the definition of an α-approximation algorithm carries over to this case.

For many multi-criteria optimization problems it turns out that, given an instance x, even the problem of deciding whether $FS(\Gamma(x, B))$ is nonempty is NP-complete. This defers to construct an efficient approximation algorithm in the sense described above. We show how to dodge this situation. To this end, one introduces a set of *almost feasible solutions*. Let $\alpha' = (\alpha_2, \ldots, \alpha_k)$ be a *performance* vector of numbers $\alpha_i \geq 1$. Similar to above, let $AFS_{\alpha',m',g'}$ be a selection function on the set of solutions which is defined by

$$AFS_{\alpha',m',g'}(Y, B) = \left\{ y \in Y \;\middle|\; \begin{array}{ll} m_i(y) \leq \alpha_i \cdot B_i, & \text{for all } i \text{ where } g_i = \min, \\ m_i(y) \geq 1/\alpha_i \cdot B_i, & \text{for all } i \text{ where } g_i = \max. \end{array} \right\}$$

An almost feasible solution violates the constraints on the optimization objectives in a controlled fashion.

Definition 1.2 (Multi-criteria Approximation Algorithm, Performance)
Let π be an optimization problem with k objectives and A be a (deterministic) polynomial time algorithm. Algorithm A is called ***multi-criteria approximation algorithm*** *for π* ***with performance*** *$(\alpha_1, \ldots, \alpha_k)$, if for each instance (x, B) where $FS \circ \Gamma(x, B) \neq \varnothing$, the algorithm outputs a solution*

$$A(x, B) \in AFS \circ \Gamma(x, B)\,,$$

and, besides, each solution $A(x, B)$ output satisfies

$$R(x, A(x, B)) \leq \alpha_1\,.$$

This definition implies that, for the case $FS \circ \Gamma(x, B) = \varnothing$, the algorithm has two choices: either it outputs the undefined solution $\perp$ meaning "there is no feasible solution", or, since $m^*(x, B)$ is undefined and $\max\{\varnothing\} = -\infty$, the relation $R(x, A(x, B)) \leq \alpha_1$ becomes trivial and the algorithm can output *any* almost feasible solution. (We refer to [KM+98a] for a suggestion on a post-processing step which produces even in this case a final solution with a "good" objective value m_1 with respect to a further relaxed instance.)

On the other hand, even if the algorithm outputs a solution, one can not draw a conclusion to the fact whether $FS \circ \Gamma(x, B) \neq \varnothing$. Therefore the existence of multi-criteria approximation algorithms does not imply $P = NP$, even if the decision problem on the set of feasible solutions is NP-complete.

Notational Conventions

For convenience, we use a simpler notation for naming the optimization problems throughout this thesis. The set I of instances is usually the set of graphs with additional node and edge attributes depending on the particular problem. It is specified explicitly in the definition of the problem but withdrawn from the problem name. The set Γ of feasible instances consists in most cases of a class of subgraphs. This is also specified in the definition. In the problem name, the class appears in the designation of the objective functions. Moreover, from the context it is usually evident which of the feasibility objectives have maximization or minimization goal.

Thus we use the following canonical notation for k-criteria optimization problems: each problem is denoted by a k-tuple. The first entry consists of the optimization objective and its goal. The remaining entries describe the bounds on the remaining objectives. For instance, the tuple

(MAX: FLOW, COST IMPROVEMENT, TOTAL FLOW COST)

(taken from Definition 6.6 on page 150) denotes the problem of maximizing the value of a flow in a network (which itself is a directed graph furnished with edge capacities, flow costs and some more parameters), subject to a restriction on the budget available for the edge cost improvement and a restriction on the flow cost of the resulting flow. It will later be clear from the context which improvement model to apply and that both restrictions are upper bounds. Within the running text, we also use a further abbreviation MAXFL-COI-FC derived in an obvious acronym-like fashion from the full tuple.

Related Multi-criteria Optimization Problems

From the definition of a k-criteria optimization problem with objective functions $(m_1, \ldots, m_k)$ is is evident that objective m_1 plays a special role while objectives $m_2, \ldots, m_k$ are treated equally. This means, objectives $m_2, \ldots, m_k$ (together with the corresponding bounds) can be interchanged with each other without affecting the spirit of the problem.

By interchanging the first objective m_1 with one of the remaining objectives m_i, $i > 1$, one constructs a *family of related problems*. In the case of bicriteria problems, the result of this construction is also called *pair of dual problems*.

There is a strong relationship between the approximability within a family of related problems. The following result is an easy generalization of a theorem formulated for a pair of dual problems in [KM+98a].

Theorem 1.3 (Approximability of Related Problems)
Let $\pi = (m_1, \ldots, m_k)$ be a k-criteria optimization problem. Let π' be a related problem constructed from π by interchanging objectives m_1 and m_i. Without loss of generality $i = 2$, i.e., $\pi' = (m_2, m_1, m_3, \ldots, m_k)$. If π is approximable with performance $(\alpha_1, \ldots, \alpha_k)$, then for any $\varepsilon > 0$ problem π' is approximable with performance $((1+\varepsilon)\alpha_2, \alpha_1, \alpha_3, \ldots, \alpha_k)$. Moreover, if objective m_2 is integer valued, then problem π' can be approximated with performance $(\alpha_2, \alpha_1, \alpha_3, \ldots, \alpha_k)$.

Proof. We assume that objectives m_1 and m_2 have both minimization goals. The other cases are analogous. Let A be the approximation algorithm for π. The proof uses A as a subroutine within a binary search to construct an approximation for π' (see [KM+98a]).

Let B_1 be the bound on m_1 in problem π', let $B_3, \ldots, B_k$ be the remaining bounds. Let OPT be the optimal m_2-value for this problem. Since the size of numbers is polynomially bounded (confer page 7), one can guess in polynomial time a number $\Omega \in \mathbb{N}$ with $0 \leq \mathrm{OPT} \leq \Omega$.

Perform a binary search on the set

$$M = \{ (1+\varepsilon)^i \mid i = 0, 1, \ldots, \lceil \log_{1+\varepsilon} \Omega \rceil \}$$

to find the smallest number $B_2 \in M$ such that algorithm A with bound B_2 on objective m_2 and bounds $B_3, \ldots, B_k$ on the remaining objectives outputs a solution y with $m_1(y) \leq \alpha_1 B_1$. Then, $B_2 \leq (1+\varepsilon)\mathrm{OPT}$. Consequently, $m_2(y) \leq \alpha_2 B_2 \leq (1+\varepsilon)\alpha_2 \mathrm{OPT}$. Clearly, $m_j(y) \leq \alpha_j B_j$ for all $j = 3, \ldots, k$. Thus, solution y has performance $((1+\varepsilon)\alpha_2, \alpha_1, \alpha_3, \ldots, \alpha_k)$ for problem π'.

If objective m_2 is integer valued, the binary search can be performed on the interval $[1, \Omega]$ of integers, and the minimal successful number B_2 satisfies $B_2 = \mathrm{OPT}$.

We briefly estimate the running time in the case of non-integer valued objective functions. The binary search on set M needs $\log_2 \log_{1+\varepsilon} \Omega$ calls to algorithm A. Since $\Omega \leq 2^{p(|x|)}$ for some polynomial p on input x, the number of iteration equals

$$\log_2 \log_{1+\varepsilon} \Omega \leq \log_2 \log_{1+\varepsilon} 2^{p(|x|)} = \log_2 \left(p(|x|) \frac{\ln 2}{\ln(1+\varepsilon)} \right) .$$

Since $1/\ln(1+\varepsilon) \in o(1/\varepsilon^2)$ for $\varepsilon \to 0$, the number of iterations (and, finally, the overall running time) is polynomial in the input size and in $1/\varepsilon$. □

Similarly to the definition of FPAS for the uni-criterion case, we now define the class mFPAS containing all problems which admit a *multi-criteria fully polynomial approximation scheme*: A multi-criteria optimization problem belongs to mFPAS, if there is an approximation algorithm which computes for each $\varepsilon_1 > 0$ and $\varepsilon_2 \geq 0, \ldots, \varepsilon_k \geq 0$ a solution with performance $(1+\varepsilon_1, \ldots, 1+\varepsilon_k)$, and whose running time is a polynomial in the input size and in $\prod_{\varepsilon_i > 0} 1/\varepsilon_i$.

We obtain the following result:

Theorem 1.4
Let π and π' be two members of a family of related problems. If $\pi \in$ mFPAS, then also $\pi' \in$ mFPAS.

Proof. Let $(1+\varepsilon_2, 1+\varepsilon_1, 1+\varepsilon_3, \ldots, 1+\varepsilon_k)$ be the desired performance guarantee for π'. Choose $\varepsilon > 0$ such that $(1+\varepsilon)^2 \leq 1+\varepsilon_2$. Apply Theorem 1.3 with performance vector $(1+\varepsilon_1, 1+\varepsilon, 1+\varepsilon_3, \ldots, 1+\varepsilon_k)$ for π to obtain an approximation with performance $((1+\varepsilon)^2, 1+\varepsilon_1, \ldots, 1+\varepsilon_k)$ for π'. Since $(1+\varepsilon)^2 \leq 1+\varepsilon_2$, the solution obeys the initial performance guarantee. From the observations at the end of the proof of Theorem 1.3 it follows that the running time of the overall algorithm is also polynomial in $\prod_{\varepsilon_i > 0} 1/\varepsilon_i$. □

1.3.3 Reductions and Hardness of Approximation

In order to derive an approximation algorithm for a new problem, it is useful to use a reduction to a problem where the approximability is already known. The guaranteed performance carries over to the new problem if this reduction complies with some special properties. In particular, if there is a linear relation between the optimal measures and a linear relation between the absolute errors, the resulting reduction is called *L-reduction*.

Definition 1.5 (L-reduction [PY91])
*Let π_1 and π_2 be two NPO problems. Problem π_1 is **L-reducible to** π_2, in symbols $\pi_1 \leq_L \pi_2$, if polynomial time computable functions f and g and two constants $\beta > 0, \gamma > 0$ exist, such that*

1. *for each $x \in I_{\pi_1}$, we have $f(x) \in I_{\pi_2}$ with*

$$\begin{aligned} m^*_{\pi_2}(f(x)) &\leq \beta \cdot m^*_{\pi_1}(x) && \text{(for minimization problems)} \\ m^*_{\pi_2}(f(x)) &\geq 1/\beta \cdot m^*_{\pi_1}(x) && \text{(for maximization problems)} \end{aligned}$$

2. *for each $x \in I_{\pi_1}$ and $y \in \mathrm{SOL}_{\pi_2}(f(x))$, we have $g(x,y) \in \mathrm{SOL}_{\pi_1}(x)$ with*

$$\left| m^*_{\pi_1}(x) - m_{\pi_1}(x, g(x,y)) \right| \leq \gamma \cdot \left| m^*_{\pi_2}(f(x)) - m_{\pi_2}(f(x), y) \right| .$$

*The tuple (f, g, β, γ) is called **L-reduction** from π_1 to π_2.*

An L-reduction is *approximation preserving* (confer [PY91]): Let A_2 be an approximation algorithm for π_2 with performance α. If $\pi_1 \leq_L \pi_2$, then one can derive an approximation algorithm A_1 for π_1 as follows. Let (f, g, β, γ) be the L-reduction from π_1 to π_2. Define

$$A_1(x) := g(x, A_2(f(x))) .$$

Clearly, A_1 is a polynomial time algorithm. For simplicity, we assume that both problems are minimization problems (the other case is similar). The performance of algorithm A_1 is given by

$$\begin{aligned} R(x, g(x, A_2(f(x)))) &= \frac{\left| m^*_{\pi_1}(x) - m_{\pi_1}(x, g(x, A_2(f(x)))) \right|}{m^*_{\pi_1}(x)} \\ &\leq \gamma \cdot \frac{\left| m^*_{\pi_2}(f(x)) - m_{\pi_2}(f(x), A_2(f(x))) \right|}{m^*_{\pi_1}(x)} \\ &\leq \gamma\beta \cdot \frac{\left| m^*_{\pi_2}(f(x)) - m_{\pi_2}(f(x), A_2(f(x))) \right|}{m^*_{\pi_2}(f(x))} \\ &\leq \gamma\beta \cdot \alpha . \end{aligned}$$

Hence, the α-approximation algorithm for π_2 implies an $\alpha\beta\gamma$-approximation for problem π_1. We remark that in many of the proofs the reduction satisfies $\beta = \gamma = 1$.

The existence of an L-reduction can also be exploited in the opposite direction: Assume there is a non-approximability result for a problem π_1. With providing a proof for $\pi_1 \leq_L \pi_2$ one can derive a non-approximability result for problem π_2.

The L-reduction is a sufficiently powerful tool for the aims of this work. For a further discussion of other approximation preserving reductions, in particular the stronger *AP-reduction* and its applications, we refer e. g. to [AC+99].

For deriving the hardness of approximating problems investigated in this thesis, we make particularly use of two strong non-approximability results from [LY93, Fei96] (stated in the following two theorems) where logarithmic lower bounds on the approximability of MINIMUM SET COVER and MINIMUM DOMINATING SET have been provided. Further non-approximability results can be found e. g. in [AL97].

An instance of MINIMUM SET COVER [GJ79, Problem SP5] is given by a finite set $Q = \{q_1, \ldots, q_s\}$ of ground elements and a family $F = \{Q_1, \ldots, Q_l\}$ of subsets of Q. The problem consists of finding a covering $C \subseteq F$ of Q with minimum number $|C|$ of sets.

Theorem 1.6 (Non-approximability of MINIMUM SET COVER)
Unless NP $\subseteq$ *DTIME*$(N^{O(\log\log N)})$, *for any* $\varepsilon > 0$, *there is no approximation algorithm for* MINIMUM SET COVER *with performance* $(1-\varepsilon)\ln|Q|$. □

In [ESW98] there has been shown that the instances constructed in the reduction proof of the preceding theorem have a special property: The number of subsets is polynomially bounded by the number of ground elements. This fact will later be exploited in reductions from MINIMUM SET COVER.

Property 1.7
Let (Q, F) *be any instance produced by the reduction in [Fei96] used for the proof of Theorem 1.6. Then,* $|F| \leq |Q|^5$. □

An instance of MINIMUM DOMINATING SET [GJ79, Problem GT2] is given by a graph $G = (V, E)$. A node v *dominates* a node $w \in V$ if either $v = w$ or $(v, w) \in E$. A subset $D \subseteq V$ is called *dominating set*, if each node in V is dominated by at least one node from D. The problem consists of finding a dominating set of minimal cardinality.

Theorem 1.8 (Non-approximability of MINIMUM DOMINATING SET)
Unless NP $\subseteq$ *DTIME*$(N^{O(\log\log N)})$, *for any* $\varepsilon > 0$, *there is no approximation algorithm for* MINIMUM DOMINATING SET *with performance* $(1-\varepsilon)\ln|V|$. □

1.4 Network Design and Network Upgrade

Graphs are a useful tool to model network planning problems. There are many practical applications of network planning, in particular in the area of communication networks and logistics. Edge and node attributes are used to model the parameters (including length of links, cost of transportation, delay of routing, capacity of transmission) extracted from the realistic application. A challenging goal is to develop models which are both close enough to reality to draw reasonable conclusions from the theoretical results towards an implementation and simple enough to be handled within the theoretical framework.

A main area of network planning is the field of *network design problems*. Problems of this type are given by a set of possible solutions which is usually modeled by a single graph. Problem specific requirements and constraints define feasible solutions. In many applications those constraints can be reflected by specifying a particular graph class. One or more appraisal functions define a relation on the set of feasible solutions and a notion of optimality. This naturally yields multi-criteria optimization problems on weighted graphs where the goal is to *find a substructure of the input instance* which best meets all demands of the individual problem situation.

One of the first network design problems to study is the *minimum spanning tree* problem [Kru56]. Further problems include the search for a k-*minimum spanning tree* [AA+95, BCV95, BRV96, BRV99], for a *Steiner tree* [KZ97], for a *shortest path tree* [Dij59, MM93, Tho97], for a *minimal total path length tree* [WL+98, WCT00], and for a *minimum diameter spanning tree* [Ple81, HT95]. Other types of edge weight functions yield the *most uniform spanning tree* problem [CM+86, GS88]. This area of research also covers the *minimum label spanning tree* problem [CL97, KW98] and the *minimum reload cost* problems with respect to the *length of a path* and to the *diameter* and *radius* of a spanning tree [WS99, WS01] studied in this thesis.

Natural network design problems arise also in the area of location planning. We briefly recall some important problems including the p-*center* [KH79a, DF85, MF90, HS85, Vis97, Arc00, Kru95, CGR98, KPS00] and p-*median* problem [KH79b, LV92, ARR98, CG+99b, CG99a, JV99] and the field of *facility location* problems [GK99a, CG99a, JV99, STA97, JV00, Shm00, KPR00].

Another important class of network design problems are tour planning problems. Probably one of the most famous problems is the *traveling salesperson* problem [Kru56, Chr76, AF+95], which consists in finding a shortest round tour visiting all nodes. A tour visiting all edges of a graph yields the *chinese postman* problem [EJ73, Fre79, FHK78, FG93]. Many authors exploit the geometric context behind tour planning problems [AA+95, Mit96, Mit98,

AMN98, NMK99, Mit00]. A shortest tour meeting a set of arcs representing transportation requests yields the *dial a ride* problem (also known as *stacker crane*) [FHK78, CR98, AKR00, HK+01, FS01] which is also contributed to in this thesis.

Opposed to the network planning problems described above, *network upgrade problems* use a different approach which is motivated from the observation that in realistic applications one seldomly can spend the effort for implementing a solution from scratch after theoretically computing an optimal network. In most cases the situation is more likely as follows: one is given a current network, and the goal is to *change the existing network in order to improve its efficiency*. The modification of the graph is caused by an investment on several parts of the network which is limited by an overall bound on the available budget.

Problems where the structure of the graph is subject to change (by inserting new edges) are known as *augmentation problems* in literature starting with [ET76]. Other upgrade models retain the graph structure but modify the weights of edges and nodes [KM+98b].

A *node upgrade model* can be found in [PS95], where upgrading a node effects a decrease in edge length of all incident edges by a factor $\delta < 1$. We use a more general node upgrade model [KM+99b, KM+99a, KM+01] in this thesis where the decrease can be adjusted separately for each incident edge. A node upgrade model is motivated from the fact that in particular in high speed optical networks switches are the limiting factor with respect to the capacity of the network, hence an upgrade of a node improves the throughput in all incident links.

In *edge upgrade models* there is usually one linear or, more general, monotone function per edge which describes the dependency between the change of the edge weight and the amount of budget needed for the improvement. The problems considered in this area of research include the search for *minimizing a minimal spanning tree* [KN+98], *minimizing a shortest path tree* [Ber92], *maximizing a minimal spanning tree* [FSO96], *minimizing longest paths* [HT97], or *minimizing the capacity of a minimum cut* [Phi93]. Flow problems with capacities to upgrade have been considered in [CGS98, SC+98, KN+99, DNW00] and in this thesis.

1.5 Literature

There are numerous monographs and standard textbooks on elementary graph theory. We refer the reader for instance to [Har72, Nol76, Die96]. Flow algorithms have been surveyed in [AMO93]. Graph problems and applications with special focus on directed graphs can be found in [BJG00]. For implementations

of efficient graph algorithms see the LEDA library [MN99].

For further books on computational complexity we refer the reader for instance to [GJ79, WW85, Pap94, AC+99]. Efficient algorithms and data structures can be found in [CLR90, MN99].

For books on combinatorial optimization we refer the reader e. g. to [Law76, PS82, GLS88]. Approximation issues with special focus on graph problems can be found in [Hoc97, AC+99]. We also refer to [CK] which is the regularly updated online database version of the list of results for NPO problems from [AC+99]. A survey on multi-criteria optimization problems with focus on pareto optimality can be found in [Ehr00, EG00].

Part I

Network Design Problems

Chapter 2

Edge Labeled Graphs

The problems considered in the current chapter are characterized as network design problems on edge labeled graphs. In all cases the goal is to find an optimal subgraph of an edge labeled graph. Optimality of a solution is measured in the number of labels used by the subgraph. Feasible subgraphs are specified by a graph class. Typical graph classes are the class of all spanning trees or the class of all paths connecting two distinguished vertices.

As pointed out in the introduction, graph attributes such as edge functions are a general model to specify measurements including the length of a link, the duration or delay of transmission, the effort of construction, or the cost of transportation. Edge functions of this type are summarized as *numerical edge attributes*. We will speak of *edge weights*, *edge lengths*, and *edge costs*, respectively, depending on the context.

A common property of numerical edge attributes is that they represent (integer, rational, or real) numbers. This allows to do *calculations* with those attributes and enables the definition of complex objective functions for graphs which measure e. g. the length of paths, the total weight of trees or the transportation costs of flows. This is the basis for the formulation and definition of many optimization problems.

There are other types of edge functions, called *edge labels*, which we try to carefully distinguish from numerical edge attributes in this thesis. For motivation of edge labels, consider an electronic communication network which is subdivided into parts of different transmission technologies. Each subnetwork uses its own specifications for transmitting data. These specifications may include transmission speed, packet size, and special protocols for error recovery. The subnetworks are connected via gateways which have the task to adapt the protocols. If information is routed through this network, data has to be converted

at gateways. In order to keep the effort for installing and running data conversion gateways as small as possible, a reasonable goal is to minimize the number of subnetworks of different technologies involved in a routing network.

A practical way to model this scenario is to choose a discrete set of labels or colors and assign a unique label to each subnetwork. The overall instance is then modeled by an edge colored graph where each graph induced by the set of edges of the same color represents one of the above mentioned subnetworks.

By now, there is no other operation defined on the set of labels than counting them. We remark that in real implementations labels are usually stored as integer numbers. Nevertheless, when speaking of labels instead of numerical attributes, we do not make use of the usual properties of integer numbers which are, in particular, arithmetic calculations and comparability.

2.1 Preliminaries and Problem Formulation

We continue with stating some basic definitions of edge labeled graphs and subgraphs.

Definition 2.1 (Edge labeled graph)
*An **edge labeled graph** $G = (V, E, c)$ is a graph (V, E) together with an edge label function $c\colon E \to \mathbb{N}$.*

Notice that an edge labeled graph may contain parallel edges of different colors, i. e., we do not require the graph to be simple throughout this chapter.

Definition 2.2 (k-labeled subgraph)
*Let $G = (V, E, c)$ be an edge labeled graph. A subgraph $H \sqsubset G$ is called **k-labeled** if*

$$\left|\{\, c(e) \mid e \in E(H) \,\}\right| = k,$$

i. e., the subgraph uses k different labels.

We assume without loss of generality that the graph does not contain parallel edges of the same color throughout the chapter.

In the sequel we investigate *minimum label subgraph* problems. Typical classes of subgraphs are the class of paths or the class of spanning trees. Given an edge labeled graph, the goal of a minimum label subgraph problem is to find a k-labeled subgraph of the specified graph class where k is as small as possible.

Definition 2.3 (MINIMUM LABEL SPANNING TREE Problem)
Let $G = (V, E)$ be a connected undirected graph and $c\colon E \to \mathbb{N}$ be an edge labeling function. The goal is to find a k-labeled spanning tree of G with minimum k.

The problem is defined on connected graphs only. The analogous problem on general (i. e., unconnected) graphs is called MINIMUM LABEL SPANNING FOREST problem. A *spanning forest* is defined to be a family of trees each spanning one connected component of the graph. The MINIMUM LABEL SPANNING FOREST problem can easily be reduced to MINIMUM LABEL SPANNING TREE: Solve the tree problem on each of the connected components, and collect the resulting trees. This obvious observation suggests to restrict the view to connected graphs only.

We will provide an approximation algorithm with logarithmic performance guarantee and prove a logarithmic lower bound for the approximability.

The second problem considered in this chapter is the MINIMUM LABEL PATH problem:

Definition 2.4 (MINIMUM LABEL PATH Problem)
Let $G = (V, E, c)$ *be an edge labeled graph,* $v, w \in V$ *two distinguished nodes. The goal is to find a* k*-labeled path in* G *from* v *to* w *where* k *is minimized.*

It will turn out that MINIMUM LABEL PATH is NP-hard and very hard to approximate. This is proved by a reduction from RED-BLUE SET COVER which is introduced in Section 2.5.

2.2 Related Work

The MINIMUM LABEL SPANNING TREE problem was introduced by Chang and Leu in [CL97]. By a reduction from MINIMUM SET COVER the authors show that the corresponding decision problem of MINIMUM LABEL SPANNING TREE is NP-complete. This result holds even if the underlying graph is complete. Moreover, some heuristics for solving MINIMUM LABEL SPANNING TREE were given.

From the viewpoint of the definition, a closely related problem is the MOST UNIFORM SPANNING TREE problem. An instance is given by an undirected graph with edge weights. The uniformity of a tree is measured by the difference between the most and least weighted edge in the tree, i. e., by the size of the interval spanned by the used edges' weights. The goal is to find a spanning tree which minimizes the size of that interval. The MOST UNIFORM SPANNING TREE problem can be solved optimally in time $O(|E| \log |V|)$ [CM+86, GS88].

Minimum *label* subgraph problems are also related to minimum *weight* subgraph problems. The problem of finding a shortest path in an edge weighted

graph is one of the first graph algorithmic problems to investigate in literature. Starting with the algorithm of Dijkstra [Dij59] with running time in $O(|E| + |V| \log |V|)$, there have been many contributions on SHORTEST PATH culminating in Thorups $O(|V| + |E|)$-algorithm [Tho97]. For the case of trees, the problem MINIMUM SPANNING TREE is not less important. There are numerous algorithms including the algorithm of Kruskal [Kru56] with running time $O(|E| \log |V|)$ up to an algorithm with running time $O(|E| \cdot \alpha(|E|, |V|))$ suggested by Chazelle [Cha00].

One may observe that for edge weights, the path problem appears to be less complex than the tree problem. On the other hand, from the results of this chapter one can draw the conclusion that the situation for edge labels is opposed: here it turns out that the path problem, MINIMUM LABEL PATH, is harder to approximate than the tree problem.

2.3 Approximating Minimum Label Spanning Tree

We first note a trivial upper bound on the performance of an approximation algorithm for MINIMUM LABEL SPANNING TREE: Any spanning tree with node set V consists of $|V| - 1$ edges and hence of at most $|V| - 1$ different labels. Therefore, performance $|V| - 1$ is achieved by any spanning tree.

Chang and Leu [CL97] suggest two heuristical polynomial time algorithms and evaluate the quality of the solutions experimentally. In this section we describe the first heuristic algorithm. Confer Algorithm 2.1 on the next page for a detailed description of the algorithm.

Starting with an arbitrary spanning tree, the algorithm considers each non-tree edge once as the *current* edge. The algorithm identifies an edge on the cycle induced by the current edge whose label has the least number of appearances in the cycle; then this edge is replaced by the current edge. Obviously the current subgraph is a spanning tree at each time.

The main idea behind this approach is the following observation: If one removes an edge where the set of edges of the same color in the tree is rather sparse, it seems to be likely that by this procedure the color might disappear from the tree at all. In this case the number of used labels would be reduced. Unfortunately there are examples where this heuristic can be misled as much as possible.

We give a worst case example which shows that this heuristic might produce a solution with performance guarantee $|V|-1$ matching the trivial upper bound. See Figure 2.2 on page 28 for an illustration. The graph consists of a wheel of $n - 1$ nodes plus one center node. One rung of the wheel together with

Input: A graph $G = (V, E)$, and edge labels $c\colon E \to \{1, \ldots, l\}$

```
1  Find an arbitrary spanning tree of G
2  for each non-tree edge i with label c(i) do
3    Let current ← c(i)
4    if label current appears in the spanning tree then
5      Let C be the cycle induced by edge i in the tree
6      Let least be a label with the least number of appearances in C
7      if label current appears more often than label least in cycle C then
8        Replace an edge of label least in C by edge i
9      end if
10   end if
11 end for
```

Output: Current spanning tree

ALGORITHM 2.1: First heuristic of [CL97].

$n - 2$ edges on the periphery of the wheel are assigned the same color 0. The remaining $n - 1$ edges are colored by $n - 1$ more (pairwise different) colors.

The optimal solution uses the edges of the single color 0 which form a spanning tree of the graph.

Assume that Algorithm 2.1 starts with the spanning tree which contains no edge of color 0 (see Figure 2.2 (right)). Then in each iteration, the current edge of color 0 induces a cycle of three or four edges in which no two edges have the same color. Therefore no edge swaps will be performed by the algorithm. Consequently, the heuristic ends up with the initial spanning tree. This tree uses $n - 1 = |V| - 1$ colors.

It can be concluded that $|V| - 1$ is a lower bound on the performance of this heuristic. With the observations above it follows that the heuristic might behave as badly as possible.

In the sequel we will provide an algorithm based on the second heuristic suggested by Chang and Leu [CL97]. While they observe experimentally that this approach behaves well in practice, we prove a logarithmic performance guarantee of the algorithm. This result is completed by a matching lower bound on the approximability of MINIMUM LABEL SPANNING TREE in Section 2.4.

2.3.1 The Algorithm

The second algorithm is formally described in Algorithm 2.3 on page 29. It starts with the graph where all edges are removed. This graph consists of n connected

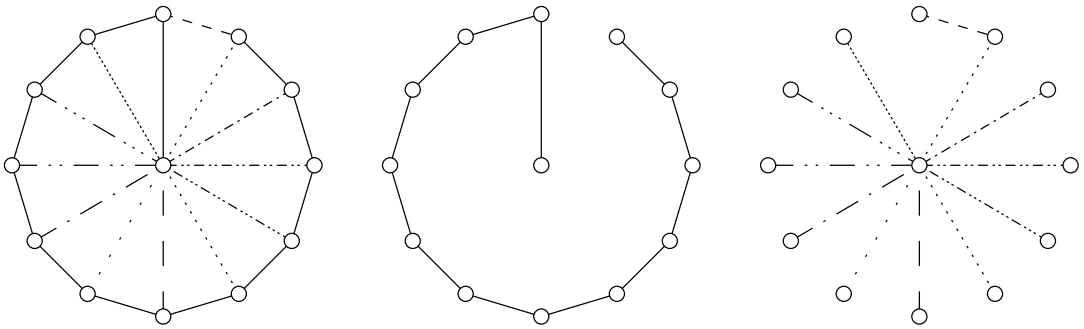

FIGURE 2.2: Worst case example for the first heuristic (left). Optimal solution (center) and heuristic solution (right).

components. Iteratively the algorithm selects a color and throws in all edges of that color. By this procedure the number of connected components is reduced. The iteration is repeated until the graph is connected.

At the end the number of iterations is an upper bound on the number of colors needed for a spanning tree of the graph. Therefore the algorithm tries to keep the number of iterations small. To achieve this goal, in each iteration it decides for that color which reduces the number of connected components by the largest amount.

2.3.2 Performance Guarantee

We now prove that Algorithm 2.3 approximates the MINIMUM LABEL SPANNING TREE problem with a logarithmic performance guarantee. In this section we denote by OPT the value of an optimal solution, i. e., we assume that the graph admits a spanning tree with OPT different labels. If $\mathrm{OPT} = 1$, the algorithm chooses an optimal color in the first iteration and stops immediately. Hence we assume $\mathrm{OPT} \geq 2$ in the sequel.

At first we give the main idea of the proof. The algorithm adds one color per iteration and stops when the current graph is connected. Therefore the number of iterations performed by the algorithm is an upper bound on the number of labels used in the final solution.

The analysis divides the iterations into two phases. The first phase ends after k_1 iterations, where k_1 is given by

$$k_1 := \left\lceil \mathrm{OPT} \cdot \ln \frac{n}{\mathrm{OPT}} \right\rceil . \tag{2.1}$$

Input: A graph $G = (V, E)$, and edge labels $c\colon E \to \{1, \ldots, l\}$

```
1  Let C ← ∅ the set of used colors
2  repeat
3     Let H be the subgraph of G restricted to edges with colors from C
4     Contract each connected component in H to a supernode
5     for all i ∈ {1, ..., l} \ C do
6        Let E_i be the set of edges of color i
7        Determine the number of connected components
         of graph H augmented by edge set E_i
8     end for
9     Choose color i with smallest resulting number of components
10    Let C ← C ∪ {i}
11 until graph H is connected
12 Compute a spanning tree T of graph H
```

Output: Spanning tree T

ALGORITHM 2.3: Approximation Algorithm for MINIMUM LABEL SPANNING TREE.

We will show that during the first phase the number of connected components reduces essentially by a constant factor in each iteration. This can be exploited to bound the number k_2 of connected components which remain at the end of the first phase. During the second phase we use the basic observation that the number of connected components reduces by at least one per iteration. Therefore, $k_2 - 1$ is an upper bound on the number of iterations of the second phase. Consequently,

$$k_1 + k_2 - 1$$

is an upper bound on the total number of iterations and even on the total number of colors in the final solution. Observe that the breakpoint between the two phases is only virtually introduced for the sake of the analysis: since the algorithm does not have any knowledge about OPT in advance, it cannot determine this point.

We will now continue with proving the ingredients needed to estimate the number of connected components remaining at the end of the first phase.

Lemma 2.5
Let H be a connected graph with r nodes which has a p-colored spanning tree. Then there is a color i such that H, restricted to edges of color i, has no more

than

$$r(1-1/p)+1/p$$

connected components.

Proof. Let T be the p-colored spanning tree. Then T contains $r-1$ edges. By an averaging argument, there must be a color i such that the number of i-colored edges in T is at least $(r-1)/p$. Moreover, since the number must be integral, we can conclude that this number is in fact at least $\lceil (r-1)/p \rceil$.

Remove all edges from H and then reinsert all tree edges of color i one by one. Each time inserting one edge, the number of connected components connected component decreases by one, since otherwise the edges would form a cycle in the tree T. Therefore, the resulting graph has at most

$$r - \left\lceil \frac{r-1}{p} \right\rceil = \left\lfloor r - \frac{r-1}{p} \right\rfloor = \left\lfloor r\left(1-\frac{1}{p}\right)+\frac{1}{p} \right\rfloor$$

connected components. This shows the claim. □

Let OPT be the number of colors in the optimal spanning tree. Define

$$f(n) := n(1-1/\mathrm{OPT})+1/\mathrm{OPT}\,.$$

Then, by Lemma 2.5 and the fact that the algorithm chooses the best possible color in each iteration, it follows that after the first iteration no more than $f(n)$ connected components remain. We define the k-fold iteration of f by $f^k(n) := f(f^{k-1}(n))$ and conclude by induction that $f^k(n)$ is an upper bound on the number of connected components after iteration k. The value of $f^k(n)$ can be bounded as follows:

Lemma 2.6
Let $f(n) := n(1-1/p)+1/p$ *for some fixed* $p \geq 1$. *Then,*

$$f^k(n) \leq n\left(1-\frac{1}{p}\right)^k + \frac{k}{p}\,.$$

Proof. We establish the claim by induction on k. The claim is trivial for $k=0$. Assume that it is correct for some $k \geq 0$. Then, using the induction hypothesis,

$$\begin{aligned} f^{k+1}(n) = f(f^k(n)) &= f^k(n)\left(1-\frac{1}{p}\right)+\frac{1}{p} \\ &\leq n\left(1-\frac{1}{p}\right)^k\left(1-\frac{1}{p}\right)+\frac{k}{p}\left(1-\frac{1}{p}\right)+\frac{1}{p} \\ &\leq n\left(1-\frac{1}{p}\right)^{k+1}+\frac{k+1}{p} \end{aligned}$$

as proposed in the lemma. □

At this point we can bound the number of components remaining after k iterations. We now try to estimate how many iterations are necessary until the number of components has fallen below a constant. To this end, we formulate the following lemma which can be shown by elementary analysis:

Lemma 2.7
For $n, p \in \mathbb{N}$, $n \geq p$, we have:

$$n(1-1/p)^k \leq p \qquad \text{if } k \geq p \ln \frac{n}{p}\,.$$

Proof. Since the expression given on the left hand side of the inequality is decreasing as long as k increases, it suffices to show that $n\big(1-1/p\big)^k \leq p$ for $k = p \ln \frac{n}{p}$. Taking the natural logarithm on both sides and plugging in the value of k, we have the following chain of equivalent inequations:

$$\begin{aligned}
& \left(1-\frac{1}{p}\right)^k \leq \frac{p}{n} \\
\iff \quad & e^{k \cdot \ln \frac{p-1}{p}} \leq \frac{p}{n} \\
\iff \quad & p \cdot \ln \frac{n}{p} \cdot \ln \frac{p-1}{p} \leq \ln \frac{p}{n} \\
\iff \quad & p \cdot \ln \frac{p}{p-1} \geq 1\,.
\end{aligned}$$

The function $p \mapsto p \ln \frac{p}{p-1}$ is non-increasing on interval $]1, \infty[$ and has limit of 1 for $p \to \infty$. Thus, $p \ln \frac{p}{p-1} \geq 1$ for all $p > 1$, and the claim follows. □

We now are able to bound the number of iterations the algorithm performs. Recall that $k_1 := \lceil \mathrm{OPT} \cdot \ln(n/\mathrm{OPT}) \rceil$ is the number of iterations of the first phase. Let k_2 be the number of connected components at the end of this phase. By Lemma 2.6 and Lemma 2.7, k_2 is bounded from above by

$$\begin{aligned}
k_2 &\leq f^{k_1}(n) \\
&\leq n\left(1-\frac{1}{\mathrm{OPT}}\right)^{k_1} + \frac{k_1}{\mathrm{OPT}} \\
&\leq \mathrm{OPT} + \frac{k_1}{\mathrm{OPT}}\,.
\end{aligned} \tag{2.2}$$

Now we consider the number of iterations in the second phase. Since the number of connected components decreases by at least one in each iteration, phase 2 consists of no more than $k_2 - 1$ additional iterations.

As observed before, the total number of iterations is not greater than $k_1 + k_2 - 1$. Using the previous estimations, we conclude further

$$\begin{aligned} k_1 + k_2 - 1 &\overset{(2.2)}{\leq} k_1 + \mathrm{OPT} + \frac{k_1}{\mathrm{OPT}} - 1 \\ &\overset{(2.1)}{=} \left\lceil \mathrm{OPT} \cdot \ln \frac{n}{\mathrm{OPT}} \right\rceil + \mathrm{OPT} + \frac{\left\lceil \mathrm{OPT} \cdot \ln \frac{n}{\mathrm{OPT}} \right\rceil}{\mathrm{OPT}} - 1 \\ &\leq \mathrm{OPT} \cdot \ln \frac{n}{\mathrm{OPT}} + 1 + \mathrm{OPT} + \left\lceil \ln \frac{n}{\mathrm{OPT}} \right\rceil - 1 \\ &\leq \mathrm{OPT} \cdot \ln \frac{n}{\mathrm{OPT}} + \mathrm{OPT} + \ln \frac{n}{\mathrm{OPT}} + 1 \\ &\leq \mathrm{OPT} \cdot (2 \ln n + 1) \end{aligned} \tag{2.3}$$

In the last line of (2.3) we have used the observation $\ln(n/p) + 1 \leq p \ln(n/p)$ which is true for $1 < p \leq n$. Hence, $\mathrm{OPT} \cdot (2 \ln n + 1)$ is an upper bound on the number of iterations performed by the algorithm. This is summarized by the following theorem:

Theorem 2.8 (Approximability of MINIMUM LABEL SPANNING TREE)
Algorithm 2.3 is an approximation algorithm for the MINIMUM LABEL SPANNING TREE *problem with performance guarantee of* $2 \ln n + 1$. □

The running time of the algorithm is polynomial and will be precisely estimated in Section 2.3.4.

2.3.3 Asymptotic Performance Guarantee

The approximation result from Theorem 2.8 can further be improved using an asymptotic performance analysis:

Theorem 2.9 (Approximability of MINIMUM LABEL SPANNING TREE)
For any $\varepsilon > 0$*, problem* MINIMUM LABEL SPANNING TREE *can be approximated with performance* $(1 + \varepsilon) \ln n$ *in polynomial time.*

Proof. Let $\varepsilon > 0$ be fixed. We first show that there exist constants $p_0 \in \mathbb{N}$ and $n_0 \in \mathbb{N}$ such that for any $n \geq n_0$ and any $\mathrm{OPT} \geq p_0$, the last estimation in (2.3) can be strengthened yielding a bound of $(1 + \varepsilon) \ln n$.

To this end, choose $\delta := \min\{\varepsilon/3, 1\}$. Further we define

$$p_0 := \left\lceil \frac{1}{\delta} \right\rceil \qquad \text{and} \qquad n_0 := \left\lceil e^{1/\delta} \right\rceil .$$

Then for any $n \geq n_0$, $\mathrm{OPT} \geq p_0$, we have

$$\frac{1}{\mathrm{OPT}} \leq \delta, \qquad \frac{1}{\ln n} \leq \delta, \qquad \frac{1}{\mathrm{OPT} \cdot \ln n} \leq \delta^2 \leq \delta\,.$$

In this case, the right hand side of (2.3) can be estimated as follows:

$$\begin{aligned}
&\mathrm{OPT} \cdot \ln \frac{n}{\mathrm{OPT}} + \mathrm{OPT} + \ln \frac{n}{\mathrm{OPT}} + 1 \\
&\qquad \leq \mathrm{OPT} \cdot \ln n + \mathrm{OPT} + \ln n + 1 \\
&\qquad = \left(1 + \frac{1}{\ln n} + \frac{1}{\mathrm{OPT}} + \frac{1}{\mathrm{OPT} \cdot \ln n}\right) \cdot \mathrm{OPT} \cdot \ln n \qquad (2.4) \\
&\qquad \leq (1 + 3\delta) \cdot \mathrm{OPT} \cdot \ln n \\
&\qquad \leq \mathrm{OPT} \cdot (1 + \varepsilon) \cdot \ln n\,.
\end{aligned}$$

Notice that n_0 and p_0 are constants which do not depend on the input instance.

An approximation for MINIMUM LABEL SPANNING TREE can be computed as follows: If the number n of nodes in the input instance obeys $n < n_0$, then a solution can be output using a table lookup in constant time. For $n \geq n_0$, we have two phases: First, for each $i = 1, \ldots, p_0 - 1$ we check all $\binom{l}{i} \in O(m^i)$ combinations of i colors for a feasible solution. The running time for the first phase is in $O(m^{p_0})$ and in particular polynomial. If no feasible solution has been found, we start the second phase and run Algorithm 2.3. Since in this situation we know that $\mathrm{OPT} \geq p_0$, we can use (2.4) to improve the estimation (2.3). This shows that the performance guarantee of the algorithm is bounded by $(1 + \varepsilon) \ln n$. □

We point out that this is only a theoretical result. Albeit the actual running time of the suggested algorithm is shown to be polynomial, it may be far too large to be of practical use.

2.3.4 Running Time

We now estimate the running time of Algorithm 2.3 on page 29. Let l be the number of colors of the input graph. We show that the algorithm can be implemented to run in time $O(m + \alpha(m, n) \cdot l \cdot \min\{ln, m\})$. Here, α is the inverse of Ackerman's function (see page 8 for a definition).

The implementation uses efficient data types and data structures for disjoint sets. See [CLR90] for a detailed description of the *path compression* and *union-by-rank* heuristics for union-find data structures. Recall that t union-find operations on s elements can be implemented with total running time $O(t \cdot \alpha(t, s))$.

We first do a preprocessing step to reduce the number of edges of each color to at most $n-1$. Let $E_c \subseteq E$ be the set of edges of color c and choose $F_c \subseteq E_c$ such that $G[F_c]$ is a spanning forest of the graph $G[E_c]$. Then $|F_c| \leq n-1$ while both edge sets E_c and F_c still induce the same connected components of graph G. Replace E_c by F_c. Since the main algorithm operates on connected components rather than on edge sets, this preprocessing does not have an impact on the correctness. The computation of the spanning forests for all colors $c = 1, \ldots, l$ can be implemented using the *depth first search* algorithm (see e.g. [CLR90]) in total time $O\big(\sum_{c=1}^{l}(n+|E_c|)\big) \subseteq O(ln+m)$.

To determine the number of connected components when adding edge set F_c to the graph (Step 7 of Algorithm 2.3) it suffices to perform $|F_c|$ union-find operations on the set of nodes of H for a fixed color c. Since there are at most l colors to test in each iteration, the running time per iteration can be bounded by

$$O\Big(\sum_{c=1}^{l}|F_c|\,\alpha(|F_c|,n)\Big) \subseteq O\Big(\alpha(m,n)\cdot\sum_{c=1}^{l}|F_c|\Big) \subseteq O\big(\alpha(m,n)\cdot\min\{ln,m\}\big)\,.$$

The last estimation uses the fact that $|F_c| < n$ for all colors c and on the other hand $\sum_{c=1}^{l}|F_c| \leq |E| = m$. Now observe that the number of iterations is at most l. Respecting the effort used for the preprocessing, this results in

$$O(m+\alpha(m,n)\cdot l\cdot\min\{ln,m\})$$

as an upper bound for the overall running time of Algorithm 2.3.

Theorem 2.10 (Running time)
Algorithm 2.3 has a running time within

$$O(m+\alpha(m,n)\cdot l\cdot\min\{ln,m\})$$

on an l-colored graph. □

2.4 Hardness of Minimum Label Spanning Tree

In this section we provide a logarithmic lower bound on the approximability of MINIMUM LABEL SPANNING TREE.

Theorem 2.11 (Non-Approximability of MINIMUM LABEL SPANNING TREE)
Unless NP $\subseteq$ DTIME($N^{O(\log\log N)}$), for any $\varepsilon > 0$, MINIMUM LABEL SPANNING TREE *cannot be approximated with performance $\frac{6}{31}(1-\varepsilon)\ln|V|$.*

Proof. We give a reduction from MINIMUM SET COVER. An instance of MINIMUM SET COVER is given by a finite set $Q = \{q_1, \ldots, q_s\}$ of ground elements and a family $F = \{Q_1, \ldots, Q_l\}$ of subsets of Q. The problem consists of finding a covering $C \subseteq F$ of Q with minimum number $|C|$ of sets. The problem MINIMUM SET COVER cannot be approximated within a factor $(1 - \varepsilon) \ln |Q|$ for any $\varepsilon > 0$, unless NP $\subseteq$ DTIME($N^{O(\log \log N)}$) (see Theorem 1.6 on page 17).

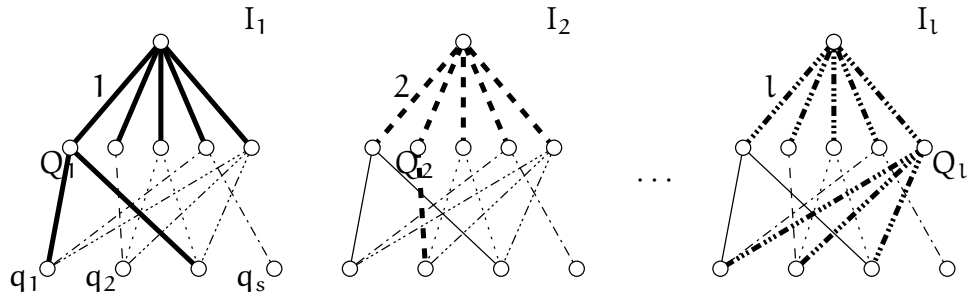

FIGURE 2.4: Reduction from MINIMUM SET COVER to MINIMUM LABEL SPANNING TREE. In instance I_j, color j is shown by thick lines.

Given an instance of MINIMUM SET COVER problem, we construct a collection $\{I_1, I_2, \ldots, I_l\}$ of instances of MINIMUM LABEL SPANNING TREE as follows (confer Figure 2.4 for an illustration of the reduction): The node set of graph I_j consists of Q (the element nodes), F (the set nodes), and an additional node r (the root). The root is connected to each set node via an edge of color j. Further, the graph contains an edge (q_ν, Q_μ) of color μ between an element node q_ν and a set node Q_μ if and only if $q_\nu \in Q_\mu$. Observe that the construction can be performed in polynomial time.

It is easy to see that there is a covering $C \subseteq F$ of Q with no more than k subsets if and only if there is a $j \in \{1, \ldots, l\}$ such that instance I_j contains a spanning tree with edges of no more than k different colors.

A spanning k-labeled tree induces a set cover of size at most k in an obvious way: If color μ appears in the tree, then choose set Q_μ to be in the cover. Since the tree is spanning, the resulting collection of sets covers the whole ground set. (Note that if a color merely appears in edges adjacent to the root, we can remove the corresponding set Q_j from the cover and obtain a valid cover of size $k - 1$.)

Conversely, let there be a covering C of Q with k subsets. Choose a $j \in \{1, \ldots, l\}$ such that $Q_j \in C$ and consider instance I_j. The subgraph consisting of all edges adjacent to the root and all edges adjacent to one of the set nodes of C consists of k colors, it is connected, and it spans all nodes. Therefore this subgraph contains a spanning tree of no more than k colors.

Consider an instance I of MINIMUM SET COVER with ground set $|Q|$ and subsets F as constructed in the proof of [Fei96] (confer Property 1.7 on page 17), let I_j be the corresponding instance of MINIMUM LABEL SPANNING TREE which admits a tree with the same optimal number OPT of colors.

From the construction it is easy to see that the number n of nodes of instance I_j satisfies $n = |Q| + |F| + 1$. By Property 1.7, we can conclude $n \leq |Q|^5 + |Q| + 1 \leq |Q|^{31/6}$ for $|Q| \geq 2$.

Assume there were an approximation algorithm for MINIMUM LABEL SPANNING TREE on an n-node graph with performance $c \cdot (1-\varepsilon) \ln n$ for some constant $c > 0$. This algorithm, when run on instance I_j, outputs a solution with at most $c \cdot (1-\varepsilon) \ln n \cdot \mathrm{OPT} \leq 31/6 \cdot c \cdot (1-\varepsilon) \ln |Q| \cdot \mathrm{OPT}$ colors which can be transformed into a set cover of the same size. If $c < 6/31$, this is a contradiction to the inapproximability of MINIMUM SET COVER (confer Theorem 1.6 on page 17). □

2.5 Hardness of Minimum Label Path

For proving the hardness of MINIMUM LABEL PATH, we use a reduction from RED-BLUE SET COVER. This problem has been introduced in [JK+99, CD+00].

2.5.1 Red-Blue Set Cover

An instance of RED-BLUE SET COVER specifies a set of ground elements, each colored red or blue, together with a family of subsets of the ground set. Each of the subsets may contain both red and blue elements, or elements of one color only. A feasible covering consists of a collection of sets in the family which cover all the blue elements. The goal is to find a feasible covering which minimizes the number of red elements.

Observe that one can assume without loss of generality that the family does not contain subsets with red elements only.

Definition 2.12 (RED-BLUE SET COVER problem)
Let R and B be two finite sets of red and blue elements, respectively, $R \cap B = \varnothing$. Let $S \subseteq 2^{R \cup B}$ be a family of subsets of $R \cup B$. A solution of the RED-BLUE SET COVER *problem is a collection $C \subseteq S$ with the following two properties:*

1. *all blue elements are covered, i. e.,*

$$\bigcup_{s \in C} s \supseteq B\,,$$

2. *the number of red elements covered, i. e.,*

$$\left| R \cap \bigcup_{s \in C} s \right|,$$

is minimized.

RED-BLUE SET COVER is a generalization of MINIMUM SET COVER. In the latter problem, the goal is to find a covering where the number of sets used for the covering is minimized: This can be reduced to RED-BLUE SET COVER if each of the subsets in the family is assigned a unique red element.

In [DS99], RED-BLUE SET COVER is viewed in the hierarchy of MINIMUM MONOTONE SATISFYING ASSIGNMENT problems. (This problem class is also known by the name AND-OR SCHEDULING [GM97].) A formula is called *monotone* if it contains only positive literals. An assignment is called *minimal* if the number of variables which are assigned the true value is minimal. A formula belongs to class Π_k if it has $k-1$ alternations between AND and OR and the first level of operators is AND. Problem class MMSA_k contains all MMSA problems restricted to formulae of Π_k. In [DS99] it is shown that MINIMUM SET COVER is equivalent to MMSA_2 while RED-BLUE SET COVER is equivalent to MMSA_3.

2.5.2 Hardness of Red-Blue Set Cover

We continue with some results on RED-BLUE SET COVER. Since the problem generalizes MINIMUM SET COVER, one can immediately use the hardness result from Theorem 1.6 on page 17 to derive a logarithmic lower bound on the approximability of RED-BLUE SET COVER under the assumption that $\text{NP} \not\subseteq \text{DTIME}(N^{\log\log N})$.

Improved lower bounds on the approximability have been shown recently in [CD+00]:

Theorem 2.13 (Hardness of RED-BLUE SET COVER [CD+00])
Unless $\text{NP} \subseteq \text{DTIME}(N^{polylog(N)})$, *for any* $\varepsilon > 0$, *problem* RED-BLUE SET COVER *can not be approximated to within a factor of* $O(2^{(\ln n)^{1-\varepsilon}})$, *where* $n = |S|^4$, *and* $|S|$ *denotes the cardinality of the family of subsets given by the problem instance.* □

Theorem 2.14 (Hardness of RED-BLUE SET COVER [CD+00])
Theorem 2.13 continues to hold even if the subsets are restricted to contain exactly one blue and two red elements. □

The proof of Theorem 2.13 uses a reduction from SYMMETRIC LABEL COVER for which a similar lower bound on the approximability is known [DK99]. The restriction to sets of one blue and two red elements can be guaranteed by the reduction used in that proof.

Theorem 2.15 (Hardness of RED-BLUE SET COVER [CD+00])
For any $c < 1/2$, *let* $\delta_c := 1/\log\log^c |R|$, *where* $|R|$ *denotes the size of the set of red elements given by the problem instance. Unless* $P = NP$, RED-BLUE SET COVER *cannot be approximated to within a factor of* $O(2^{\log^{1-\delta_c} |R|})$. □

The proof of Theorem 2.15 uses the equivalence to $MMSA_3$ and a non-approximability result from [DS99].

Notice that the first results are stated in terms of the number of sets while the other one is stated in terms of the number of red elements. Therefore the theorems are not comparable to each other with respect to the strength of the result. If one assumes that the number of subsets is polynomially bounded by the number of red elements, Theorem 2.15 provides the stronger result [CD+00].

In [CD+00] it is argued that the lower bounds of Theorem 2.13 and Theorem 2.15 improve the logarithmic lower bounds following from the straightforward reduction from MINIMUM SET COVER. In the remaining part of this section we try to comprehend this proposition.

Problem MINIMUM SET COVER can be approximated within a factor given by a function $f(n) = (\log n)^k$ for some fixed $k > 0$ (confer [CLR90]). The lower bound for RED-BLUE SET COVER noted above is given by a function $g(n) = b^{(\ln n)^{1-\varepsilon}}$ for arbitrary but fixed $0 < \varepsilon < 1$ and $b > 1$. We briefly deduce that

$$f \in o(g)$$

proving the fact that RED-BLUE SET COVER is indeed substantially harder to approximate than MINIMUM SET COVER.

To examine the behavior of f/g for large n, we substitute $N := \ln n$ and get

$$\begin{aligned} \frac{f(n)}{g(n)} &= \frac{(\ln n)^k}{b^{(\ln n)^\varepsilon}} \\ &= \frac{N^k}{b^{N^\varepsilon}} \end{aligned}$$

It is easy to see that this ratio decreases monotonically if $\varepsilon \cdot N^\varepsilon \cdot \ln b > 1$ and approaches zero for $N \to \infty$ and hence also for $n \to \infty$. Consequently, function g growths indeed asymptotically faster than function f as claimed.

2.5.3 Hardness of Minimum Label Path

After stating the hardness results on RED-BLUE SET COVER, we will now carry out the reduction from RED-BLUE SET COVER to MINIMUM LABEL PATH. It will turn out that the reduction is approximation preserving, hence the non-approximability results for RED-BLUE SET COVER also hold for MINIMUM LABEL PATH.

Theorem 2.16 (Hardness of MINIMUM LABEL PATH)
Unless NP $\subseteq$ *DTIME*$(N^{polylog(N)})$, *for any* $\varepsilon > 0$, *problem* MINIMUM LABEL PATH *can not be approximated to within a factor of* $O(2^{(\ln k)^{1-\varepsilon}})$, *where* $k \in \Omega(n^{1/4})$, *and* n *denotes the number of nodes of the input graph.*

Proof. The proof uses a reduction from problem RED-BLUE SET COVER. Let $I' = (R, B, S)$ be an instance of RED-BLUE SET COVER, where R denotes the set of red elements, B the set of blue elements, and $S \in 2^{R \cup B}$ the family of subsets.

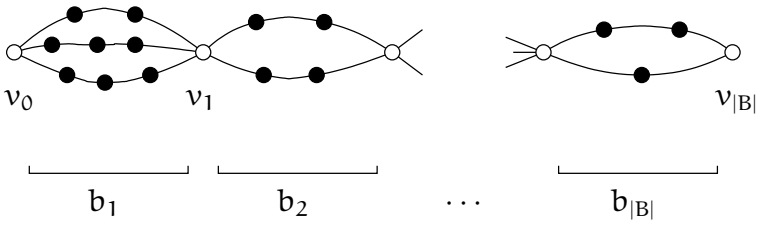

FIGURE 2.5: Reduction from RED-BLUE SET COVER to MINIMUM LABEL PATH.

We setup an instance I for MINIMUM LABEL PATH. The construction is illustrated in Figure 2.5. For each blue element b_i, create a node v_i. Add another node v_0. For each blue element b_i and each set $s \in S$ where $b_i \in s$, create a new path, called (b_i, s)-*representing*, between v_{i-1} and v_i. This path consists of $|s \cap R|$ edges and $|s \cap R| - 1$ new nodes.

Finally we have to color the edges of instance I. To this end, choose a unique color for each red element. Then, for each (b_i, s)-representing path, find an arbitrary bijection between its edges and the red elements of set s, and color the edges according to that bijection.

Denote by n the number of nodes of the constructed instance I. Following the construction as described above, one observes that

$$n = \sum_{i=1}^{|B|} \left(1 + \sum_{\substack{s \in S \\ s \ni b_i}} (|s \cap R| - 1) \right) + 1 \,. \tag{2.5}$$

If we presume the restrictions of Theorem 2.14, we have

$$|S \cap R| \leq 2 \qquad \text{and} \qquad |s \cap B| = 1 \tag{2.6}$$

for all sets $s \in S$. Consequently,

$$|S| \geq |B| \,. \tag{2.7}$$

Plugging (2.6) and (2.7) into (2.5), we can conclude

$$\begin{aligned} n &= \sum_{i=1}^{|B|} \left(1 + \sum_{\substack{s \in S \\ s \ni b_i}} (|s \cap R| - 1) \right) + 1 \\ &= \left(\sum_{i=1}^{|B|} 1 \right) + \left(\sum_{i=1}^{|B|} \sum_{\substack{s \in S \\ s \ni b_i}} |s \cap R| \right) - \left(\sum_{i=1}^{|B|} \sum_{\substack{s \in S \\ s \ni b_i}} 1 \right) + 1 \\ &= \left(\sum_{i=1}^{|B|} 1 \right) + \left(\sum_{s \in S} |s \cap B| \cdot |s \cap R| \right) - \left(\sum_{s \in S} |s \cap B| \right) + 1 \\ &\leq |B| + 2|S| - |S| + 1 \\ &\leq 3|S| \,. \end{aligned} \tag{2.8}$$

It is easy to see that there is a one-to-one correspondence between a v_0-$v_{|B|}$-path using k colors and a cover of B using sets with a total of k different red elements. Moreover, any approximation of MINIMUM LABEL PATH with performance α immediately yields an approximation of RED-BLUE SET COVER with the same factor. Using the hardness result from Theorem 2.14 and the observation in (2.8) the claim follows. □

2.6 Concluding Remarks

In Table 2.6 we briefly summarize the results on the MINIMUM LABEL SPANNING TREE problem obtained in this chapter. The running time is formulated for graphs with n nodes and m edges of l different colors. The lower bound on the approximability assumes NP $\not\subseteq$ DTIME($N^{O(\log \log N)}$).

We close this chapter with pointing to some problems related to MINIMUM LABEL PATH. The first problem, RESTRICTED LABEL PATH, is basically the decision version of MINIMUM LABEL PATH.

Approach	Performance	Reference
Algorithm 2.3 running time $O(m + \alpha(m,n) \cdot l \cdot \min\{\ln, m\})$	$2 \ln n + 1$	Theorem 2.8
polynomial time algorithm	$(1+\varepsilon) \ln n$ for any $\varepsilon > 0$	Theorem 2.9
lower bound	$6/31 \cdot \ln n$	Theorem 2.11

TABLE 2.6: Summary of results on the MINIMUM LABEL SPANNING TREE problem presented in this chapter.

Definition 2.17 (RESTRICTED LABEL PATH problem)
Let $G = (V, E, c)$ *be an edge labeled graph,* $v, w \in V$ *two distinguished nodes,* $k \in \mathbb{N}$ *an integer. Is there a path in* G *from* v *to* w *which is* k*-labeled?*

This problem is NP-complete, as follows from the NP-hardness results for MINIMUM LABEL PATH (Theorem 2.16 on page 39). An alternative and maybe more straightforward proof for this fact follows from the observation that problem HAMILTONIAN PATH can be reduced to RESTRICTED LABEL PATH: Consider an instance of HAMILTONIAN PATH. Assign each edge a unique color. Now it suffices to check all $O(|V|^2)$ pairs of nodes for a $(|V|-1)$-labeled path in order to solve HAMILTONIAN PATH.

Another related problem is MAXIMUM LABEL PATH. Here, the minimization objective from MINIMUM LABEL PATH is exchanged with a maximization objective: The goal is to find a path between two specified nodes which uses as many different colors as possible. Of course the path should be restricted to be a simple path in that case.

Definition 2.18 (MAXIMUM LABEL PATH problem)
Let $G = (V, E, c)$ *be an edge labeled graph,* $v, w \in V$ *two distinguished nodes. The goal is to find a* k*-labeled simple path in* G *from* v *to* w *where* k *is maximized.*

A very easy reduction shows that this problem is NP-hard. Moreover, one can conclude a lower bound on the approximability similar to that of MINIMUM LABEL PATH.

Corollary 2.19
Unless NP $\subseteq$ *DTIME*$(2^{O(\log^{1/\varepsilon} N)})$*, for any* $\varepsilon > 0$*, problem* MAXIMUM LABEL PATH *can not be approximated to within a factor of* $O(2^{(\ln n)^{1-\varepsilon}})$.

Proof. In [KMR97] the stated lower bound has been shown for the LONGEST PATH problem. An instance of LONGEST PATH is given by an undirected graph, and the goal is to find a simple path with as many edges as possible. The reduction from LONGEST PATH to MAXIMUM LABEL PATH can be carried out similarly to the reduction from HAMILTONIAN PATH mentioned above: Assign each edge a unique color, check $O(|V|^2)$ instances for a maximum label path, and return the best solution. □

Chapter 3

Reload Costs

Minimum label subgraph problems considered in the preceding chapter have reasonable applications in the area of transportation problems. Other applications include the design of communication networks. The minimization of the total number of colors used in a subgraph is a useful tool for network design questions of static nature.

Consider for example a problem of transporting goods through a heterogeneous network which is composed of subnetworks. The reader may imagine several flight, truck, and railway subnetworks connected to the overall transportation network. Each subnetwork is run by an independent carrier. In the graph it is represented by edges of a color which is chosen unique for each subnetwork. If transportation requests are scheduled in such a network, then the organization effort for planning and scheduling the request directly depends on the number of carriers involved at the desired route. This motivates the aim to keep this number small. This kind of optimization is well handled by the search for a minimum label subgraph discussed in the previous chapter.

Other applications put the main focus on the dynamic view of routing problems. Consider the transportation problem mentioned above. If a good is transported along a scheduled route through the network of different carriers, then the good has to be reloaded at each node where the active carrier changes. This reloading procedure may induce storage cost and time needed for repacking. We subsume all those types of cost by the notion *reload cost* further.

Obviously, a minimum label subgraph is not necessarily an optimal solution for a problem where the reload costs are to be minimized: Even if a path uses two different colors only, it may be the case that the two colors are alternating along the path, increasing the total sum of reload cost at each node. This observation motivates the formulation and examination of network design problems under a *reload cost model*.

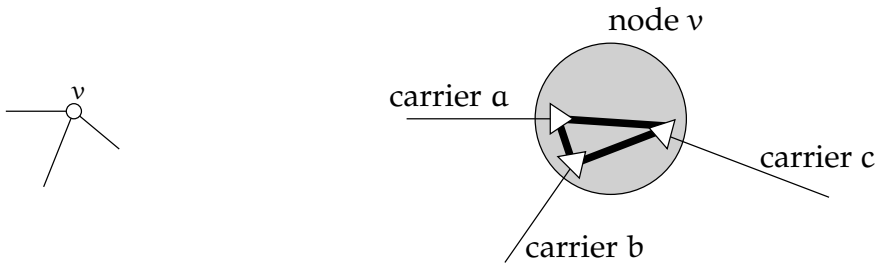

FIGURE 3.1: Node with three incident carrier edges. The reload costs (thick edges) satisfy the triangle inequality.

A pure reload cost model can be applied to problems of transmitting data through incompatible communication networks. In this area, time and cost for routing data within a particular subnetwork are negligible compared to the effort needed for converting data at gateways. Other types of applications request for a mixed model where reload costs at nodes and classical costs at edges are treated at the same time. We will argue in Section 3.1.4 on page 48 that the suggested model of reload costs is strong enough to handle also problems with this mixed cost structure.

For natural applications it is useful to assume that for each node the reload cost function satisfies the triangle inequality (see Definition 3.3 for a formal definition). Figure 3.1 shows a magnification of a node which is incident with three different carriers. A violation of the triangle inequality would imply that the cost of reloading a good between two carriers, say directly from carrier a to carrier b, would be larger than to involve a third party c and perform two reload operations a–c and c–b. It is obvious that this cost structure is very unlikely in practice.

We investigate three problems in this chapter. The first problem, MINIMUM RELOAD COST PATH, consists of finding a path of minimum reload costs connecting two given nodes. The ideas are further used as a basis for developing an algorithm for MINIMUM RELOAD COST RADIUS SPANNING TREE. This problem consists in finding a spanning tree with minimal reload cost radius with respect to a given center node. Such a tree can be used as a replacement for a "shortest reload cost path tree", since we will show that the latter structure does not exist. Finally, we investigate MINIMUM RELOAD COST DIAMETER SPANNING TREE, which consists in the search for a spanning tree of minimum diameter with respect to reload costs. This is motivated by the fact that the diameter of the tree is an upper bound on the reload costs between an arbitrary pair of nodes.

3.1 Preliminaries and Problem Formulation

3.1.1 Reload Costs

The model of reload costs differs from classical edge or node cost models in the following way: Classical cost functions map single objects (edges or nodes) to costs. In contrast to that, reload costs (which arise at nodes) depend on the *pair* of colors of incoming and outgoing edges used by a traversal through the node. For technical reasons, we require reload cost functions to satisfy some conditions:

Definition 3.1 (Reload Cost Function)
*Let X be a finite set of colors. A function $c\colon X^2 \to \mathbb{N}_0$ is called a **reload cost function**, if for all pairs $x_1, x_2 \in X$*

1. $c(x_1, x_2) = c(x_2, x_1)$,
2. $x_1 = x_2 \implies c(x_1, x_2) = 0$,
3. $x_1 \neq x_2 \implies c(x_1, x_2) > 0$.

Reload costs are defined on a set of colors. In order to make reload costs useable for graphs it is necessary to have a graph with edge colors.

Definition 3.2 (Graph with Reload Costs)
*Let $G = (V, E)$ be an undirected graph with parallel edges allowed. Let $\chi\colon E \to X$ be a mapping from the set of edges to a finite set X of colors. Let $c\colon X^2 \to \mathbb{N}_0$ be a reload cost function. Then we write $G = (V, E, X, \chi, c)$ and name this tuple a **graph with reload costs**.*

As noted before, a natural requirement on reload cost functions is that they satisfy the triangle inequality at each node.

Definition 3.3 (Reload Costs with Triangle Inequality)
*Let $G = (V, E, X, \chi, c)$ be a graph with reload costs. Reload cost function c is said to **satisfy the triangle inequality**, if for all $e_1, e_2, e_3 \in E$ which are incident in one single node,*

$$c(\chi(e_1), \chi(e_3)) \leq c(\chi(e_1), \chi(e_2)) + c(\chi(e_2), \chi(e_3))\,.$$

Observe that Definition 3.3 does not imply that the triangle inequality must hold for all triples of colors.

We continue with a notational convention. Even if reload costs are defined on pairs of colors and not on pairs of edges, it is more convenient to have a notation for costs on pairs of edges. To this end we will use the shorter notation

$$c(e_1, e_2) := c(\chi(e_1), \chi(e_2))$$

throughout this work if there are no ambiguities. Similarly, we use the abbreviation of a quadruple $G = (V, E, \chi, c)$ for a graph with reload costs. However, we keep in mind that the costs can not be chosen arbitrarily for each pair of edges, but still depend on the pair of colors.

3.1.2 Reload Cost Distance

We are now ready to use the notion of reload costs to define reload costs of paths and induced reload cost distances in graphs. Reload costs on a path arise at a node where two consecutive edges have different colors. The total reload costs of a path are determined by summing up all reload costs arising at inner nodes in the path. Consequently, a path of one edge only has reload cost zero.

Definition 3.4 (Reload Costs of a Path)
*Let $G = (V, E, \chi, c)$ be a graph with reload costs. Let $p = (e_1, \dots, e_k)$ be a path in G. The **reload costs of path** p are determined by*

$$c(p) := \sum_{i=1}^{k-1} c(e_i, e_{i+1}) .$$

Naturally, these costs induce a distance function on the graph:

Definition 3.5 (Reload Cost Distance)
Let $G = (V, E, \chi, c)$ be a graph with reload costs. By

$$\mathrm{dist}^c_G(v, w) := \min\{\, c(p) \mid p \text{ is a path from } v \text{ to } w \text{ in } G \,\}$$

*we define the **induced reload cost distance function**.*

In contrast to that, we recall that for a classical edge length function $l\colon E \to \mathbb{N}_0$, the *length* of a path p is given by

$$l(p) := \sum_{i=1}^{k} l(e_i) ,$$

and the *induced length distance function* is determined by

$$\mathrm{dist}^l_G(v, w) := \min\{\, l(p) \mid p \text{ is a path from } v \text{ to } w \text{ in } G \,\} .$$

Notice that we use character "c" to tag functions related to reload costs while we use an "l" to tag functions related to classical lengths.

3.1.3 Problem Formulation

From the motivation of reload costs as discussed in the introduction to this chapter it is strongly recommended that input graphs may contain parallel edges of different color. So we do not assume that the graphs are simple throughout this chapter. Without loss of generality we assume that the graph does not contain parallel edges of the same color.

We are now ready to define the problems under investigation. The first question is to find a path of minimum reload costs between two distinguished nodes of a graph.

Definition 3.6 (MINIMUM RELOAD COST PATH Problem)
An instance of MINIMUM RELOAD COST PATH, MRPATH *for short, consists of a graph* $G = (V, E, \chi, c)$ *with reload costs, and two distinct nodes* $s, t \in V$. *The goal is to find a path* p *from* s *to* t *of minimum reload costs* $c(p)$.

By M^{Δ}RPATH we denote the problem where the reload cost function satisfies the triangle inequality. It will turn out that this problem can be solved in polynomial time.

The second question is to find a spanning tree with minimum reload cost radius with respect to some given root node.

Definition 3.7 (MINIMUM RELOAD COST RADIUS SPANNING TREE Problem)
An instance of MINIMUM RELOAD COST RADIUS SPANNING TREE, MRRADT *for short, is given by a graph* $G = (V, E, \chi, c)$ *with reload costs and a root node* $r \in V$. *The goal is to find a spanning tree* $T \sqsubseteq G$ *of the graph, such that the radius with respect to the reload costs, i. e.,*

$$\operatorname{rad}^{c}(T, r) := \max_{v \in V} \operatorname{dist}^{c}_{T}(r, v)\,,$$

is minimized among all spanning trees of G.

By M^{Δ}RRADT we denote the problem where the reload cost function satisfies the triangle inequality.

Observe that a tree with minimal reload cost radius is not an analogon to a shortest path tree for classical edge lengths. Moreover, the notion of a shortest path tree is meaningless for reload costs: Consider the graph G displayed in Figure 3.2. It consists of edges of three different colors with pairwise reload cost as depicted in the figure. Clearly, the distance from r to w is zero, and the distance from r to v equals 2. Obviously there is no way to construct a tree $T \sqsubset G$ which reflects both distances at the same time.

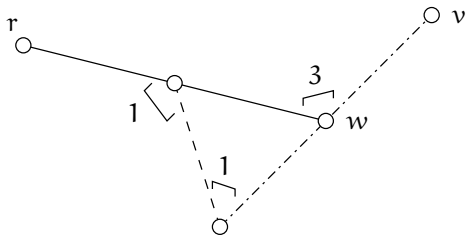

FIGURE 3.2: Graph of maximal degree 3 where the set of shortest paths from the root r is not a tree.

The last problem investigated in this chapter is to find a spanning tree with minimum reload cost diameter. We will give a characterization of both instances where this problem can be solved in polynomial time and NP-hard cases.

Definition 3.8 (MINIMUM RELOAD COST DIAMETER SPANNING TREE)
An instance of MINIMUM RELOAD COST DIAMETER SPANNING TREE*, MRDIAT for short, is given by a graph* $G = (V, E, \chi, c)$ *with reload costs. The goal is to find a spanning tree* $T \sqsubseteq G$ *of the graph, such that the diameter with respect to the reload costs, i. e.,*

$$\operatorname{diam}^c(T) := \max_{v,w \in V} \operatorname{dist}^c_T(v, w)\,,$$

is minimized among all spanning trees of G.

By M$^\Delta$RDIAT we denote the problem where the reload cost function satisfies the triangle inequality.

3.1.4 Combining Reload Cost and Length

Reload costs can be combined with classical edge lengths. Assume that there is a graph with reload cost function c and edge length function l. Naturally, the *combined length* d of a path p is defined to be

$$d(p) := c(p) + l(p)\,.$$

This combined length function induces a distance function dist^{cl} on the graph in the usual way:

$$\operatorname{dist}^{cl}_G(v, w) := \min\{\, d(p) \mid p \text{ is a path from } v \text{ to } w \text{ in } G \,\}\,.$$

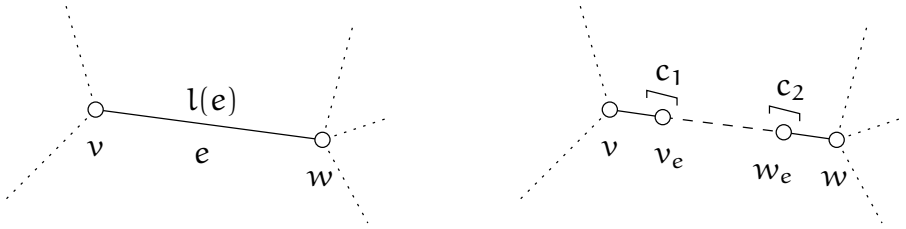

FIGURE 3.3: Reduction from combined length function (left) to sole reload cost function (right). Adjustment $l(e) = c_1 + c_2$ can be guaranteed.

The combined length distance $dist^{cl}$ admits to reformulate variants of all problems noted above. We argue that these variants can be reduced to the case with solely reload costs.

The reduction can be performed in the following way (confer Figure 3.3): For each edge $e = (v, w)$, place two new nodes v_e, w_e onto the edge. This procedure replaces each edge by a path of three edges. Let the color of the two border edges (v, v_e) and (w_e, w) be the color of the original edge, and introduce a new unique color for the central edge (v_e, w_e). Then augment the reload cost function such that the sum of the reload costs arising at the inner nodes v_e, w_e reflects the length of the original edge. Since the augmentation of the reload cost function only takes place at the new nodes of degree 2, it is easy to see that the triangle inequality is not violated by the reduction.

Notice that the resulting graph size is still polynomial in the input size: If the initial graph consists of n nodes, m edges, and k colors, then the new graph contains $n + 2m$ nodes and $3m$ edges with $k + m$ colors. Hence the reduction does not affect the hardness of the problems. Therefore we will stick to the problems with sole reload cost functions in the remainder of this chapter.

3.2 Related Work

The problem corresponding to MINIMUM RELOAD COST DIAMETER SPANNING TREE with respect to classical edge lengths is called MINIMUM DIAMETER SPANNING TREE and well known in literature. It has been shown that even for general edge lengths, the problem is equivalent to the ABSOLUTE 1-CENTER problem [KH79a, Ple81, HT95]. For the geometric case, there has been reported an exact algorithm with running time in $O(|V|^3)$ [HL+91]. Subsequently, following the above mentioned equivalence, there has been found an algorithm with running time $O(|E||V| + |V|^2 \log |V|)$ for general (nonnegative) edge lengths [HT95].

A two criteria variant with both edge lengths and costs is the MINIMUM DIAMETER problem suggested by Plesník [Ple81]. He proved that given a constraint on the total cost of a subgraph it is NP-hard to approximate the optimal diameter within a factor 5/4. This hardness remains valid even for the special cases that $l(e) \equiv l$ for all edges e, or that $l(e) = c(e)$ for all edges e.

3.3 Minimum Reload Cost Path

A first approach to deal with new types of problems is to try to reduce them to a problem which is already known. In the case of reload costs we try to map the reload cost structure to classical edge lengths. It will turn out that this idea works well for the MINIMUM Δ-RELOAD COST PATH problem.

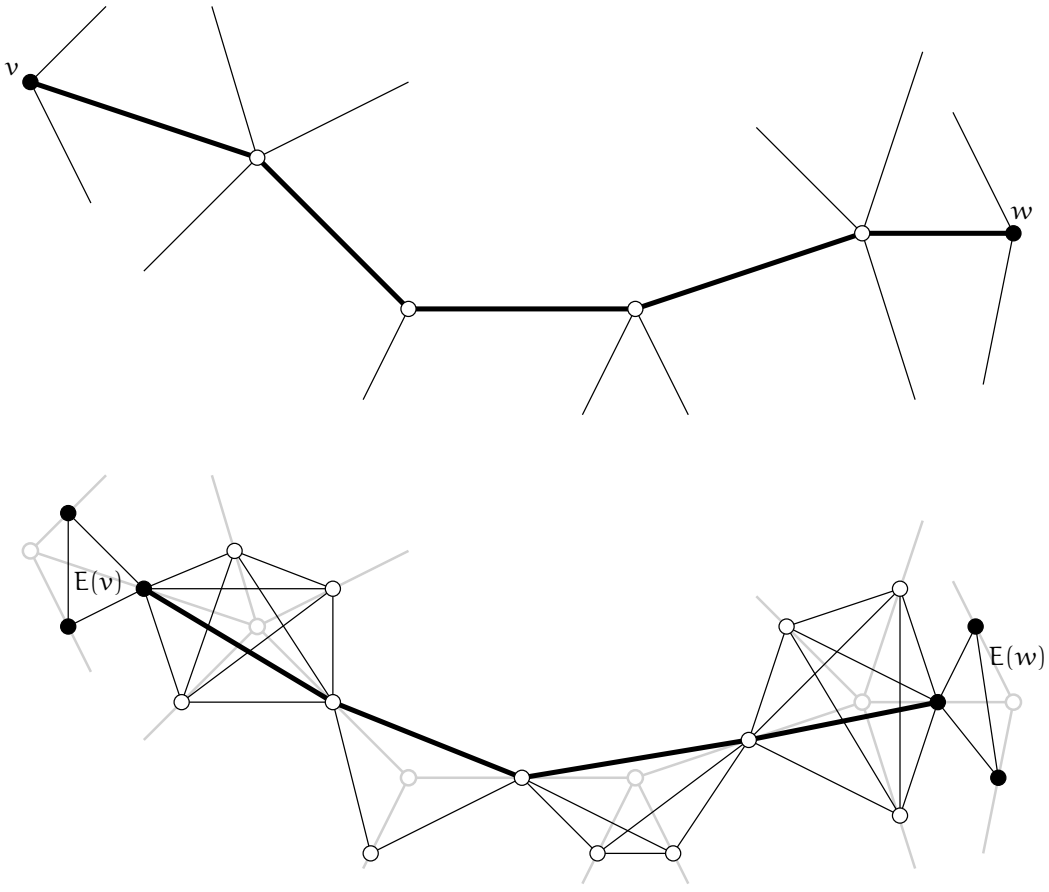

FIGURE 3.4: Reduction from reload costs on a path between v and w (top, thick edges) to edge lengths in the line graph (bottom, solid edges).

Consider a single node in a graph with reload costs. Each pair of edges incident to that node induces reload costs. If we replace the node by a clique whose order equals the degree of the node, then there is a one-to-one correspondence between pairs of edges of the original graph and the edges of the clique. Now we can map the reload costs to a length function on the clique edges. (See Figure 3.4 on the facing page for an illustration of the idea.) Then we can use classical algorithms to determine a shortest path in the new graph. After that, we must retransform the path in the new graph to a path in the original graph. The clue of this transformation is, that the length of the path in the new graph equals the reload cost of the transformed path in the original graph. Below we will argue that this claim is correct. The graph which is constructed above is well known in literature as the *line graph*.

Definition 3.9 (Line Graph)
Let $G = (V, E)$ *be a graph. The* ***line graph*** $H = (V_H, E_H)$ *of* G *is defined by*

$$V_H := E$$
$$E_H := \{\, (e_1, e_2) \mid e_1 \text{ and } e_2 \text{ are incident in } G \,\}.$$

To complete the construction we must describe how the reload costs are transformed to edge lengths in the line graph. This transformation is straightforward. Let $G = (V, E, \chi, c)$ be a graph with reload costs, $H = (V_H, E_H)$ its line graph. For each edge $(e_1, e_2) \in E_H$, define its length to be

$$l((e_1, e_2)) := c(e_1, e_2)\,. \tag{3.1}$$

Observe that each node in G is represented by a clique in H. For each node $v \in V$, denote by

$$E(v) := \{\, e \in E \mid e \text{ is incident with } v \,\}$$

the edges of G which are incident on v. Then the node set $E(v) \subseteq V_H$ describes the clique which represents node v in the line graph.

The correlation between distances in G and H is shown by the following lemma.

Lemma 3.10
Let $G = (V, E, \chi, c)$ *be a graph with reload costs,* H *its line graph with edge length function* l *as described above. Then for each pair* $v_1, v_2 \in V$ *of nodes, we have*

$$\mathrm{dist}^l_H(E(v_1), E(v_2)) = \mathrm{dist}^c_G(v_1, v_2)\,, \tag{3.2}$$

where the l-distance between sets of nodes is defined as usual by

$$\mathrm{dist}_H^l(S_1, S_2) := \min\{\, \mathrm{dist}_H^l(s_1, s_2) \mid s_1 \in S_1, s_2 \in S_2 \,\}.$$

Proof. We show (3.2) by proving two inequalities.

1. $\mathrm{dist}_G^c(v_1, v_2) \leq \mathrm{dist}_H^l(E(v_1), E(v_2))$:
   Let $p = (e_1, e_2, \ldots, e_q)$, $e_i \in V_H$, be the sequence of nodes visited by a shortest path in H between the node sets $E(v_1)$ and $E(v_2)$. By construction of the line graph, for all $i = 2, \ldots, q$ the edges e_{i-1} and e_i are incident in G, therefore the sequence p when viewed as a sequence of edges in E is a path in G. Since $e_1 \in E(v_1)$, we can assume that the start node of path p is v_1 (otherwise remove the first edge and relabel the edges). Similarly, we can assume that the end node of p is v_2. By construction of the length function we have $l((e_{i-1}, e_i)) = c(e_{i-1}, e_i)$ for each $i = 2, \ldots, q$. Summing over all edges yields

   $$\mathrm{dist}_G^c(v_1, v_2) \leq c(p) = l(p) = \mathrm{dist}_H^l(E(v_1), E(v_2)),$$

   which proves the first claim.
2. $\mathrm{dist}_H^l(E(v_1), E(v_2)) \leq \mathrm{dist}_G^c(v_1, v_2)$:
   Let $p = (e_1, \ldots, e_q)$ be the sequence of edges of a reload cost shortest path in G from v_1 to v_2. Then p is the sequence of nodes of a path in H from $E(v_1)$ to $E(v_2)$. By construction of l, we have $l(p) = c(p)$. Hence

   $$\mathrm{dist}_H^l(E(v_1), E(v_2)) \leq l(p) = c(p) = \mathrm{dist}_G^c(v_1, v_2)$$

   as stated above. □

The observations noted above and the result of Lemma 3.10 admit to construct a simple and in particular polynomial time algorithm for solving MINIMUM Δ-RELOAD COST PATH. The algorithm basically reduces the problem to the problem of finding a shortest path with respect to classical edge lengths. A detailed description is noted in Algorithm 3.5 on the next page.

We estimate the running time of Algorithm 3.5. Let n and m be the number of nodes and edges of G, respectively. The line graph H consists of m nodes and $O(n^3)$ edges. The main effort of Algorithm 3.5 is the computation of a shortest path in Step 3. If we employ Thorup's algorithm [Tho97], this step can be accomplished in time $O(|E(H)| + |V(H)|) \subseteq O(n^3 + m)$. This is summarized in the following corollary:

Corollary 3.11 (Solution of M^{Δ}RPATH)
Problem MINIMUM Δ-RELOAD COST PATH *can be solved by Algorithm 3.5 in polynomial time* $O(n^3 + m)$. □

Input: A graph $G = (V, E, \chi, c)$ with reload costs, two nodes $s, t \in V$

1 Construct line graph H
2 Setup edge length function l on H as described in (3.1)
3 Compute a shortest path with respect to l from $E(s)$ to $E(t)$
4 Transform resulting path to a path in G { *described in the proof of Lemma 3.10* }

Output: resulting path

ALGORITHM 3.5: Algorithm for solving MINIMUM Δ-RELOAD COST PATH.

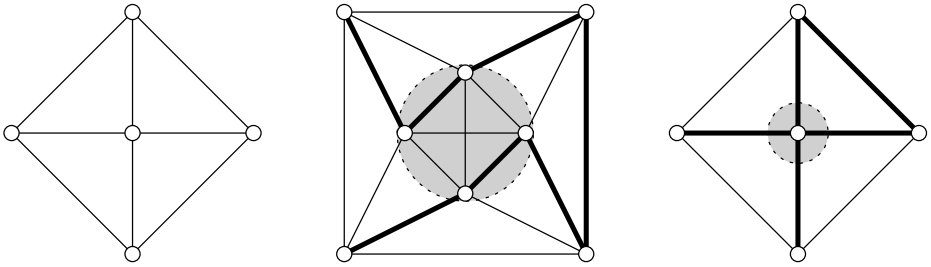

FIGURE 3.6: Basic example of a graph (left) with its line graph (center). The reduction of a tree in the line graph (thick lines) is not necessarily a tree in the original graph (right).

If the underlying graph class is more complicated than the class of paths, the idea of reducing reload cost problems to problem versions with classical edge lengths does no longer work. In particular, for the case of spanning trees there are mainly two reasons why this approach must fail.

The first reason is the fact that the reduction from a tree in the line graph to a subgraph in the original graph according to the construction described in the proof of Lemma 3.10 does not necessarily yield a tree. In Figure 3.6 we give a brief counter example for the case where the resulting subgraph contains a cycle.

Secondly, the line graph contains much more nodes than the original graph. In order to get a spanning tree in the original graph after a supposed reduction, it is sufficient to compute a tree in the line graph which spans at least one node of each clique. If one computed a tree spanning all nodes, the algorithm could be directed to a completely wrong track since the weight of idle leaf edges affects the shape of the tree substantially. For the case of paths we could deal with this problem by letting the path originate in an arbitrary node of the clique associated to the starting node. The computation of a tree spanning only one node

per clique is known as the GROUP STEINER TREE problem. Since this problem is known to be NP-hard and hard to approximate [GKR98], there is no advantage in using this approach for solving the original problem.

After that considerations it is not surprising that MINIMUM RELOAD COST RADIUS SPANNING TREE and MINIMUM RELOAD COST DIAMETER SPANNING TREE turn out to be intractable. In particular we will show in Section 3.6 that these problems are NP-hard for graphs of maximal degree 5. Nevertheless, for graphs with maximal degree 3 we provide polynomial time algorithms for both problems in the sequel.

3.4 Minimum Reload Cost Radius Spanning Tree

We continue with problem MINIMUM Δ-RELOAD COST RADIUS SPANNING TREE restricted to graphs of maximal degree 3. We argue that the problem can be solved by applying a shortest path algorithm on the line graph and performing a simple post-processing in order to construct a tree in the original graph.

Let $G = (V, E)$ be the input graph (with root node $r \in V$) and $H_0 = (V_{H_0}, E_{H_0})$ its line graph. We add to the line graph the root node r and up to three edges $\{ (r, e) \mid e \in E(r) \}$ of zero length which connect r to the neighborhood $E(r)$.

Let H be the resulting graph. From Lemma 3.10 we can conclude that

$$\mathrm{dist}^l_H(r, E(v)) = \mathrm{dist}^c_G(r, v) \tag{3.3}$$

for all nodes $v \in V$.

Now compute a shortest path tree T of graph H with root r. Consequently, for each node $v \in V$ we have

$$\mathrm{dist}^l_T(r, E(v)) = \mathrm{dist}^l_H(r, E(v)) \overset{(3.3)}{=} \mathrm{dist}^c_G(r, v)\,. \tag{3.4}$$

Let $v_0 \in V$ be a node where $\mathrm{dist}^l_T(r, E(v_0))$ is maximized. By (3.4), this node determines the reload cost radius of graph G with root r. The goal is now to construct a subtree S of G which constitutes a path from r to v_0 of length equal to the radius $\mathrm{dist}^l_T(r, E(v_0))$, and connects all remaining nodes by paths whose reload cost length does not exceed this length. We claim that this task is performed by Algorithm 3.7 on the next page. We refer also to Figure 3.8 on page 56 for an illustrating example of the algorithm.

The algorithm maintains for each node $v \in V$ a pointer $p_S(v)$ to a node from V. Denote by

$$S_p := \{ (v, p_S(v)) \mid v \in V \text{ and } p_S(v) \text{ is currently defined} \}$$

Input: Graph $G = (V, E, \chi, c)$ with reload cost,
maximum node degree 3
root node $r \in V$

Let H be the line graph of G, augmented by node r and
up to three edges connecting r to $E(r)$ as described in the text
Compute a shortest path tree T in graph H with root r
Assume that T is directed towards the root.
For each node $e \neq r$ of T, let $p_T(e)$ be the ancestor in T
for $v \in V$ **do**
  Call RELAX(v)
end for

Output: Tree S with edge set $\{ (v, p_S(v)) \mid v \in V \setminus \{r\} \}$

RELAX($v \in V$)

if $v \neq r$ and $p_S(v)$ not defined yet **then**
  Choose $e \in E(v)$ which minimizes $\text{dist}^l_T(r, e)$.
  If this is not unique, assure that $p_T(e) \notin E(v)$
  This defines $p_v \in V$ via $e = (p_v, v)$
  Set $p_S(v) := p_v$
  Call ENFORCE(p_v VIA e)
end if

ENFORCE($v \in V$ VIA $s \in E$)

if $v \neq r$ and v not marked as enforced yet **then**
  Let $e := p_T(s)$
  If also $p_T(p_T(s)) \in E(v)$, then let $e := p_T(p_T(v))$
  { *Observe that* $p_T(e) \notin E(v)$ *now* }
  This defines $p_v \in V$ via $e = (p_v, v)$
  Set $p_S(v) := p_v$
  Mark v as enforced
  Call ENFORCE(p_v VIA e)
end if

ALGORITHM 3.7: Algorithm solving M^{Δ}RRADT.

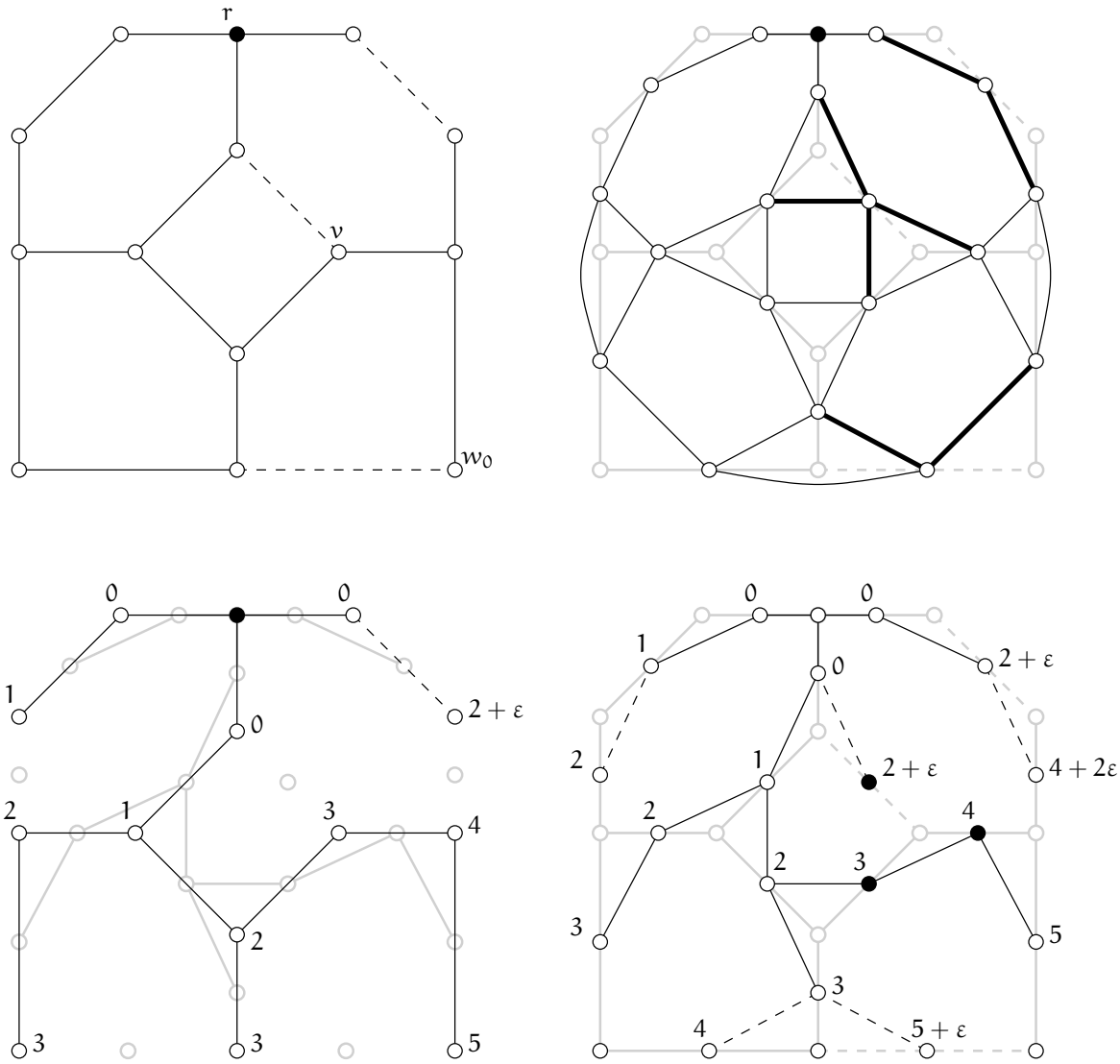

FIGURE 3.8: Example of Algorithm 3.7. — Top left: Initial graph. Its edges have pairwise different colors. Reload costs between equal shaped edges equal 1, reload cost between different shaped edges equal $2+\varepsilon$. — Top right: Resulting line graph H. Thick edges have length $2+\varepsilon$. — Bottom right: Shortest path tree with node distances. Solid lines are considered within a sequence of calls to ENFORCE. Observe further that $\min_{e\in E(v)} \mathrm{dist}^{l}_{H}(r,e) = 2+\varepsilon < 3$ (set $E(v)$ is marked black). — Bottom left: Resulting solution. Since v was enforced by its right neighbor, the distance of v is not optimal.

the subset of the edges of the initial graph G which is induced by those nodes where the pointer p_S is currently defined. We first state an easy observation:

Observation 3.12
If a node $v \in V$ is marked enforced, subsequently it remains marked enforced, and the pointer $p_S(v)$ remains unchanged.

Notice that the pointer $p_S(v)$ *may* change after assigned in procedure RELAX for the first time.

When the algorithm terminates, we claim that the output set S_p is a tree. To this end, we first prove the following statement:

Lemma 3.13
At no time, S_p contains a cycle.

Proof. Assume for contradiction that S_p contains a cycle. Since set S_p is induced by the pointers $p_S \colon V \to V$, the cycle must in fact be a directed cycle.

Consider an arbitrary edge $e = (v, p(v))$ in S_p where $p(v) \neq r$. After this edge is added to S_p (Step 14 or Step 24), procedure ENFORCE is called for node $p(v)$.

Procedure ENFORCE($p(v)$ VIA e) adds an edge $f = (p(v), p(p(v)))$ to S_p. This edge does not depend on e due to the selection in Step 22. Hence, all observations on the properties of ENFORCE are true regardless whether ENFORCE is actually executed at this time or is skipped since it has already been executed before.

By selection of f in Step 22 it is guaranteed that the tree T contains a directed path from node e to f.

We turn back to graph G: edge e was chosen arbitrarily on the cycle in S_p. Hence, the above argument can be applied to each pair e, f of subsequent edges on the directed cycle in S_p. This implies a directed cycle in tree T. □

Since at the end of the algorithm, procedure RELAX has been called for all nodes $v \in V \setminus \{r\}$ (Step 5), the pointers $p_S(v)$ have been defined for all nodes $v \in V \setminus \{r\}$. Thus, the out-degree equals 1 for each node except the root. From Lemma 3.13 we conclude:

Corollary 3.14
When Algorithm 3.7 terminates, then S_p is a directed spanning tree in G rooted towards r. □

We will now estimate the distances in S_p from particular nodes to the root.

Lemma 3.15
If for some $v, w \in V$, $e \in E$, $e = (v, w)$, procedure ENFORCE(v VIA e) *is called within procedure* ENFORCE *(Step 26), then at the end of the algorithm,*

$$\mathrm{dist}^c_{S_p}(r, w) = \mathrm{dist}^l_T(r, e)\,.$$

Proof. The proof is by induction on the number k of edges on the path from r to e in T. If $k = 1$, then $v = r$, and the claim is trivial.

Assume that the claim is true for k. Consider a call to ENFORCE(v VIA s) where the path from r to $s \in E$ in T uses $k+1$ edges. Then, edge s in finally added to S_p and in Step 26, procedure ENFORCE(p_v VIA e) is called for an ancestor e of s in T. Hence, we can apply the induction hypothesis, and get

$$\begin{aligned}\mathrm{dist}^c_{S_p}(r, w) &= \mathrm{dist}^c_{S_p}(r, v) + c(e, s)\\ &= \mathrm{dist}^c_{S_p}(r, v) + l((e, s))\\ &= \mathrm{dist}^l_T(r, v) + l((e, s))\\ &= \mathrm{dist}^l_T(r, w)\end{aligned} \tag{3.5}$$

when the current chain of calls to ENFORCE terminates. Since we have assumed that the initial call to ENFORCE(v VIA s) is within ENFORCE, node w gets marked as enforced. By Observation 3.12 it follows that (3.5) holds true until the total algorithm terminates. □

Lemma 3.16
Let $v \in V$ be a node where $p_S(v)$ is not defined at the beginning of Step 5. Then, at end of Step 5, we have

$$\mathrm{dist}^c_{S_p}(r, v) = \mathrm{dist}^c_G(r, v)\,.$$

Proof. The proof for $\mathrm{dist}^c_{S_p}(r, v) = \mathrm{dist}^l_T(r, E(v))$ is the same as the proof of Lemma 3.15, since the minimization over $E(v)$ takes place in Step 12. (Observe that in contrast to the proof of Lemma 3.15, in the current situation node v is not marked as enforced. Hence we can not guarantee that the distance of node v remains unchanged until the algorithm terminates.) With (3.4) the claim follows. □

We point out that the above result does not mean that the distance relation holds for all nodes $v \in V$ at the end of the algorithm: If a node v is marked enforced later on, the distance $\mathrm{dist}^c_{S_b}(r, v)$ may increase. However, we can guarantee that this is not the case for all nodes which determine the reload cost radius of the spanning tree:

Theorem 3.17 (Correctness of Algorithm 3.7)
Let $w_0 \in V$ be a node where $\mathrm{dist}^c_G(r, w_0)$ is maximized. Then,

$$\mathrm{dist}^c_{S_p}(r, w_0) = \mathrm{dist}^c_G(r, w_0)$$

at the end of the algorithm.

Proof. Observe that since the algorithm calls RELAX for each node except the root, the claim immediately follows from Lemma 3.16 if we can rule out a call to ENFORCE(w_0 VIA e) where $\mathrm{dist}^l_T(r, e) > \mathrm{dist}^c_G(r, w_0)$.

Assume for the sake of a contradiction that a call to ENFORCE(w_0 VIA e) happens where $\mathrm{dist}^l_T(r, e) > \mathrm{dist}^c_G(r, w_0)$. Let v be the current node when this call is performed in Step 15 or Step 26. Then,

$$\mathrm{dist}^c_G(r, v) = \mathrm{dist}^l_T(r, e) > \mathrm{dist}^c_G(r, w_0)$$

contradicting the fact that node w_0 maximizes the distance dist^c in G. □

The correctness of the algorithm follows with Corollary 3.14 which guarantees that the solution is in fact a tree.

Theorem 3.18 (Solution of M$^\Delta$RRADT)
Algorithm 3.7 solves M$^\Delta$RRADT *on graphs with maximal degree 3 in linear time* $O(|V|)$.

Proof. It remains to estimate the running time of the algorithm. Let $G = (V, E)$ be the input graph with maximal degree 3. One observes easily that

$$2|E| = \sum_{v \in V} d(v) \leq 3|V|\,. \tag{3.6}$$

The line graph H_0 consists of $|E| \leq 3/2 \cdot |V|$ nodes and $3|V|$ edges. The shortest path tree algorithm of Thorup runs in time $O(|E(H)| + |V(H)|) \subseteq O(|V|)$ [Tho97]. The further calculations of Algorithm 3.7 on page 55 can be estimated as follows: Each execution of RELAX or ENFORCE needs constant time. Moreover, it is guaranteed that the procedures are not executed more than once for each node. Hence the overall time needed is in $O(|V|)$. □

3.5 Minimum Reload Cost Diameter Spanning Tree

In this section we provide an algorithm which solves M$^\Delta$RDIAT on graphs with maximal degree 3. Notice that we assume that the reload costs satisfy the triangle inequality.

The main idea is a refinement of the mapping idea explained in Lemma 3.10. In Section 3.3 on page 50 we argued that the idea can not be transferred from paths to the general case of trees. However, a careful analysis of the problem shows how to apply the idea in the special case of degree-3 bounded graphs.

The key point of the idea of mapping reload costs to edge lengths is to replace each node by a clique. A similar approach which works for graphs of low degree is to leave the edges unchanged and try to set the edge length function in such a way that the sum of the edge lengths on a path equals the sum of the reload cost along that path. The algorithm for solving M^{Δ}RDIAT has three main steps:

1. Setup an auxiliary graph whose edge lengths reflect the reload cost of the original graph.
2. Solve problem MINIMUM DIAMETER SPANNING TREE on the auxiliary graph.
3. Transform the resulting tree into a tree in the original graph.

We outline the details in the following.

3.5.1 Setting Up the Auxiliary Graph

Given a graph $G = (V, E)$, we construct an auxiliary graph H by placing a new node on each edge. Formally, we create a graph H with node set $V(H) := V \cup E^{\varphi}$, where E^{φ} is a set of (new) nodes with cardinality $|E^{\varphi}| = |E|$. In this notation we use φ to denote the fact that there is a suitable bijection between the corresponding sets. The edge set $E(H)$ of the auxiliary graph is constructed as follows: For each edge $e = (v, w) \in E$, graph H contains the two edges (v, e^{φ}), and (e^{φ}, w). The construction is illustrated in Figure 3.9 (b).

The length of edges incident to a node from set V is adjusted such that the sum of the edge lengths on a walk through the node equals the reload costs of the related walk in the original graph, i.e.,

$$l(e_1^{\varphi}, v) + l(v, e_2^{\varphi}) = c(e_1, e_2) \qquad \text{for all } v \in V,\ e_i \text{ incident to } v. \tag{3.7}$$

For a node v of degree d, the set of equations (3.7) consists of $\binom{d}{2}$ equations with d variables. Unfortunately the system of equations will not have a solution in general if $d > 3$. This is the main reason why we restrict the graph class to graphs of maximum degree 3.

However, for degrees $d \leq 3$, (3.7) implies that the edge length function l is well defined. We use the case $d = 3$ to show this proposition: Assume that there is a node $v \in V$ which is incident to exactly three edges $e_1, e_2, e_3 \in E$. Then (3.7)

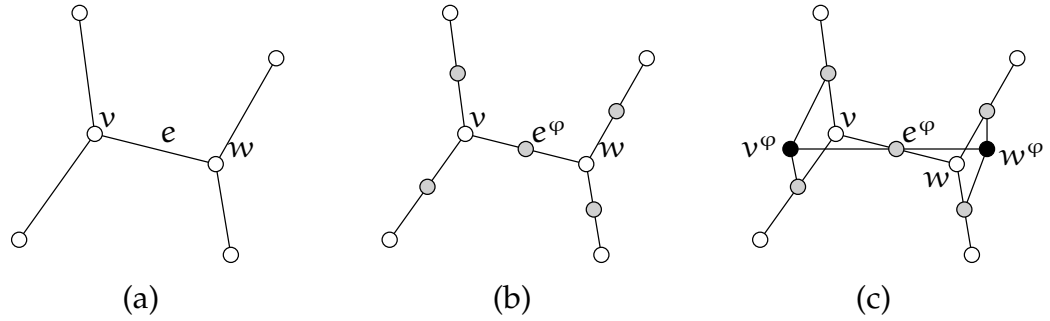

FIGURE 3.9: The original graph (a), and two steps of the construction of the auxiliary graph (b,c).

becomes

$$\begin{aligned} 2 \cdot l(e_1^\varphi, v) &= c(e_1, e_2) - c(e_2, e_3) + c(e_1, e_3) \\ 2 \cdot l(e_2^\varphi, v) &= c(e_2, e_3) - c(e_3, e_1) + c(e_2, e_1) \\ 2 \cdot l(e_3^\varphi, v) &= c(e_3, e_1) - c(e_1, e_2) + c(e_3, e_2)\,. \end{aligned} \tag{3.8}$$

It remains to state a relation between the distances in terms of reload costs in G and the distances in terms of edge lengths in the auxiliary graph H. The goal is to find a formulation similar to that of Lemma 3.10 on page 51. To this end, we complete the construction of the auxiliary graph (see Figure 3.9 (c)): The node set of H is augmented by set V^φ which is a copy of the node set V. For each edge (v, e^φ), add a new edge (v^φ, e^φ) to the edge set.

Recall from Lemma 3.10 that $\mathrm{dist}_G^c(v_1, v_2) = \mathrm{dist}_H^l(E(v_1), E(v_2))$ for all nodes $v_1, v_2 \in V$. This shows that the mapping of reload cost distances to length distances is easy with the exception of the two endpoints of a path: the minimization over the sets $E(v_i)$ is unwieldily for applying known algorithms.

This is the reason why we introduced the new level V^φ of nodes. Each node from v^φ is connected to all nodes from $E(v)$. For convenience, we refer to those connecting edges, i. e., all edges incident with nodes from V^φ, as Ω-*edges* in the sequel. Assume for now that one can assure that Ω-edges appear only at endpoints of paths. Then from the observation

$$\mathrm{dist}_H^l(v_1^\varphi, v_2^\varphi) = \mathrm{dist}_H^l(v_1^\varphi, E(v_1)) + \mathrm{dist}_H^l(E(v_1), E(v_2)) + \mathrm{dist}_H^l(E(v_2), v_2^\varphi) \tag{3.9}$$

one can derive a relationship between $\mathrm{dist}_H^l(v_1^\varphi, v_2^\varphi)$ and $\mathrm{dist}_G^c(v_1, v_2)$. The relationship cannot be equality: From (3.9) and Lemma 3.10 one would conclude that the weight of all Ω-edges equals zero. It is easy to observe that in this case

all distances in the line graph become zero as well. However, it will turn out that setting the length of the Ω-edges to some large constant

$$\Omega > 2 \cdot \max\{\, c(x, x') \mid x, x' \in X \,\}. \tag{3.10}$$

is sufficient to recover the suggested approach:

Lemma 3.19 (Distance mapping)
Let $G = (V, E)$ be a graph with reload cost function c. Let H be the auxiliary graph constructed as described above, and l be the computed edge length function. Then for each pair $v, w \in V$,

$$\mathrm{dist}^c_G(v, w) = \mathrm{dist}^l_H(v^\varphi, w^\varphi) - 2\Omega\,.$$

Further, if c satisfies the triangle inequality, then $l \geq 0$.

Proof. The proof of the first claim uses arguments very similar to that of the proof of Lemma 3.10 on page 51. There are only small changes necessary: Equation (3.1) must be replaced by (3.7). The mechanism of Lemma 3.10 where the path is allowed to start in any node from set $E(v)$ is reflected in the current lemma by using an arbitrary Ω-edge incident to node v^φ. This implies an additive offset of 2Ω between path length and its reload cost. Notice that due to the construction Ω-edges appear never inside a path of minimal length.

To prove the second claim, namely $l \geq 0$, it suffices to plug in the triangle inequality of Definition 3.3 into the solution of (3.7). For the case of degree $d = 3$ the result is immediate from (3.8). The remaining cases $d = 1, 2$ are easy. □

We remark that from (3.8) we can conclude that for each non-Ω-edge e in the auxiliary graph its length satisfies

$$l(e) < 1/2 \cdot \Omega\,. \tag{3.11}$$

This completes the construction of the auxiliary graph. Algorithm 3.10 on the next page summarizes the construction.

Lemma 3.19 implies that the reload costs can be reduced to classical edge lengths. This suggests to call known algorithms for the problem of finding a minimum diameter spanning tree of the auxiliary graph at this point. A suitable and efficient algorithm for this task has been reported by Hassin and Tamir [HT95].

Input: Graph $G = (V, E, \chi, c)$ with reload cost, maximum node degree 3

1 Let V^φ and E^Φ be copies of V and E, respectively
2 Let $V(H) := V \cup V^\varphi \cup E^\varphi$
3 Let $E(H) := \varnothing$
4 **for** each edge $e \in E$, $e = (v, w)$ **do**
5 Add edges (v, e^φ), (e^φ, w) of length $l(.)$ according to (3.7) to edge set $E(H)$
6 Add edges (v^φ, e^φ), (e^φ, w^φ) of length $l(.) = \Omega$ to edge set $E(H)$
7 **end for**

Output: Auxiliary graph $H = (V(H), E(H))$ with edge lengths l

ALGORITHM 3.10: Setting up the auxiliary graph.

FIGURE 3.11: Images under functions $\check{\Phi}$ and $\hat{\Phi}$.

3.5.2 Projections

In this section we introduce the notion of *projection*. A projection is a mapping from a subset of the auxiliary graph into the original graph. We will use this projection later on to transform the solution found in the auxiliary graph into a solution in the original graph.

The notion "projection" comes from a point of view where we identify two layers in the auxiliary graph: the lower layer consists of the edges incident with nodes from V, while the upper layer contains all Ω-edges. A natural transformation from the auxiliary graph to the original graph is to insert an edge into the original graph whenever the solution in the auxiliary graph contains a corresponding edge from lower or upper layer. The resulting solution can be seen as a kind of "shadow" of the auxiliary solution.

To be more formally, we introduce two functions $\hat{\Phi}, \check{\Phi}$ on the set of edges of the auxiliary graph (see Figure 3.11) which establish a correspondence between

the edges of the two layers:

$$\check{\Phi}((v, e^{\varphi})) := (v, e^{\varphi}) \qquad \hat{\Phi}((v, e^{\varphi})) := (v^{\varphi}, e^{\varphi})$$
$$\check{\Phi}((v^{\varphi}, e^{\varphi})) := (v, e^{\varphi}) \qquad \hat{\Phi}((v^{\varphi}, e^{\varphi})) := (v^{\varphi}, e^{\varphi})$$

where $e = (v, w)$ is an arbitrary edge in the original graph with endpoints $v, w \in V$, and corresponding nodes $v^{\varphi}, w^{\varphi} \in V^{\varphi}$.

The projection of an edge set from the auxiliary graph into the original graph is performed in two steps: First the application of $\check{\Phi}$ projects the edge set into one layer. Then we consider for each edge $e = (v, w)$ of the original graph the corresponding node $e^{\varphi} \in E^{\varphi}$: Edge e is added to the solution if and only if node e^{φ} is incident with two edges after applying $\check{\Phi}$. This is stated in the following definition. Figure 3.12 illustrates the construction.

Definition 3.20 (Projection)
Let A be an edge subset of the auxiliary graph. By

$$\mathrm{prj}(A) := \left\{ e = (v, w) \mid (v, e^{\varphi}) \in \check{\Phi}(A) \wedge (w, e^{\varphi}) \in \check{\Phi}(A) \right\}$$

we define an edge subset of the original graph. This subset $\mathrm{prj}(A)$ *is called the* ***projection*** *of* A*.*

There is no well-defined inverse mapping of the projection. We decided to choose the following definition:

Definition 3.21 (Lifting)
Let A be an edge subset of the original graph. By

$$\mathrm{lift}(A) := \{ (v, e^{\varphi}), (e^{\varphi}, w) \mid e = (v, w) \in A \}$$

we define an edge subset of the auxiliary graph. This subset $\mathrm{lift}(A)$ *is called the* ***lifting*** *of* A*.*

In the remaining part of this section we make the projection useable as a tool for problem reduction. This is done by stating a correlation between the diameter of a spanning tree in the auxiliary graph and the diameter of its projection. As one could expect, it will turn out that there is an offset of 2Ω under certain conditions. We introduce the notion of a *dangling edge* before we formulate the main theorem of this section.

Definition 3.22 (Dangling edge, Hook)
An Ω-edge in a tree T is called ***dangling*** *edge, if its image under $\check{\Phi}$ is not contained in T. A node from E^{φ} is called a* ***hook****, if the set of incident non-Ω-edges contains exactly one edge.*

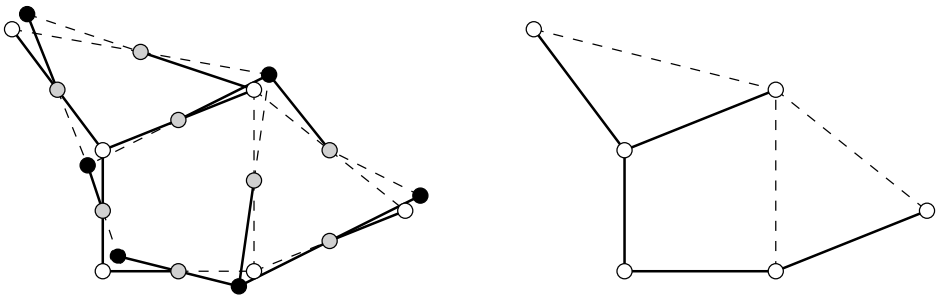

FIGURE 3.12: Tree in the auxiliary graph (solid lines, left) and its projection in the original graph (solid lines, right).

FIGURE 3.13: Tree in the auxiliary graph (left, solid lines). Its projection (right, solid lines) may contain cycles.

Observe that a tree without dangling edges has a special structure: all nodes from V^φ appear as leaves. The proof is easy: Assume there is a node from V^φ of degree 2 or greater. Consider two incident Ω-edges. Since there are no dangling edges, the images of the incident edges under $\check{\Phi}$ must also be contained in the tree. Moreover, the four edges form a cycle.

Observation 3.23
If a tree contains no dangling edges, then all nodes from V^φ are leaves of the tree. □

The projection of a tree is not necessarily a tree. In fact, it may induce a cycle. An example is given in Figure 3.13. However, we can formulate conditions where the projection of a tree is itself a tree.

Lemma 3.24
Let T_{aux} be a tree in the auxiliary graph spanning V^φ. Assume that T_{aux} contains no dangling edges. Then the projection $T := \mathrm{prj}(T_{aux})$ is a spanning tree.

Proof. Observe that since T_{aux} spans V^φ and does not contain dangling edges, T_{aux} must span node set V also.

We first argue that T is connected. Consider an arbitrary pair v^φ, w^φ of nodes. Let p be the connecting path in T_{aux}. Since T_{aux} contains no dangling edges, the edge set $\check{\Phi}(p)$ is part of T_{aux} and is itself a tree. Let p' be the path between v and w in $\check{\Phi}(p)$. Then $\mathrm{prj}(p')$ is a path in the original graph between v and w. Since v^φ, w^φ were chosen arbitrarily, this shows that T is connected.

It remains to show that T is cycle free. Assume there is a cycle c in T. Let $c' := \mathrm{lift}(c)$ be the lifting of cycle c. Since T_{aux} is cycle free, there is a lower layer edge $(v, e^\varphi) \in c'$ which is not in T_{aux}. Since c is part of the projection of T_{aux}, T_{aux} must contain the corresponding upper layer edge (v^φ, e^φ). This contradicts the fact that T_{aux} does not contain dangling edges. □

The correlation between the diameter of a tree and that of its projection is stated by Theorem 3.25.

Theorem 3.25 (Distance Mapping on the Tree)
Let T_{aux} be a tree in the auxiliary graph spanning node set V^φ. Assume that T_{aux} contains no dangling edges and no hooks. Then

$$\mathrm{diam}^c(\mathrm{prj}(T_{aux})) = \mathrm{diam}^l(T_{aux}) - 2\Omega\,.$$

Proof. Let $T := \mathrm{prj}(T_{aux})$. Let p be a longest path in T, i. e., $c(p) = \mathrm{diam}^c(T)$. Denote by $v, w \in V$ the endpoints of p (see Figure 3.14 (a)). Since T_{aux} contains no dangling edges, the lifting $\mathrm{lift}(p)$ must be part of T_{aux}. Take the path $\mathrm{lift}(p)$, remove one edge at each end, and name the resulting path q. Then

$$l(q) = c(p) \tag{3.12}$$

by construction of l (see Figure 3.14 (b)).

Consider the path from v to v^φ in T_{aux} (see Figure 3.14 (c)). This path must meet the endpoint e^φ of q: Otherwise it would have to visit a node f^φ different from all nodes of q. Since this node is not a hook, it must be incident with another non-Ω-edge to a node $x \in V$. The projection of those two edges would yield an edge (x, v) which would augment path p in T. This is a contradiction to the fact that p was a longest path.

The same argument applies to the other endpoint of q. Hence we can augment path q in T_{aux} by one Ω-edge at each side. Call the resulting path q'. Therefore,

$$\mathrm{diam}^c(\mathrm{prj}(T_{aux})) = c(p) = l(q) = l(q') - 2\Omega \le \mathrm{diam}^l(T_{aux}) - 2\Omega\,.$$

To prove equality, observe that v (and also w) is a leaf in T_{aux}, otherwise either T_{aux} contains a hook or p was not a longest path. From Observation 3.23 it follows that even v^φ and w^φ are leaves. Consequently the augmented path q' constructed above is a longest path in T_{aux}. □

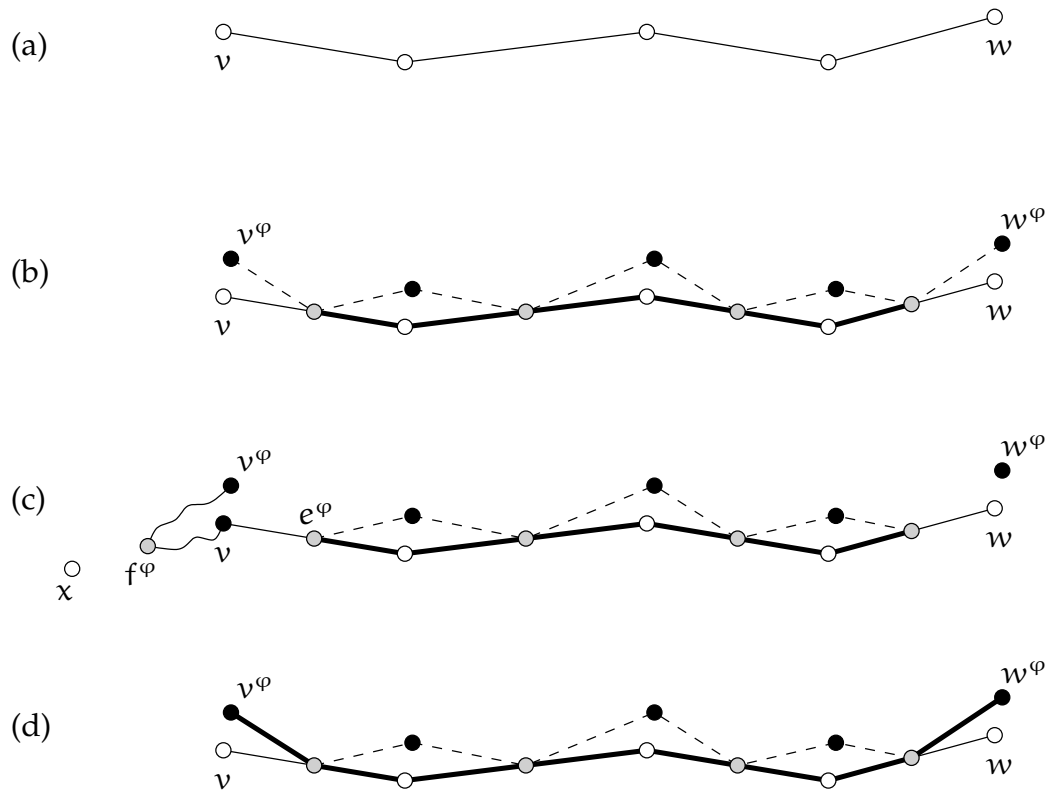

FIGURE 3.14: Illustration on the proof of Theorem 3.25. (a) initial path p in $\mathrm{prj}(T_{\mathrm{aux}})$. (b) path q in T_{aux} (thick edges). (c) forbidden situation: the path in T_{aux} from v to v^{φ} does not meet q. (d) augmented path q' (thick edges).

3.5.3 Transformation of the Solution to Original Graph

Let T_1 be the spanning tree of the auxiliary graph as computed by the algorithm of Hassin and Tamir [HT95]. In order to apply Theorem 3.25, we show that T_1 can be transformed into a tree T_2 which obeys the prerequisites of Theorem 3.25 without increasing the diameter.

Lemma 3.26
Let H be the auxiliary graph constructed from a graph $G = (V, E)$ with $|V| \geq 2$. Let T_1 be a spanning tree in H. Then there is a tree T_2 in H spanning all nodes from node set V^φ with

$$\operatorname{diam}^l(T_2) \leq \operatorname{diam}^l(T_1)\,,$$

such that T_2 has no dangling edges and no hooks.

Proof. We give a constructive proof. Algorithm 3.15 on the facing page performs the transformation. The algorithm consists of a loop where in each iteration exactly one of six possible modifications is executed. We show that in each iteration the diameter of the tree does not increase. All edge swaps operate on a local region which consists of a corresponding pair $\{v, v^\varphi\}$ and the three neighbor nodes from set E^φ. In order to verify that the diameter does not increase it suffices to check the pairwise distance of the three nodes from E^φ and—if there are new leaves—additionally the distance from each new leaf to each of those nodes. See Figure 3.16 on page 70 for an illustration.

1. The first case is easy. Removal of a leaf (Line 3) cannot increase the diameter. Moreover, the tree spans V^φ if this was true before the operation.
2. Case 2 applies to nodes in V^φ of degree 3. Since the tree spans $V \cup V^\varphi$ and does not contain a cycle, it follows that exactly two of the incident edges are dangling. The edge swap (Line 5) does neither increase the diameter nor affect the connectivity.
3. Case 3 applies to nodes in V^φ of degree 2 where not both incident edges are dangling. There are only the two cases possible which are depicted in Figure 3.16. The operation in Line 8 does not increase the diameter or change the connectivity.
4. We have two different situations to consider in case 4. The situation depicted in the left hand side of the figure is easy: Since the whole tree is connected and $|V| \geq 2$, there must be another Ω-edge in the subtree hanging from e^φ. Hence the new inserted edge $\psi := (v, e^\varphi)$ does not dominate the diameter.
   The situation depicted at the right hand side is more difficult. If the subtree hanging from e^φ contains an Ω-edge, the arguments from above can

Input: A spanning tree T_1 in the auxiliary graph

```
1  repeat
2    if there is a leaf e^φ ∈ E^φ in the tree then                    { case 1 }
3      Remove e^φ from the tree
4    else if there is a node v^φ ∈ V^φ with degree d(v^φ) = 3 then    { case 2 }
5      Replace the two incident dangling edges by their projections
6    else if there is a node v^φ ∈ V^φ with degree d(v^φ) = 2 then    { case 3 }
7      if only one incident edge is dangling then
8        Replace the incident dangling edge by its projection
9      end if
10   else if there is a dangling edge (v^φ, e^φ) such that v is a leaf then  { case 4 }
11     Replace the edge connecting v to the tree by edge (v, e^φ)
12   else if there is a dangling edge δ := (v^φ, e^φ) such that v is not a leaf then
13     Let ψ be an edge incident with v                               { case 5 }
14     Replace δ by Φ̂(ψ)
15   else if there is a hook e^φ then                                 { case 6 }
16     Let v^φ be the adjacent node from V^φ
17     Remove the hook and both incident edges
18     Connect v^φ by another non-dangling edge to the tree
19   end if
20 until no more changes have been made
```

Output: A tree T_2 spanning V^φ without dangling edges and without hooks

ALGORITHM 3.15: Algorithm constructing a tree without dangling edges and hooks.

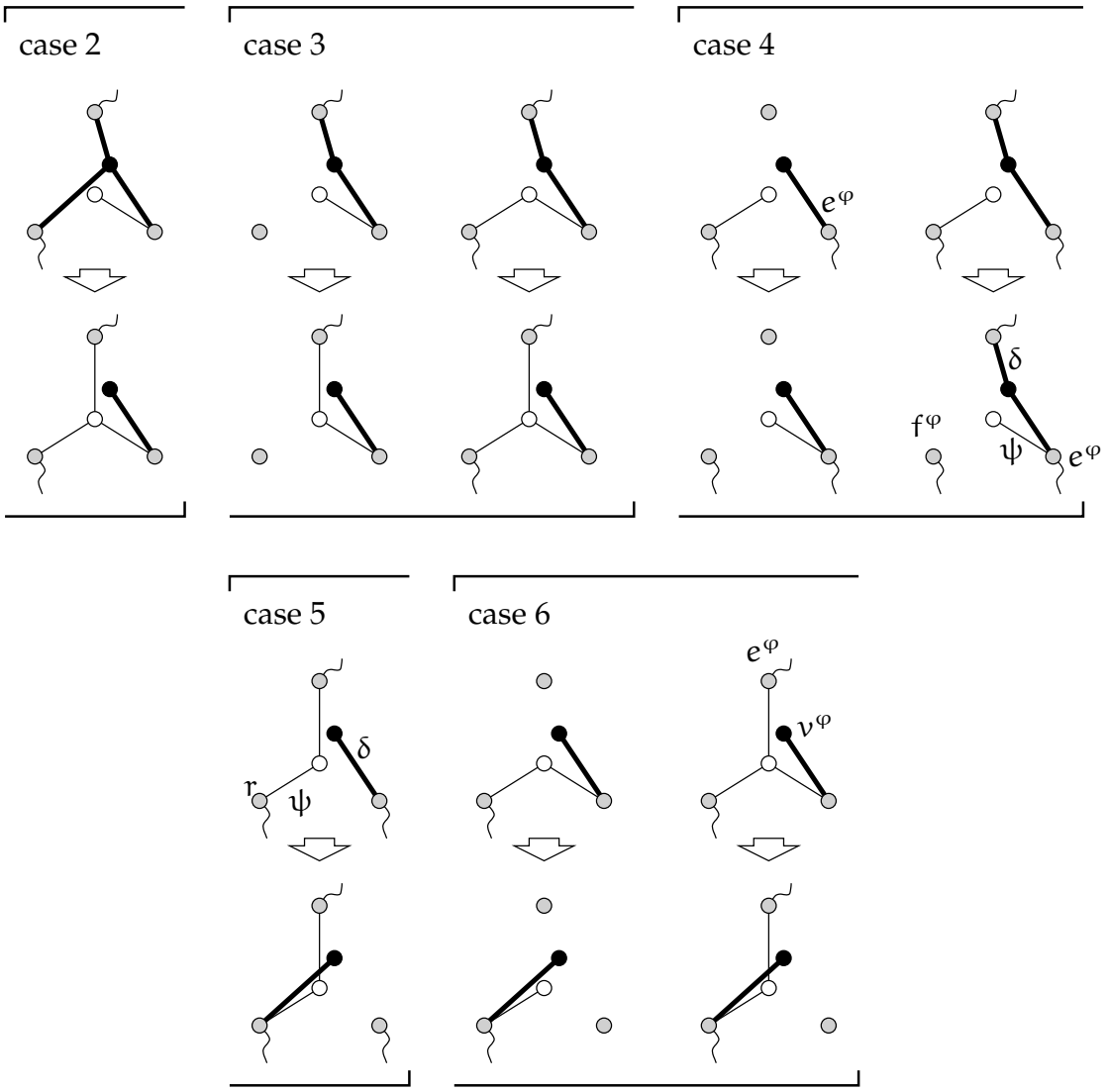

FIGURE 3.16: Illustration to the proof of Lemma 3.26. We display a node $v \in V$ (white), the corresponding node $v^\varphi \in V^\varphi$ (black), and the three adjacent nodes from E^φ (gray).

be applied. Otherwise, the new edge ψ may be longer than the subtree, and we cannot prove that the diameter does not increase.
Notice that in this situation the next operation is the removal of leaf f^φ. After that, case 3 applies, and the Ω-edge δ is replaced by its projection. As a result, the former longest path using two Ω-edges and the subtree is replaced by edges $\check{\Phi}(\delta)$, ψ, and the remaining Ω-edge. The weight reduction of the two operations is at least $\Omega - l(\check{\Phi}(\delta)) - l(\psi)$ which is nonnegative due to Equation (3.11) on page 62.

5. Notice that if this point is reached then all nodes from V^φ are leaves. Thus we have only one case to consider.
We prove now that the operation in Line 14 does not increase the diameter of the tree. Let r be the endpoint of edge ψ different from v. Assume that the current tree (before the operation) is rooted at r. Node r is not a leaf, otherwise it would have been removed in Line 3. Hence there are at least two subtrees hanging from r.
Each of the subtrees must contain at least one Ω-edge: otherwise all leaves of the subtree must be either from set E^φ (and would have been removed in case 1) or from set V (where case 4 would have been applied). Adding the Ω-edge $\hat{\Phi}(\psi)$ as a new subtree to r does not increase the diameter.
6. If this point is reached, there are no dangling edges. Hence the hook is exactly of degree 2 and we have only the two cases depicted in the figure. In the second case, the distance between v^φ and e^φ may increase, but arguments similar to the previous case show that the new edge does not dominate the diameter. □

To show the optimality of our algorithm, we need to compare the reload cost diameter of arbitrary trees with the solution produced by our algorithm. The missing chain link in the proof is contributed by the following lemma.

Lemma 3.27
Let T be an arbitrary spanning tree in G. There is a spanning tree T_{aux} in H with

$$\operatorname{diam}^l(T_{aux}) = \operatorname{diam}^c(T) + 2\Omega\,. \tag{3.13}$$

Proof. Start with the tree defined by $\operatorname{lift}(T)$. Connect each node from V^φ to the tree via an Ω-edge which is not dangling. This is possible since the tree spans V. The resulting tree satisfies the condition (3.13) due to Theorem 3.25. It remains to connect the nodes from node set E^φ which are not already spanned.

Consider one such node e^φ. Connect e^φ by edge (e^φ, v) to an arbitrary one of its neighbors $v \in V$. Since the sum of each pair of edges incident with v is dominated by Ω, the new added edges do not appear in longest paths. Therefore the diameter of the resulting tree T_{aux} does not increase. □

Input: Graph $G = (V, E, \chi, c)$ with reload costs obeying the triangle ineq., maximum node degree 3

1 Construct auxiliary graph H with edge lengths l
2 Compute a minimum diameter spanning tree T_1 in H with respect to edge lengths l
3 Call Algorithm 3.15 to compute tree T_2 out of T_1
4 Let $T := \mathrm{prj}(T_2)$

Output: tree T

ALGORITHM 3.17: Algorithm for problem M^{Δ}RDIAT on graphs with maximum degree 3.

We now summarize the results of this section.

Theorem 3.28 (Solution of M^{Δ}RDIAT)
Algorithm 3.17 solves M^{Δ}RDIAT *on graphs with degree bound* 3. *The running time is within* $O(|V|^2 \log |V|)$.

Proof. Let T be the final solution found by the algorithm. We prove optimality by showing

$$\mathrm{diam}^c(T) \leq \mathrm{diam}^c(T') \tag{3.14}$$

for arbitrary spanning trees T' of the graph.

Let T_1 and T_2 be the trees as computed by the algorithm. Notice that T_2 contains no dangling edges and no hooks. By Theorem 3.25 and Lemma 3.26 we have

$$\mathrm{diam}^c(T) = \mathrm{diam}^l(T_2) - 2\Omega \leq \mathrm{diam}^l(T_1) - 2\Omega\,. \tag{3.15}$$

Let T' be an arbitrary spanning tree of the graph. Apply Lemma 3.27 to T' in order to construct a spanning tree T_0 in H. By optimality of T_1 and Lemma 3.27, we can conclude

$$\mathrm{diam}^l(T_1) - 2\Omega \leq \mathrm{diam}^l(T_0) - 2\Omega = \mathrm{diam}^c(T')\,. \tag{3.16}$$

Hence (3.14) holds and the correctness of the algorithm is shown.

It remains to show the claim on the running time. Let $G = (V, E)$ be the input graph. From the degree constraint we have $|E| \leq 3/2 \cdot |V|$ (confer (3.6) on page 59). The auxiliary graph H has $2|V| + |E| \leq 7/2 \cdot |V|$ nodes and $4|E| \leq 6|V|$ edges.

From [HT95] it follows that the computation of a minimum diameter spanning tree in a graph with n nodes and m edges can be accomplished in time $O(mn + n^2 \log n)$. This yields a running time of $O(|V|^2 \log |V|)$ for the computation of the tree in the auxiliary graph. Algorithm 3.15 runs $O(|V|)$ iterations since it reduces the number of dangling edges or the number of edges by at least one in each iteration. The running time in each iteration does not exceed $O(|V| \log |V|)$ since only standard data structures are needed. Hence the total running time is within $O(|V|^2 \log |V|)$ as claimed. □

3.6 Hardness Results

In this section we show that MRDIAT and even M$^{\Delta}$RDIAT are NP-hard. Additionally we provide inapproximability results. We will show that the hardness results extend to graphs where the node degree is bounded by 5. This complements the results from before where a polynomial time algorithm was given which solves the problem on graphs with maximal degree 3. Some of the hardness results can easily be extended to problems MRRADT and M$^{\Delta}$RRADT.

3.6.1 General Reload Cost Functions

Theorem 3.29 (Hardness of MRDIAT)
Unless $P = NP$, MRDIAT *is not approximable within any factor $f(n)$ on a graph with n nodes. Here, f is any polynomial time computable function.*

Proof. We perform a reduction from 3-SATISFIABILITY [GJ79, Problem LO2]. An instance π of 3-SATISFIABILITY is given by a set $A = \{a_1, a_2, \ldots, a_n\}$ of variables and a set $C = \{C_1, \ldots, C_k\}$ of clauses over A. Each clause C_i is of the form $C_i = \{l_{i1}, l_{i2}, l_{i3}\}$, where l_{ij} is a literal, i.e., a variable a or its negation $\bar{a}$. The goal is to find a truth assignment satisfying all clauses.

We now construct an instance π' of MRDIAT. The construction of the graph $G = (V, E)$ is illustrated in Figure 3.18. Set $V := \{s\} \cup A \cup C$. For each variable $a \in A$, we introduce two colors $x_a, x_{\bar{a}}$ representing the positive and negative literal.

For each variable $a \in A$, insert two parallel edges between the nodes a and s of color x_a and $x_{\bar{a}}$, respectively. For each literal l of a clause $C_i \in C$, where l is the positive or negative variable a, add an edge between nodes C_i and a of color x_l to the graph.

Let the reload cost function c be given as

$$c(x_{l_1}, x_{l_2}) := \begin{cases} K, & \text{if } l_1 = \bar{l}_2, \\ 0, & \text{if } l_1 = l_2, \\ 1, & \text{otherwise,} \end{cases} \tag{3.17}$$

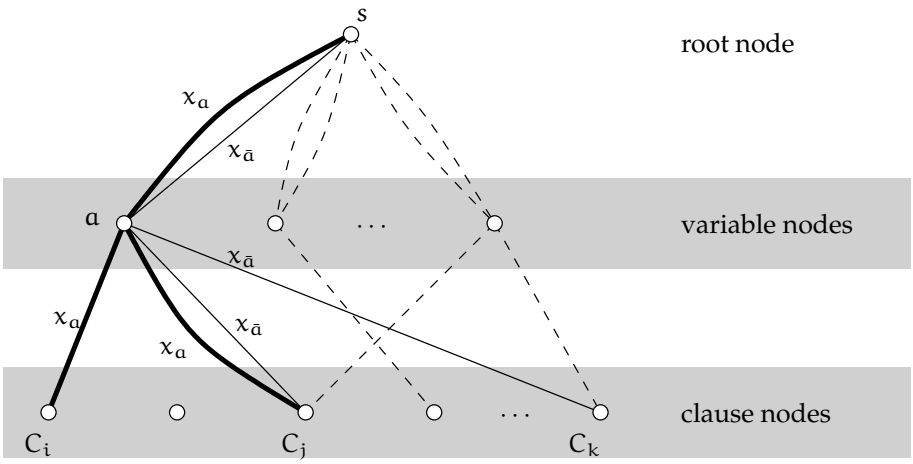

FIGURE 3.18: Reduction from 3-SATISFIABILITY to MINIMUM RELOAD COST DIAMETER SPANNING TREE used in the proof of Theorem 3.29.
We assume for the example that $C_i \ni a$, $C_j \ni a, \bar{a}$, and $C_k \ni \bar{a}$.

where $K > 1$ is some large constant. Informally, the reload costs are expensive if a pair of incident edges represents a variable and its negation, while they are low if the edges represent different variables.

Assume that there is a spanning tree T of G such that each clause node C_i is connected by a path of cost strictly less than K to the root s. Without increasing path lengths to the root, one can enforce that each of the clause nodes is a leaf. Therefore, the path from a clause node to the root uses exactly one variable node and is in fact of cost 0. Consequently, none of the auxiliary clause nodes is incident to edges of more than one color, and the diameter of the tree equals 1. Moreover, the colors of the edges adjacent to the variable nodes induce a valid assignment for π. Conversely, it is easy to see that a valid solution for π can be used to construct a spanning tree with diameter 1.

As a consequence, if π has a valid solution, then an optimal spanning tree has diameter 1. On the other hand, the diameter is at least $K + 1$ if π admits no valid assignment.

Assume that there is an approximation algorithm for MINIMUM RELOAD COST DIAMETER SPANNING TREE with performance $f(n)$. Choose $K > f(n)$. Then the algorithm must solve the instance π' exactly which is equivalent to solving 3-SATISFIABILITY by our observations. □

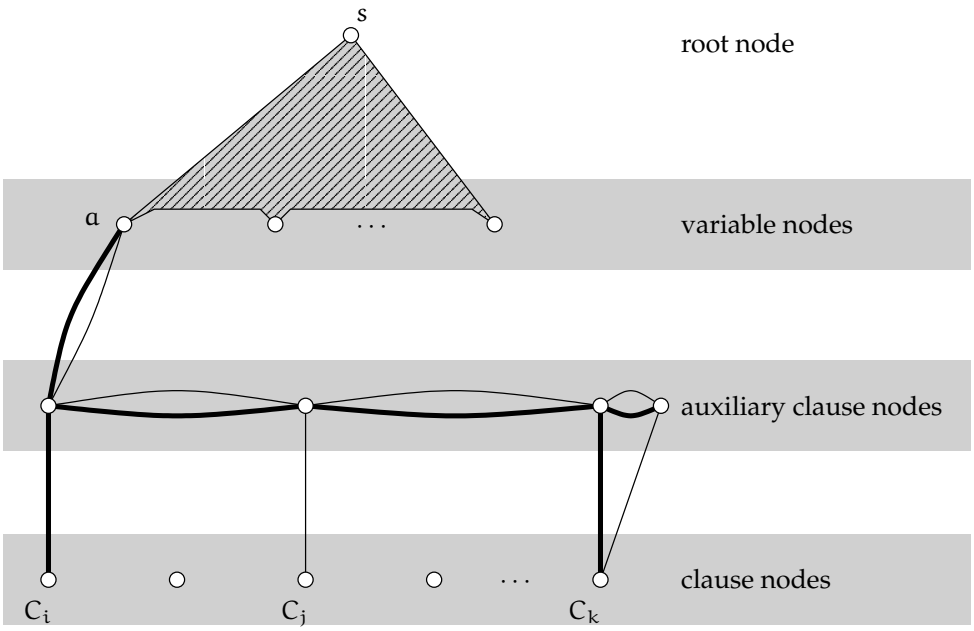

FIGURE 3.19: Modification of the reduction for degree 5 bounded graphs.

The construction in the proof of Theorem 3.29 can be slightly modified to guarantee that the constructed graph is of bounded degree 5 (confer Figure 3.19). There are only a few changes necessary:

Consider variable a. Instead of connecting the clause nodes directly to variable node a, we introduce auxiliary nodes which can be connected to node a by a subgraph of maximum degree 5. To avoid a root node of high degree, we choose a new color and replace the root by a suitable (with respect to the degree bound) tree of edges of the new color. Notice that all the modifications are acceptable for the reduction since they keep the graph in polynomial size.

Similar to the above proof, assume that there is a spanning tree T of G such that each clause node C_i is connected by a path of cost strictly less than $K + 1$ to the root s. Without increasing path lengths to the root, one can enforce that each of the clause nodes is a leaf. Therefore, the path from a clause node to the root uses exactly one variable node and is of cost 1. Consequently, none of the auxiliary clause nodes is incident to edges of more than one color. Hence the colors of the edges adjacent to the variable nodes induce a valid assignment for π. Conversely, it is easy to see that a valid solution for π can be used to construct a spanning tree with the property that each clause node is connected by a path of cost one to the root.

As a consequence, if π has a valid solution, then an optimal spanning tree has diameter 2. On the other hand, the diameter is at least $K + 2$ if π admits no valid assignment. Thus we can conclude:

Corollary 3.30 (Hardness of MRDIAT with bounded degree)
Unless $P = NP$, MRDIAT *is not approximable within any factor $f(n)$ on a graph with n nodes, even when restricted to graphs of maximum degree 5. Here, f is any polynomial time computable function.* □

It is easy to observe that the same construction can be used to show the hardness of problem MRRADT if the root of the instance is chosen to be the root s used in the construction. Therefore the same hardness results as before hold also for the reload cost radius problem.

Corollary 3.31 (Hardness of MRRADT with bounded degree)
Unless $P = NP$, MRRADT *is not approximable within any factor $f(n)$ on a graph with n nodes, even when restricted to graphs of maximum degree 5. Here, f is any polynomial time computable function.* □

3.6.2 Reload Cost Functions with Triangle Inequality

For the remaining part of this section we turn over to the problems where the reload cost functions obeys the triangle inequality as defined in Definition 3.3 on page 45. This case is much more interesting for practical applications than the general case without triangle inequality as pointed out in the introduction. Moreover, this metric case enabled us to design efficient algorithms for solving the problems on graphs with bounded degree.

A first inapproximability result for M^{Δ}RDIAT can directly be derived from Theorem 3.29 and Corollary 3.30. Consider the definition of reload cost function c in (3.17). In order to obey the triangle inequality, one must restrict K to satisfy

$$K \leq 2\,. \tag{3.18}$$

This immediately yields a lower bound of 2 for the approximability of problem M^{Δ}RDIAT. The same arguments can be applied to problem M^{Δ}RRADT. We state this fact in the following:

Corollary 3.32 (Hardness of M^{Δ}RRADT with bounded degree)
Unless $P = NP$, M^{Δ}RRADT *is not approximable within any factor $\alpha < 2$ on a graph with n nodes, even when restricted to graphs of maximum degree 5.* □

We will show now how the construction given above can be modified in order to slightly increase the lower bound for problem M^{Δ}RDIAT. The general idea behind the modification is to increase the gap of K in the reduction.

We start with the construction for the problem without degree constraints as given in the proof of Theorem 3.29. Setup two identical copies G_1, G_2 of the graph given above and connect the graphs by identifying the root nodes. See Figure 3.20 (left) for an illustration.

Now consider there is a spanning tree of diameter strictly less than $2K + 1$. Observe that the longest paths in the tree, which determine its diameter, visit the root node. The root node contributes reload cost 1 to these paths. Divide the tree at the root node so that the tree decomposes into two components: one component in G_1 and a second component in G_2. By an averaging argument it is evident that in at least one of the partial trees the maximal distance from any leaf to the root node is strictly less than K. This means that each clause node is connected to the root by a path of cost $< K$, hence the underlying instance of 3-SATISFIABILITY must have a solution.

Conversely, if the underlying instance of 3-SATISFIABILITY has a valid assignment, then in each copy G_i we can construct a spanning tree which connects each clause node to the root by a path of reload cost zero. Joining the two trees at the root node causes additional reload costs of 1. Hence the diameter of the spanning tree is equal to 1.

We have observed by now that the construction guarantees a gap between $2K + 1$ (where we can make no prediction on the existence of a solution) and 1 (where the 3-SATISFIABILITY instance is to be solved exactly). Recall that $K \leq 2$ from (3.18). This implies a lower bound of

$$\frac{2K + 1}{1} \leq 5$$

on the approximability of M^{Δ}RDIAT.

We set down this result in the following corollary.

Corollary 3.33 (Hardness of M^{Δ}RDIAT)
Unless $P = NP$, M^{Δ}RDIAT *is not approximable within any factor $\alpha < 5$.* □

The analysis is only slightly different for the case of the reduction where the degree of the graph is restricted. See Figure 3.20 (right) for an illustration.

By arguments similar to those given above one can show the following: If there is a spanning tree with diameter strictly lower than $2K + 2$, then one of the trees induces a solution to the 3-SATISFIABILITY instance. Conversely, if there

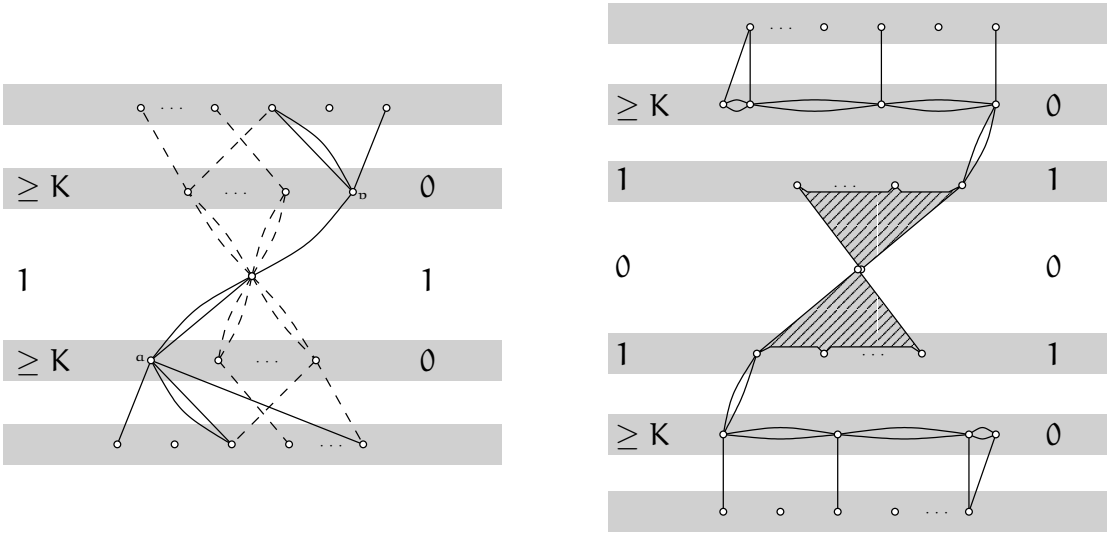

FIGURE 3.20: Construction of an approximability gap for the general case (left) and the degree constrained case (right). Numbers denote the cost for the case where the instance of 3-SATISFIABILITY is not solvable (left to the graph) or is solvable (right to the graph).

is a solution to 3-SATISFIABILITY, then one can construct a tree with diameter 2. From these observation it follows that

$$\frac{2K+2}{2} \leq 3$$

is a lower bound on the approximability of M^{Δ}RDIAT with degree constraint.

We summarize these results in the following corollary.

Corollary 3.34 (Hardness of M^{Δ}RDIAT with bounded degree)
Unless $P = NP$, M^{Δ}RDIAT is not approximable within any factor $\alpha < 3$ when restricted to instances with maximum node degree 5. □

In the remaining part of this section we will use an alternative reduction to strengthen the result of Corollary 3.33. However, this result holds on general graphs only, and we can not use it to improve the lower bound on the approximability for degree bounded graphs.

Theorem 3.35 (Hardness of M^{Δ}RDIAT)
Unless NP $\subseteq$ DTIME($N^{O(\log\log N)}$), M^{Δ}RDIAT is not approximable within any factor $\alpha < 1/4 \cdot \ln n$ on a graph with n nodes.

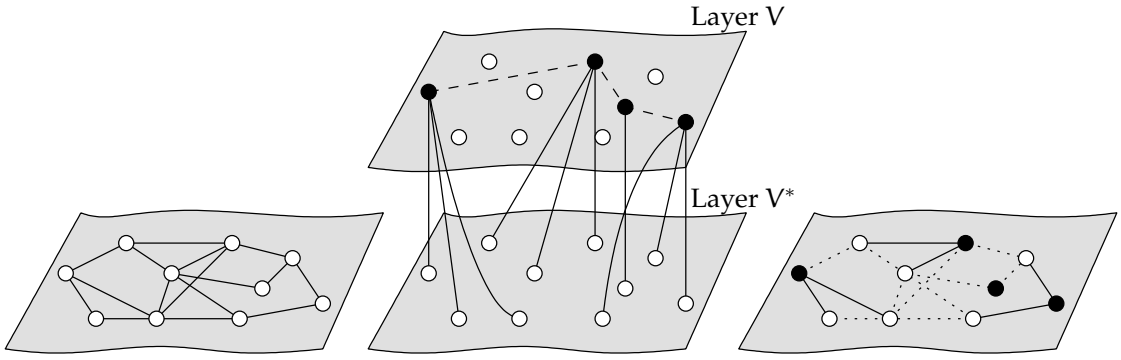

FIGURE 3.21: Illustration of the main idea of the proof of Theorem 3.35: Example of a graph (left). Constructed instance (center) with spanning tree consisting of stars (solid edges) and joining path (dashed edges). Resulting dominating set (right, black nodes).

Proof. We use a reduction from MINIMUM DOMINATING SET [GJ79, Problem GT 2]. Given a graph $G = (V, E)$, a node set $D \subseteq V$ is called *dominating*, if each node $v \in V \setminus D$ is adjacent to a node from D. A *minimum dominating set* is a dominating set of minimum cardinality. It is known that for any $\varepsilon > 0$ it is impossible to approximate MINIMUM DOMINATING SET within a factor $(1-\varepsilon)\ln|V|$ unless $NP \subseteq DTIME(N^{O(\log\log N)})$ (confer Theorem 1.8 on page 17).

Let $G' = (V', E')$ be an instance of MINIMUM DOMINATING SET, $V' = \{v'_1, \ldots, v'_n\}$. In the following we will construct an instance of M^{Δ}RDIAT from G' where the spanning tree is related to a dominating set and the reload cost diameter to the cardinality of this set.

We first outline the main idea behind the construction. See Figure 3.21 for an illustration. Start with a graph consisting of two layers, V and V^*, each being a copy of node set V', i. e., $V := \{v_i \mid v'_i \in V'\}$ and $V^* := \{v^*_i \mid v'_i \in V'\}$. Make layer V to be a complete graph. Then insert edges (v_i, v^*_j) between the layers if and only if v'_j is adjacent to v'_i in G'. These edges form stars with centers in layer V and leaves in V^*.

Each of the stars is assigned a unique color. Observe that stars of different colors touch each other in layer V. Assume that one can assure that the reload costs between stars of different colors are large. Then a spanning tree of minimum reload cost diameter must prefer nodes of V^* to be leaves. So it appears that this tree is in fact a collection of stars joined by some edges in layer V. Moreover, the set of centers of those stars form a dominating set in the original graph.

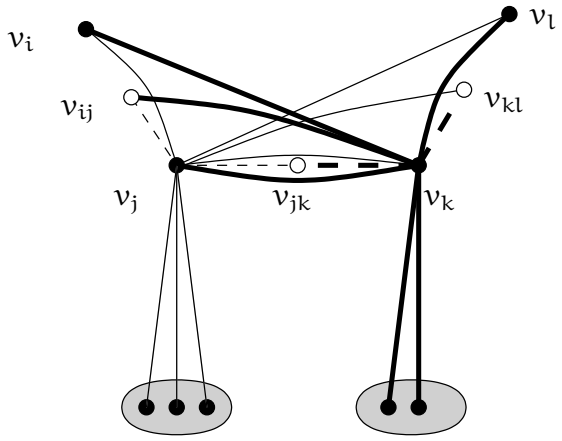

FIGURE 3.22: Graph used in the reduction of Theorem 3.35. The upper layer shows nodes from V in black and nodes from $V^\times$ in white. The display is restricted to two stars S^{ik}_j (thin lines) and S^{jl}_k (thick lines). Path colors are shown by dashed lines.

Assume for now that we can guarantee that the edges which join the stars in layer V form in fact a simple *path*. Then it is easy to see that the diameter of the tree is related to the number of joining edges which is again related to the cardinality of the dominating set. We now describe how this path property can be enforced.

For each edge (v_i, v_j) in layer V place a node v_{ij} in the center of that edge. For each node $v_k \in V$, replace the star around v_k by multiple copies, one copy for each pair v_i, v_j. Assure that reload costs between stars with identical centers are large, while costs equal 1 at those nodes which are placed in the middle of edges. This construction enforces that star centers are incident to edges of one color only, and hence the joining edges described above form a path in layer V.

We will now present a formal description of the details. Given $G' = (V', E')$, construct a graph G with node set $V \cup V^\times \cup V^*$, where

$$\begin{aligned} V &:= \{ v_i \mid v'_i \in V' \}, \\ V^\times &:= \{ v_{ij} \mid v_i, v_j \in V, i \neq j \}, \quad \text{with } v_{ij} \text{ and } v_{ji} \text{ identified} \\ V^* &:= \{ v^*_i \mid v'_i \in V' \}. \end{aligned}$$

See Figure 3.22 for an illustration of the construction. For any pairwise disjoint

$1 \leq i, j, k \leq n$, add a star S_j^{ik} with center v_j and fingers

$$\{ v^* \mid v' \in N(v'_j) \} \cup (V \setminus \{v_j\}) \cup (V^\times \setminus \{v_{ij}, v_{jk}\})$$

of color c_j^{ik}. Again, identify colors c_j^{ik} and c_j^{ki}, and remove parallel edges of the same color. Now complete star S_j^{ik} by fingers v_{ij} and v_{jk} of color $c_j'^{ik}$. Call a color of that type a *path-color*.

Finally, for $1 \leq i, j, k, l \leq n$, pairwise disjoint, $i = l$ allowed, set the reload costs

$$c(c_j^{ik}, c_j'^{ik}) := 1,$$
$$c(c_j'^{ik}, c_k'^{jl}) := 1,$$

otherwise set the reload costs to some large constant $\Omega > n + 1$.

Notice that the resulting graph is of size polynomial in the size of G'. In fact, if $n' = |V'|$ is the number of nodes of G', then the constructed graph consists of

$$n = 1/2 \cdot (n'^2 + 3n') \leq n'^2 \tag{3.19}$$

nodes (the inequation holds for all $n' \geq 3$).

We claim that the triangle inequality is satisfied. Consider a node from V^*. Since there are no path-colored edges incident, for all pairs of incident edges the reload costs are equal to Ω and the triangle inequality is satisfied. Consider a node $v_{jk} \in V^\times$. The set of incident colors contains a set of path colors with pairwise reload cost 1. All remaining costs equal Ω. Therefore the triangle inequality holds even in this case. Finally, consider a node $v_j \in V$. There are no three (pairwise different) incident colors such that two of the pairs have reload cost 1 each. Hence the triangle inequality is satisfied even in this remaining case.

We remark that the graph induced by node set $V \cup V^\times$ is independent on the structure of G'. Observe that there is always a spanning tree with diameter at most $n+1$: Choose a path-colored path in the graph induced by $V \cup V^\times$ such that the two edges adjacent to a node $v_j \in V$ are of the same color, say $c_j'^{ik}$. Connect the remaining nodes from V^* to the tree by edges of the star S_j^{ik}. Consequently, in any optimum solution to M^ΔRDIAT there is no node at which reload costs Ω arise.

Consider a spanning tree with no node where reload cost Ω appear. This tree must consist of a path-colored path in the graph induced by $V \cup V^\times$, supplemented by some stars with fingers in $V \cup V^\times \cup V^*$. If K, $K \subseteq V$, is the set of nodes from V on the path, then the diameter of the tree is at most $|K| + 1$.

Moreover, the corresponding set K' is a dominating set in G'. Conversely, it is easy to construct a spanning tree of diameter at most $|K'| + 1$ in graph G out of a dominating set K' in graph G'.

Let OPT be the diameter of an optimum solution in G, then the optimal dominating set in G' has size $OPT' = OPT - 1$. Assume that there is an approximation algorithm for M^{Δ}RDIAT with performance $c \cdot (1 - \varepsilon) \ln n$. Then from the approximate solution on G we can construct a dominating set in G'. Using (3.19) we can conclude that the size of the dominating set in G' is at most

$$\begin{aligned} c \cdot (1 - \varepsilon) \ln n \cdot OPT - 1 &\leq 2c \cdot (1 - \varepsilon) \ln n \cdot (OPT - 1) \\ &\leq 4c \cdot (1 - \varepsilon) \ln n' \cdot OPT' \end{aligned}$$

If $c < 1/4$ this would imply a contradiction to Theorem 1.8 on page 17. □

3.7 Concluding Remarks

To our knowledge, a cost structure similar to reload costs has not been considered in the literature so far. This cost structure is related to node weighted graphs, but the new aspect is that the cost at a node depend on the edges used by the walk through that node.

Table 3.23 shows a summary of results presented in this chapter. The notion "approximability" refers to the existence of a polynomial time approximation algorithm with the specified performance, assuming that the inclusion $NP \subseteq DTIME(N^{O(\log \log N)})$ is not true. As usual, V denotes the set of nodes of the graph. Function f is any function which can be evaluated in polynomial time. The complexity of the problem for graphs with degree bound 4 has not yet been determined.

As noted in Section 3.1.4 on page 48, the complexity of the problem does not change if reload costs are combined with classical edge lengths.

Problem	Degree bound	Lower bound on approximability	Reference
M^{Δ}RPATH	none	solvable in $O(\|V\|^3 + \|E\|)$	Corollary 3.11
MRRADT	5	any $f(\|V\|)$	Corollary 3.31
M^{Δ}RRADT	5	2	Corollary 3.32
	3	solvable in $O(\|V\|)$	Theorem 3.18
MRDIAT	5	any $f(\|V\|)$	Corollary 3.30
M^{Δ}RDIAT	none	$1/4 \cdot \ln \|V\|$	Theorem 3.35
	5	3	Corollary 3.34
	3	solvable in $O(\|V\|^2 \log \|V\|)$	Theorem 3.28

TABLE 3.23: Summary of results presented in this chapter.

Chapter 4

Dial a Ride

We conclude the part on network design problems with a chapter on a family of problems motivated from the area of transportation problems. The scenario we have in mind is a discrete metric space where objects are to be transported between given sources and destinations. Applications include the routing of pick-up-and-delivery vehicles, the control of automatic storage systems and scheduling of elevators.

The transportation job is executed by a server moving between the points of the space. The server can carry one good at a time. Given a set of transportation requests, the goal is to find a shortest transportation schedule which serves all requests.

We use an edge weighted graph to represent the metric space. Depending on the underlying graph structure the complexity of the problem changes from polynomially solvable cases up to NP-hard cases.

The basic problem is extended to the case where precedence relations between the particular requests specify a partial order in which the requests have to be performed. Of particular interest is the restriction where precedence constraints only appear between jobs which start at the same node. This is motivated by problems with (first-in first-out) waiting queues such as cargo elevator systems where at each floor conveyor belts deliver the goods to be transported. Elevators also motivate the restriction of the problem to paths and a variant of the problem with additional start and stop penalties.

It turns out that the problems can equivalently be formulated as graph augmentation problems. In this case, an optimal transportation schedule correlates with an Eulerian graph of minimum weight. We will prove structural facts about Eulerian cycles in a graph that respects precedence constraints on the arcs and use the results for designing the algorithms.

4.1 Preliminaries and Problem Formulation

4.1.1 Basic Problem

In the DIAL A RIDE problem we are given a finite transportation network and a finite set of transportation jobs. Each job specifies the source and target location which are both part of the network. A server moves on the transportation network to process the transportation requests. All goods have the same weight, and the server has unit capacity, i. e., it can carry one object at most at a time. The problem DIAL A RIDE consists of finding a shortest transportation for the jobs. Additionally, the tour is requested to start and end at a designated start location.

We model the transportation network by an edge weighted undirected graph. The length of an edge corresponds to the time or cost the server needs while traversing that edge. Each job request is modeled by an arc from its source node to its target node. The length of the arc is adjusted to reflect the length of a shortest path in the network connecting its endpoints. A transportation schedule is *valid* if each of the requests is satisfied. A valid solution corresponds to a closed walk in the mixed graph which traverses each of the arcs. More formally, we define DIAL A RIDE as follows:

Definition 4.1 (DIAL A RIDE Problem)
The input for DIAL A RIDE *consists of a finite mixed graph* $G = (V, E, A)$*, an origin vertex* $o \in V$ *and a nonnegative weight function* $c\colon E \to \mathbb{R}_0^+$*. The weight function* c *is extended to arc set* A *by defining*

$$c(a) := \operatorname{dist}_E(v, w) \quad \text{for all } a \in A,\ a = (v, w)\,,$$

i. e., the length of each arc equals the length of a shortest path along $G[E]$ *between the arc's endpoints. The goal of* DIAL A RIDE *is to find a closed walk in* G *of minimum cost which starts (and ends) in origin* o *and traverses each arc in* A*.*

It turns out that for the purpose of stating the algorithms in a more convenient way, it is helpful to use an equivalent formulation of DIAL A RIDE as a *graph augmentation* problem (cf. [AK88]). This formulation uses another view at the problem: A feasible solution can be constructed starting with the sole arc set. Then it is augmented using a multi-subset of the edges such that the resulting graph admits a closed walk. At this point the resulting graph must be Eulerian.

To this end consider the arc set $E^{\rightleftarrows}$ defined in (1.1) on page 3. Let the cost of each arc in $E^{\rightleftarrows}$ equal the cost of the corresponding edge in E.

Definition 4.2 (Graph augmentation version of DIAL A RIDE)
Given a mixed graph $G = (V, E, A)$*, origin* $o \in V$*, and cost function* $c\colon E \to \mathbb{R}_0^+$*, extend* c *to* $A + E^{\rightleftarrows}$ *as described in Definition 4.1 and the previous paragraph. The goal is to find a multi-set* R*,* $R \sqsubset E^{\rightleftarrows}$*, of minimum cost such that graph* $G[A + R]$ *is Eulerian and contains* o*.*

We argue that this is an equivalent formulation of DIAL A RIDE. In the sequel, we will also use the shorter notation DARP for the problem.

Observation 4.3
Definition 4.1 and Definition 4.2 are equivalent formulations of the same problem.

Proof. Let W be a feasible solution for DARP as stated in Definition 4.1, that is, a closed walk that starts in o and traverses each arc in A. Construct a multi-set $R \sqsubset E^{\rightleftarrows}$ of arcs in the following way: Traverse edges and arcs along W. For each time an undirected edge $e \in E$, $e = [u, v]$, is traversed from u to v, add a copy of the directed arc (u, v) to multi-set R. Then graph $G[A + R]$ contains o, and since W defines a cycle, graph $G[A + R]$ must be Eulerian.

Conversely, let $R \sqsubset E^{\rightleftarrows}$ be a multi-set of arcs such that $G[A + R]$ is Eulerian and includes the origin o. Construct a walk W as follows: Traverse an Eulerian cycle C in $G[A + R]$ starting in o. If the current arc r from C is in A then add r to walk W, otherwise add the undirected edge corresponding to r. By this construction, W is a closed walk in G traversing each arc from A and including o. In both cases we have $c(W) = c(A + R)$, i. e., the cost of walk W equals the cost of multi-set R plus cost of arc set A. □

It turns out that the complexity of the problem family depends highly on the structure of the graph $G[E]$ induced by the edge set. We use the notion DARP *on paths*, *on trees*, *on general graphs* to denote the fact that the underlying graph $G[E]$ is restricted to the respective graph class.

Throughout the chapter we will use $S^* \sqsubset E^{\rightleftarrows}$ to denote an optimal solution of the investigated problem, i. e., an augmentation set. By

$$\mathrm{OPT} := c(A + S^*)$$

we denote the cost of an optimal solution. Recall that by $m_E := |E|$ and $m_A := |A|$ we denote the cardinalities of the edges and arcs of the input graph $G = (V, E, A)$.

4.1.2 Precedence Constraints

In real applications of DIAL A RIDE there are often additional constraints on the order of the execution of transportation requests. This can be modeled by introducing a partial order $\prec$ on the set of arcs. For arbitrary arcs $a, b \in A$, the relation $a \prec b$ means that arc a must be traversed before arc b by a feasible walk.

In some transportation networks there is a "waiting pool" at each node where transportation requests originate. Each of the pools constrains the order of execution of requests starting from this node while requests starting from other nodes are not affected. For instance there might be waiting pools with first-in first-out queues or waiting stacks (last-in first-out). This motivates the definition of a *source order*, which is a partial order where only arcs originating at the same node are compared to each other.

Definition 4.4 (Source Order, Total Source Order)
*A partial order $\prec$ on the set of arcs is called **source order**, if*

$$a \prec b \implies a \text{ and } b \text{ share the same source node}.$$

Moreover, if for a source order $\prec$,

$$a \text{ and } b \text{ share the same source node} \implies a \prec b \vee b \prec a,$$

*then the source order is called **total**.*

This definition leads to SOURCE ORDER DIAL A RIDE problem (SOURCE-DARP for short) which is the main focus of this chapter.

Definition 4.5 (SOURCE ORDER DIAL A RIDE Problem)
An instance of SOURCE-DARP *consists of an instance of* DIAL A RIDE*, together with a source order $\prec$ on the arc set A. The goal is to find a closed walk satisfying the requirements specified in Definition 4.1 and the precedence constraint given by $\prec$.*

Problem SOURCE-DARP is indeed different from DIAL A RIDE. Figure 4.1 on the next page shows an example where an optimal source order respecting transportation schedule starting at node o_1 is strictly longer than an optimal schedule neglecting the precedences: If no constraints have to be obeyed, then the jobs can be served traversing along the arcs without using an edge. If the constraint $b \prec a$ must be obeyed then two additional empty moves along undirected edges are necessary. (Notice that this does not happen if we choose o_2 to be the start point of the tour.)

For the sake of presentation it will be useful to formulate SOURCE-DARP as a graph augmentation problem. To this end it is necessary to introduce the notion of an Eulerian cycle which respects the precedence constraints.

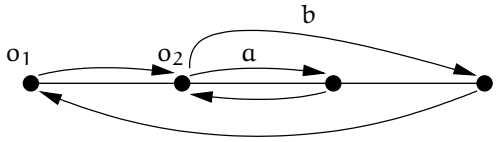

FIGURE 4.1: Precedence constraint $b \prec a$ increases the cost.

Definition 4.6 ($\prec$-respecting Eulerian Cycle, $\prec$-Eulerian)
*Let $H = (V, R)$ be a directed graph, $\prec$ be a source order on the arc set R, and $o \in V$. A **$\prec$-respecting Eulerian cycle** in H **with start** o is a Eulerian cycle C in G such that $a \prec b$ implies that in the walk from o along C the arc a appears before b. The graph H is then called **$\prec$-Eulerian with start** o.*

We emphasize that—in contrast to the case of classical Eulerian cycles—for $\prec$-respecting Eulerian cycles it is meaningful to specify a start node explicitly. Consider the example depicted in Figure 4.1, restricted to the set of arcs. Given precedence constraint $b \prec a$, it is easy to see that there is a $\prec$-respecting Eulerian cycle starting in o_2, but there is no feasible solution with start point o_1.

Definition 4.7 (Graph Augmentation Version of SOURCE-DARP)
An instance of the problem SOURCE-DARP *consists of the same input as for* DARP *and additionally a source order $\prec$ on the arc set A. The goal is to find a multi-set S of arcs from $E^{\rightleftarrows}$ minimizing the weight $c(A + S)$ such that the graph $G[A + S]$ is $\prec$-Eulerian with start o, and to determine a $\prec$-respecting Eulerian cycle in $G[A + S]$.*

4.1.3 Basic Observations

We continue with some technical assumptions about input instances of problem SOURCE-DARP. While all these assumptions are without loss of generality they greatly simplify the presentation of our algorithms. Recall that an instance is given in the form $(G = (V, E, A), c, o, \prec)$.

Definition 4.8 (Essential Node)
*A node is called **essential** if it is incident with an arc from set A.*

Consider the problem on trees. We argue that we can assume that most of the nodes are essential. Let $v \neq o$ be a node of degree two which is not essential. Let $[u, v]$ and $[v, w]$ be the two incident edges. Obviously one can replace that edges by the single edge $[u, w]$ of cost equal to the sum of the two edges and

afterwards remove v without affecting the length of an optimal solution. If $v \neq o$ is a non-essential leaf, it can be removed anyway (cf. [FG93] for DARP on trees).

Assumption 4.9 (Technical assumption for SOURCE-DARP on trees)
Each vertex of degree one or two in $G[E]$*, except for the origin, is essential.*

If $G[E]$ is a path there are no nodes with degree three of higher in $G[E]$. This is the reason why we can make an even stronger assumption without loss of generality (cf. [AK88] for DARP on paths):

Assumption 4.10 (Technical assumption for SOURCE-DARP on paths)
Each vertex is essential.

We now turn to SOURCE-DARP on general graphs.

Assumption 4.11 (Tech. assumption for SOURCE-DARP on general graphs)
1. *Each vertex is essential.*
2. $G[E]$ *is complete.*
3. *The cost function* c *obeys the triangle inequality, i. e., for any edge* $e \in E$*,* $e = [u, v]$*, the cost* $c(e)$ *equals* $\mathrm{dist}_E(u, v)$.

Assumption 4.11 can be enforced without increasing the value of an optimal solution. If the start vertex o is not essential, insert a new vertex o' joined by a dummy arc (o, o') and edge $[o, o']$ to the start vertex, each of cost zero. To satisfy the triangle inequality, for every pair u, v of nodes add a new edge $[u, v]$ of cost equal to the shortest path in $G[E]$ between u and v. Afterwards each bundle of parallel edges can be replaced by retaining the cheapest edge of the bundle, and nodes which are not incident to an arc can be removed safely (cf. [FHK78] for DARP.

However, Assumption 4.11 can not be made without loss of generality for SOURCE-DARP on trees, since the suggested modification of the graph destroys the "tree-property".

4.2 Related Work

The problem DARP is also known as the STACKER CRANE problem. In [FG93] it is shown that the problem is NP-hard even on trees. In [FHK78] the authors present a 9/5-approximation algorithm for the problem on general graphs. An improved algorithm for trees with performance 5/4 is given in [FG93]. On paths DIAL A RIDE can be solved in polynomial time [AK88].

The extension of DIAL A RIDE where a vehicle of capacity $C > 1$ is used to serve the transportation jobs has been addressed in [Gua98, CR98]. For capacity $C > 1$ the problem becomes NP-hard even on paths. In [CR98] an approximation algorithm for the single server dial-a-ride problem with performance $O(\sqrt{C} \log n \log \log n)$ has been given, where C denotes the capacity of the server and n denotes the number of nodes in the graph.

Precedence constraints have been studied in the case of chinese postman tours [DST87]. The CHINESE POSTMAN problem consists of finding a shortest walk in a graph that traverses all edges and arcs. The authors show that for general precedence relations it is NP-hard to determine a chinese postman tour of minimum length. Under strong restrictions on the precedence relation the problem can be solved in time $O(n^5)$, where n denotes the number of nodes in the input graph.

Online variants of DIAL A RIDE have been investigated in [AKR98, AKR00, FS01]. All of the known competitive algorithms have to solve offline instances of DIAL A RIDE during their run. The performance of the employed offline algorithm directly affects the competitive ratio of the online algorithm. If a ρ-approximation algorithm for DIAL A RIDE is given, there is a polynomial time algorithm for ONLINE-DIAL A RIDE with competitive ratio $\rho+1/4+1/4\cdot\sqrt{1+8\rho}$ [AKR00].

4.3 Balancing

The formulation of SOURCE ORDER DIAL A RIDE as a graph augmentation problem (see Definition 4.7 on page 89) suggests to investigate the structure of Eulerian graphs. Confer [FG93] for the following observations.

A necessary condition for a graph to be Eulerian is that for each node its in-degree equals its out-degree. It turns out to be helpful for solving SOURCE-DARP to search for augmenting sets which guarantee the resulting graph to be *balanced* in that way.

Definition 4.12 (Balancing set)
*Let $G = (V, E, A)$ be a mixed graph. A multi-set $B \sqsubset E^{\rightleftarrows}$ of arcs is called a **balancing set** if*

$$d^+_{A+B}(v) = d^-_{A+B}(v)$$

for all nodes $v \in V$.

Suppose that $G[E]$ is a tree and that Assumption 4.9 is satisfied. We now try to identify a multi-set of edges which is necessarily contained in every optimal solution of SOURCE ORDER DIAL A RIDE.

Consider an arbitrary edge $e \in E$, $e = [x, y]$. This edge implies in a natural way a partition $V = X \cup Y$ of the node set: X and Y are defined as the two connected components which result from a removal of edge e. Now define the cut $\delta(X)$ to consist of all edges and arcs from $E + A$ with one endpoint in X and the other one in Y, formally

$$\begin{aligned}\delta(X) := &\{ [v, w] \in E \mid v \in X \wedge w \notin X \} \\ &\cup \{ (v, w), (w, v) \in A \mid v \in X \wedge w \notin X \}\end{aligned}$$

Observe that for a partition $V = X \cup Y$, we have $\delta(X) = \delta(Y)$ by definition.

Obviously, the cut $\delta(X)$ must be traversed by any closed walk W the same number of times in each direction. If the walk is feasible, then some of the traversals are predefined by the set of arcs in the cut. To achieve equal numbers of traversals forward and backward, the remaining traversals must use the underlying edge.

We formalize the above observation. For an arbitrary cut $\delta(X)$ induced by an edge e of the tree $G[E]$, denote by

$$\begin{aligned}\varphi^+(X) &:= \left|\{ (x, y) \in A \mid x \in X \wedge y \notin X \}\right| \\ \varphi^-(X) &:= \left|\{ (y, x) \in A \mid y \notin X \wedge x \in X \}\right|\end{aligned}$$

the number of arcs emanating from (respectively, entering) X. Notice that $\varphi^+(X) = \varphi^-(Y)$ for a partition $V = X \cup Y$. Then a feasible walk traverses edge $[x, y]$ in direction from x to y at least $\varphi^-(X) - \varphi^+(X)$ times. This is an empty proposition if this number is non-positive. However, if $\varphi^-(X) = \varphi^+(X) = 0$, then edge $[x, y]$ must be traversed at least once: This is true due to the fact that the walk must traverse both components of the cut, which is an immediate consequence of Assumption 4.9. Hence, edge $[x, y]$ must be traversed in direction from x to y at least $b(x, y)$ times, where

$$b(x, y) := \begin{cases} \varphi^-(X) - \varphi^+(X) & \text{if } \varphi^-(X) > \varphi^+(X) \\ 1 & \text{if } \varphi^+(X) = \varphi^-(X) = 0 \\ 0 & \text{otherwise.} \end{cases}$$

This observation has consequences for the graph augmentation version of problem SOURCE-DARP: Assume that we have a multi-set $B \sqsubset E^{\rightleftarrows}$ which contains exactly $b(x, y)$ copies of each directed arc (x, y). Then this solution does not contain dispensable edges, i. e., it can be augmented to build an optimal solution. This yields the following result:

Lemma 4.13 (Construction of an Optimal Balancing Set)
Let (G, c, o) *be an instance of* DARP *such that* $G[E]$ *is a tree. Then one can find in time* $O(nm_A)$ *a balancing set* $B \sqsubset E^{\rightleftarrows}$ *such that* $B \subseteq S^*$ *for some optimal solution* S^**. If* $G[E]$ *is a path, the procedure can be carried out in time* $O(n + m_A)$. □

The lemma is proven in [AK88, FG93]. For trees one must take some care to compute function b efficiently. For paths, it is sufficient to employ a sweep-line approach to compute $b(e)$ for each edge $e \in E$. Notice that the solutions do not actually *create* the balancing arcs (this would be too time consuming) but instead provide a counter for each bundle of parallel arcs. As is also shown in [AK88, FG93], even for trees the time bound of $O(nm_A)$ can be improved to $O(n + m_A)$ by allowing balancing arcs to be from $V \times V$ instead of just $E^{\rightleftarrows}$. This does not change the problem: the cost function c can basically be extended from $E^{\rightleftarrows}$ to $V \times V$ by the length of shortest paths.

Notice that Lemma 4.13 remains valid even in the presence of source orders.

4.4 Euler Cycles Respecting Source Orders

Assume that $\prec$ is a total source order. Then it is easy to decide whether a given graph H is $\prec$-Eulerian with start o: The $\prec$-respecting cycle (if it exists) is uniquely determined. It can be found by a walk through the graph where at each vertex v we always choose among the yet unused arcs from A_v an arc which is minimal with respect to $\prec$.

In the sequel we prove a necessary and sufficient condition on the graph for the analogous question for general source orders.

Consider an Eulerian cycle in a connected directed graph. The cycle visits each node, and by this it defines for each node an arc which is last used.

Definition 4.14 (Set of Last Arcs)
Let C *be an Eulerian cycle with start* o*. By* L_C *we denote the* ***set of last arcs*** *of* C*.* L_C *contains for each node* $v \in V$ *the unique arc emanating from* v *which is traversed last by* C*.*

Observation 4.15 (Characterization of a Set of Last Arcs)
Let C *be an Eulerian cycle with start* o*, let* L_C *be the set of last arcs. Then* L_C *consists of a spanning tree rooted towards* o*, extended by one single arc emanating from* o*.*

Proof. Denote by T the set L_C without the arc emanating from o. Since $|T| = |V| - 1$ it suffices to show that T contains for each $v \in V$ a path from v to the

origin o. Assume for the sake of a contradiction that this is not true. Since the out-degree of each node $v \neq o$ equals 1, the set T must contain a cycle t. Let $v \in t$ be the node from the cycle which is visited for the last time by C. Let (v, w) be the arc in t emanating from v. Since (v, w) is a last arc, node w is visited later than v by C. This contradicts that v was the last node from t visited by C. □

In the following we will explain the situation for source order respecting Eulerian cycles. We define maximal elements with respect to a source order in the usual way:

Definition 4.16 (Maximal Elements)
Let $\prec$ be a source order. Denote by

$$M_{\prec} := \{\, a \in A : \textit{there is no arc } b \in A \textit{ such that } a \prec b \,\}$$

*the **set of maximal elements** with respect to $\prec$.*

It will turn out that it is helpful to search for a set of arcs which is of the structure noted in Definition 4.14. Any such set can appear as a set of last arcs of an appropriate cycle. We will denote an arc set of that type as a *possible set of last arcs* in the sequel.

Definition 4.17 (Possible Set of Last Arcs)
*Let $H = (V, R)$ be a directed graph and $o \in V$ be a distinguished vertex. A set $L \subseteq R$ is called a **possible set of last arcs**, if it satisfies the following conditions:*

1. $d^+_L(v) = 1$ *for all* $v \in V$, *and*
2. *for each* $v \in V$ *there is a path from* v *to* o *in* $H[L]$.

We remark that this definition is equivalent to the following: a set L is a possible set of last arcs, if $H[L]$ is a directed spanning tree rooted towards o, plus one arbitrary arc emanating from o. This has been shown in Observation 4.15.

The use of a possible set of last arcs is justified by the following theorem. A possible set of last arcs can be used to construct a closed walk as described in the introduction to the current section. It is easy to see that for existence of a $\prec$-respecting solution it is a necessary condition that the set of last arcs does not contain arcs which are dominated by other arcs. The following theorem shows that this condition is already sufficient.

Theorem 4.18 (Construction of a $\prec$-respecting Eulerian cycle)
Let $H = (V, R)$ be a directed Eulerian graph with a distinguished vertex $o \in V$. Let $\prec$ be a source order with maximal elements $M_\prec$. Suppose that a possible set L of last arcs satisfies $L \subseteq M_\prec$.

Then there exists an $\prec$-respecting Eulerian cycle C with start o in H such that $L_C = L$, i. e., such that L is the set of last arcs of C. This cycle can be found in time $O(|V| + |R|)$.

Proof. Color the arcs from L red and the arcs in $R \setminus L$ blue. We claim that by the following procedure we construct an Eulerian cycle C in H with the desired properties.

Start with current vertex o. As long as there is a blue untraversed arc emanating from the current vertex, choose one which is not $\prec$-preceeded by any other untraversed arc, otherwise choose the red arc. Traverse the chosen arc, let its target be the new current vertex, and repeat the iteration. Stop, if there is no untraversed arc emanating from the current vertex.

Call the resulting path of traversed arcs C. Since H is Eulerian by assumption, for each vertex its in-degree equals its out-degree. Therefore, C must end in the origin o and forms in fact a cycle. Moreover, C is $\prec$-respecting by construction since $L \subseteq M_\prec$. Hence, if we can show that C traverses all arcs from R then this implies $L = L_C$ and the proof is complete.

To this end, define for each node $v \in V$, the value $\mathrm{dist}(v, o)$ to be the distance (i. e., the number of arcs) on the shortest path from v to o in the subgraph $H[L]$. We show by induction on $\mathrm{dist}(v, o)$ that all arcs emanating from v are contained in C.

If $\mathrm{dist}(v, o) = 0$ then $v = o$. Since our procedure stopped, all arcs emanating from o are contained in C. This proves the induction basis. Assume that the claim holds true for all nodes with distance $t \geq 0$ and let $v \in V$ with $\mathrm{dist}(v, o) = t+1$. Let $a = (v, w)$ be the unique red arc emanating from v. Then $\mathrm{dist}(w, o) = t$ and by the induction hypothesis all arcs emanating from w are contained in C. Since $d^+_H(w) = d^-_H(w)$, it follows that all arcs entering w are also contained in C, in particular arc a is. Since red arc a is chosen last by our procedure, all other arcs emanating from v must be contained in C. This completes the induction.

Hence, the path C is actually an Eulerian cycle with the claimed properties. The claim on the running time follows immediately from the construction described above. □

We summarize the results of this section in the following characterization of graphs which are $\prec$-Eulerian:

Theorem 4.19 (Characterization of $\prec$-Eulerian graphs)
Let $H = (V, R)$ be a graph, $o \in V$ and $\prec$ a source order. Then the following two statements are equivalent:

1. *H is $\prec$-Eulerian with start o.*
2. *H is Eulerian and the set $M_\prec$ of maximal elements with respect to $\prec$ contains a possible set of last arcs.*

Proof. Suppose that H is $\prec$-Eulerian with start o, and let C be an $\prec$-respecting Eulerian cycle with start o in H. Then $L_C \subseteq M_\prec$. Thus Statement 1 implies 2. The other direction is an immediate consequence of Theorem 4.18. □

The above Theorem 4.19 implies an efficient algorithm for deciding whether a given graph H is $\prec$-Eulerian with start o. Provided H is Eulerian it suffices to check whether the subgraph formed by the arcs from $M_\prec$ contains a possible set of last arcs. By the remark from above this task can be performed by testing whether $M_\prec$ contains a directed spanning tree rooted towards o (which can be done in linear time).

4.5 A Polynomial Time Algorithm for Source-Darp on Paths

We now present an algorithm which solves SOURCE-DARP on paths. The instance of the problem is given by a mixed graph $G = (V, E, A)$ such that $G[E]$ is a path, additionally a source order $\prec$ and a start $o \in V$. We assume throughout this section that Assumption 4.10 on page 90 holds. See Algorithm 4.2 for the details. An illustration of the computation can be found in Figure 4.3.

The algorithm starts in Step 2 by determining a balancing set $B \sqsubset E^{\rightleftarrows}$. Following the results of Lemma 4.13, it is guaranteed that B is contained in some optimal solution. At this point, the graph $G[A + B]$ is degree balanced but may consist of several connected components. (See Figure 4.3 (center) for an example.)

In order to turn the graph $\prec$-Eulerian with start o, the idea is to connect the components by pairs of antiparallel arcs from set $E^{\rightleftarrows}$ and in the same moment ensuring the existence of a possible set of last arcs.

This task is performed in two parts by the algorithm: In Step 3, a directed spanning tree rooted towards o of minimum cost is computed with respect to an auxiliary cost function. The cost function c' is adjusted to measure only the additional arcs which are not yet contained in $A + B$. In Step 5 an auxiliary arc set N is defined which contains those additional arcs together with their

Input: A mixed graph $G = (V, E, A)$, such that $G[E]$ is a path, a cost function c on E, a start node $o \in V$, and a source order $\prec$

1 Let $M_\prec \subseteq A$ be the set of maximal elements with respect to $\prec$.
{ *Notice that each node is essential.* }

2 Compute a balancing set $B \sqsubset E^{\rightleftarrows}$ such that $B \subseteq S^*$ for some optimal solution S^*.

3 Compute a directed spanning tree D rooted towards o in $G[B + M_\prec + E^{\rightleftarrows}]$ of minimum weight $c'(D)$, where cost function c' is defined as follows:

$$c'(r) = \begin{cases} 0 & \text{if } r \in B + M_\prec \ , \\ c(r) & \text{if } r \in E^{\rightleftarrows} \setminus (B + M_\prec) \, . \end{cases}$$

4 Define a possible set L of last arcs by adding an arbitrary arc from $A_o \cap (M_\prec + B)$ to tree D { *Such an arc must exist since* o *is essential and* $G[A + B]$ *is degree-balanced.* }

5 Let $D_+ := D - (B + M_\prec)$ and $N := E^{\rightleftarrows} \cap (D_+ \cup D_+^{-1})$.
{ *The set* N *contains the set* D_+ *of "new arcs" from the tree* D *and their inverses* D_+^{-1}. }

6 Use the method from Theorem 4.18 to find a $\prec$-respecting Eulerian cycle C with start o in $G[A + B + N]$ such that L is the set of last arcs of C.

Output: the multi-set $B + N$ and the cycle C

ALGORITHM 4.2: Algorithm for solving SOURCE-DARP on paths.

inverse arcs. This guarantees that $G[A + B + N]$ is in fact $\prec$-Eulerian. (See Figure 4.3 (bottom).)

Lemma 4.20
The set $B+N$ *returned by Algorithm 4.2 is a feasible solution for* SOURCE-DARP. *In other words,* $G[A + B + N]$ *is* $\prec$*-Eulerian with start* o.

Proof. Since $G[A + B]$ is degree balanced by construction and N consists of pairs of anti-parallel arcs, also $G[A + B + N]$ is degree balanced. On the other hand, graph $G[A + B + N]$ contains a directed spanning tree rooted towards o, namely the tree D computed in Step 3. Hence the graph is strongly connected and Eulerian.

The set L of arcs determined in Step 4 satisfies the conditions of Definition 4.17, hence it is a possible set of last arcs. The claim now follows from Theorem 4.18. □

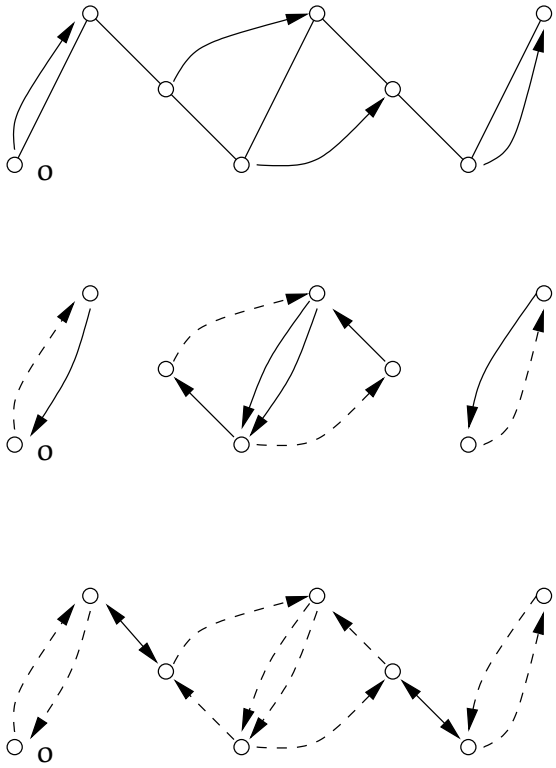

FIGURE 4.3: Steps of Algorithm 4.2. Initial instance (top), graph $G[A+B]$ after computing a balancing set (center), final solution (bottom).

It remains to show that the solution produced by Algorithm 4.2 is not only feasible but also of minimum cost.

Theorem 4.21 (Solution of SOURCE-DARP on paths)
Algorithm 4.2 finds an optimal solution for SOURCE-DARP *on paths. It needs running time* $O(n + m_A + \min\{(m_A + n)\log n, n^2\})$.

Proof. Let S^* be an optimal solution such that $B \subseteq S^*$ (by Lemma 4.13 such a multi-set S^* exists). By feasibility of S^* the graph $G[A + S^*]$ is $\prec$-Eulerian with start o.

Consider the multi-set Z

$$Z := (A + S^*) - (A + B) = S^* - B$$

which contains the arcs from the optimal solution S^* which are not contained in the current intermediate balanced graph $G[A + B]$. We show that the sum of the weight which the algorithm adds in the remaining steps is bounded by $c(Z)$. This shows that the solution is optimal.

Observe that both graphs $G[A + B]$ and $G[A + S^*] = G[A + B + Z]$ are degree balanced. Since $Z \cap (A + B) = \varnothing$, also $G[Z]$ must be degree balanced and we can decompose the set Z into arc disjoint cycles. From $Z \sqsubset E^{\rightleftarrows}$ and the property that $G[E]$ is a path it follows that

$$r \in Z \implies r^{-1} \in Z \quad \text{for all } r \in Z\,. \tag{4.1}$$

Let C be a $\prec$-respecting Eulerian cycle in $G[A + S^*]$ and let L be its set of last arcs. Notice that $L \subseteq B + M_\prec + Z$, where $M_\prec$ is the set of maximal elements with respect to $\prec$ as defined in Step 1 of the algorithm.

The set L must contain a directed tree D' rooted towards o. This tree spans at least all those components which contain essential nodes. Since by Assumption 4.10 all nodes are essential, we can conclude that tree D' is a spanning tree. Now we can compare its weight to the weight of tree D which is computed in Step 3 of the algorithm. We have easily

$$c'(D') \geq c'(D)\,, \tag{4.2}$$

since D is of minimum weight.

We partition D' into the two sets $D'_1 := D' \cap Z$ and $D'_0 := D' \cap (B + M_\prec)$. By construction of weight function c' in Step 3, we have $c'(D'_0) = 0$ and $c'(D'_1) = c(D'_1)$. From (4.1) and the fact that D'_1 does not contain a pair of anti-parallel arcs we conclude $c(Z) \geq 2c(D'_1)$. Summarizing the results yet yields

$$c(Z) \geq 2c(D'_1) = 2c'(D'_1) + 2c'(D'_0) = 2c'(D') \overset{(4.2)}{\geq} 2c'(D)\,. \tag{4.3}$$

The set N computed in Step 5 has cost

$$c(N) = 2c(D - (B + M_{\prec})) = 2c'(D - (B + M_{\prec})) = 2c'(D) \overset{(4.3)}{\leq} c(Z)\,. \tag{4.4}$$

Using this result yields that

$$\begin{aligned} c(A + B + N) &= c(A + B) + c(N) \\ &= c(A + (S^* - Z)) + c(N) \\ &\overset{(4.4)}{\leq} c(A + (S^* - Z)) + c(Z) \\ &= c(A + S^*)\,. \end{aligned}$$

Thus, $B + N$ is an optimal solution as claimed.

We briefly comment on the running time of Algorithm 4.2. A balancing set B can be found in time $O(n + m_A)$ as argued in Lemma 4.13 on page 93. A rooted spanning tree of minimum weight in a graph with n nodes and m arcs can be computed in time $O(\min\{m \log n, n^2\})$ by the algorithm from [Tar77]. Thus, the problem SOURCE-DARP on the class of paths can be solved in total time $O(n + m_A + \min\{(m_A + n) \log n, n^2\})$ by Algorithm 4.2. □

4.6 An Approximation for Source-Darp on General Graphs

In this section we present an approximation algorithm for SOURCE-DARP on general graphs. We use an approach similar to that of [FHK78]. Throughout this section we will assume tacitly that Assumption 4.11 on page 90 is satisfied.

The final algorithm actually consists of two different sub-algorithms. The first of the sub-algorithms uses the observation that a cycle in the graph describing a transportation schedule is somehow related to a traveling salesperson tour. The second sub-algorithm is a slight modification of the algorithm presented in the previous section. The final algorithm simply runs both sub-algorithms and picks the best solution.

4.6.1 TSP-based Algorithm

A necessary condition for a solution to be valid is that the tour visits the source of each arc at least once. Denote by $V_{\text{source}} \subseteq V$ the set of nodes which is defined by

$$V_{\text{source}} := \{\, v \in V \mid d_A^+(v) > 0 \,\},$$

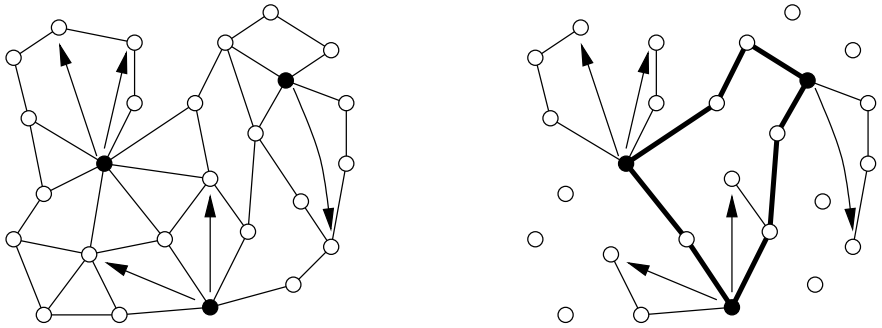

FIGURE 4.4: Illustration of Algorithm 4.5. Instance with marked nodes from V_{source} (left), resulting solution (right) consisting of TSP-tour (thick lines) and loops (arcs and thin lines).

i. e., V_{source} contains the nodes which are sources of arcs from A. The algorithm first computes (an approximation of) a TRAVELLING SALESPERSON instance with set V_{source} to be visited. Then the solution is augmented at each node from V_{source} by traversing a number of loops. Each loop consists of one outgoing arc and returns to the current node using edges of the graph. An illustration can be found in Figure 4.4. The details are displayed in Algorithm 4.5.

We now prove a bound on the quality of the solution found by the TSP-based Algorithm 4.5.

Theorem 4.22 (Performance of TSP-based Algorithm)
Let ρ_{TSP} be the performance of an approximation algorithm for TRAVELLING SALESPERSON. *Assume we employ that algorithm in Step 3 of Algorithm 4.5. Then Algorithm 4.5 finds a solution of cost at most*

$$\rho_{TSP} \cdot \mathrm{OPT} + 2c(A)\,.$$

Proof. Let S^* be an optimal augmenting set and C^* be a $\prec$-respecting Eulerian cycle in $G[A + S^*]$ starting at o. Recall that per definition, $c(C^*) = \mathrm{OPT}$.

Let l be the length of an optimal TSP tour visiting all nodes from V_{source}. Clearly $l \leq \mathrm{OPT}$ since even C^* must visit at least all nodes from V_{source}.

Let C be the approximate TSP tour computed in Step 3 of Algorithm 4.5. Then

$$c(C) \leq \rho_{TSP} \cdot l \leq \rho_{TSP} \cdot \mathrm{OPT}\,.$$

The additional cost incurred in Step 9 is bounded by $2c(A)$: each path added has the weight of the corresponding arc from A, which is a consequence of the definition of cost function c (see Definition 4.1 on page 86). □

Input: A mixed graph $G = (V, E, A)$, a cost function c on E, an initial node $o \in V$, and a source order $\prec$

1 Let V_{source} be the set of nodes which are sources of arcs from A.
2 Compute a complete undirected auxiliary graph U with node set V_{source}. Set the weight $d(v, w)$ of each edge $[v, w]$ to

$$d(v, w) := \text{dist}_E(v, w) .$$

3 Find an approximately shortest TSP tour p in U.
Assume that p visits the nodes of V_{source} in order $(v_0, v_1, \ldots, v_s, v_{s+1})$ where $v_0 = v_{s+1} = o$.
4 Construct a feasible tour C for SOURCE-DARP as follows:
5 Start with the empty tour C.
6 **for** $i := 0, \ldots, s$ **do**
7 Consider current node v_i.
Assume that $A_{v_i} = \{a_1, \ldots, a_k\}$ with $a_1 \prec \cdots \prec a_k$.
8 For $j = 1, \ldots, k$, let p_j be a directed shortest path in $G[E^{\rightleftarrows}]$ from the endpoint of a_j back to current node v_i.
9 Add the k loops $a_1 p_1, \ldots, a_k p_k$ to C.
10 Append to C the directed shortest path in $G[E^{\rightleftarrows}]$ from v_i to v_{i+1}.
11 **end for**
12 Let $S \leftarrow C - A$.

Output: the set S and the cycle C

ALGORITHM 4.5: TSP-based algorithm for SOURCE-DARP.

Obviously the cost of an optimal tour serving all jobs is at least $c(A)$. Thus,

$$\rho_{TSP} \cdot OPT + 2c(A) \leq (\rho_{TSP} + 2) \cdot OPT\,.$$

This means, that Algorithm 4.5 approximates SOURCE-DARP with approximation factor $\rho_{TSP} + 2$. Using Christofides' algorithm for approximating TRAVELLING SALESPERSON [Chr76] we have $\rho_{TSP} = 3/2$ and thus a final performance of $7/2$.

Corollary 4.23 (Performance of TSP-based Algorithm)
Algorithm 4.5 has a performance of

$$\rho_{TSP} + 2 \leq 7/2$$

for approximating SOURCE-DARP. □

4.6.2 Algorithm Based on Set of Last Arcs

We now describe Algorithm 4.6. This algorithm is based on similar ideas as Algorithm 4.2 from Section 4.5 for SOURCE-DARP on paths. It first computes a set B of balancing arcs which makes $G[A+B]$ degree balanced. Then it computes a rooted tree directed towards the origin o of minimum cost. After that the new arcs which are not yet in $A + B$, together with their inverse arcs, are added to the solution.

The difference between the current problem on general graphs and the problem on paths from the section before is that the computation of a balancing set is more complicated. Particularly, Lemma 4.13 on page 93 is no longer true, i. e., we can not guarantee that a specific balancing set is a subset of an appropriate optimal solution. This enforces to use a different analysis of the quality of the solution found by the algorithm.

At first it is easy to see that the set $B + N$ found by Algorithm 4.6 is indeed a feasible solution. The proof of this claim parallels that of Lemma 4.20 on page 97.

We now formulate some statements on the quality of the solution. We start with an estimation of the weight of the balancing set computed at the beginning.

Lemma 4.24
The balancing set B found in Step 1 of Algorithm 4.6 has cost at most $OPT - c(A)$.

Proof. Let $S^* \sqsubset E^{\rightleftarrows}$ be an optimal solution. Recall that $OPT = c(A) + c(S^*)$. Since solution S^* is feasible, graph $G[A + S^*]$ is $\prec$-Eulerian with start o. In particular, it is degree balanced. Therefore

$$OPT - c(A) = c(S^*) \geq c(B)\,,$$

since B is chosen as a balancing set of minimal weight. □

Input: A mixed graph $G = (V, E, A)$, a cost function c on E, an initial node $o \in V$, and a source order $\prec$
{ *Recall that Assumption 4.11 holds, in particular, each vertex is essential.* }
1 Compute a balancing multi-set $B \sqsubset E^{\rightleftarrows}$ of minimum cost.
{ *How this step can be accomplished with the help of a minimum cost flow computation is described in detail in Lemma 4.27.* }
2 Follow Steps 1 and 3 to 6 of Algorithm 4.2 to compute a set N of arcs and a $\prec$-respecting Eulerian cycle C with start o.
Output: the set $B + N$ and the cycle C

ALGORITHM 4.6: Algorithm for SOURCE-DARP based on a set of last arcs.

We continue to prove an upper bound on the cost of the set N of new arcs added in Step 2 of Algorithm 4.6 to the solution.

Lemma 4.25
The cost of the arc set N *computed by Algorithm 4.6 is at most* $2(\mathrm{OPT} - c(A))$.

Proof. The proof of the claim is similar to the one for Theorem 4.21 on page 99. The major difference is that in general we cannot assure that the balancing set B computed in Step 1 is a subset of an optimal solution. Therefore we use another approach for the analysis.

Let S^* be an optimal augmenting set and L be the set of last arcs of a $\prec$-respecting Eulerian cycle in $G[A + S^*]$. Then $L \subseteq M_\prec + S^* \subseteq A + S^*$ and hence

$$\mathrm{OPT} - c(A) = c(S^*) \geq c(L - (A + B)) .$$

Clearly, L contains no elements from $A \setminus M_\prec$. This implies that $L - (A + B) = L - (M_\prec + B)$ and

$$c(L - (A + B)) = c(L - (M_\prec + B)) = c'(L) .$$

The latter equality follows from the definition of cost function c'. Since tree D was of minimal weight, we have

$$c'(L) \geq c'(D) .$$

Using the same arguments as in Equation (4.4) from Theorem 4.21 on page 99 we conclude that

$$2c'(D) = c(N) ,$$

which shows the claim. □

We are now ready to estimate the weight $c(A + B + N)$ of the solution produced by our algorithm. By Lemma 4.24, $c(A + B) \leq \mathrm{OPT}$. Lemma 4.25 establishes that $c(N) \leq 2 \cdot \mathrm{OPT} - 2c(A)$. Thus $c(A + B + N) \leq 3 \cdot \mathrm{OPT} - 2c(A)$.

Theorem 4.26 (Performance of Algorithm Based on Set of Last Arcs)
Algorithm 4.6 finds a solution of cost at most $3 \cdot \mathrm{OPT} - 2c(A)$. □

We conclude this section with implementation details on the computation of the balancing set.

Lemma 4.27 (Construction of a Balancing Set)
Step 1 of Algorithm 4.6 can be accomplished in the time needed for one minimum cost flow computation on a graph with n *nodes and* $2m_E$ *arcs.*

Proof. We setup an auxiliary graph $F = (V, E^{\rightleftarrows})$. A node $v \in V$ has charge $d^-_A(v) - d^+_A(v)$. The cost of sending one unit of flow over arc $r \in E^{\rightleftarrows}$ equals its cost $c(r)$. Let $t\colon E \to \mathbb{N}_0$ be an arbitrary feasible integral flow in F, and define the multi-set B via

$$B(e) := t(e) \quad \text{for all } e \in E^{\rightleftarrows}.$$

Then B is a balancing set. Conversely, any balancing set implies a feasible integral flow in the graph of equal cost by applying the same rule backwards. Hence the cost of the balancing set and the flow are minimized at the same time. □

4.6.3 Combining Both Algorithms

As noted before, the final algorithm just runs both Algorithm 4.5 and Algorithm 4.6 and picks the better solution. The following analysis shows that by this approach one can gain an overall approximation factor which improves the performance of each sub-algorithm.

Theorem 4.28 (Approximability of SOURCE-DARP on General Graphs)
The combined algorithm has a performance of $1/2 \cdot (\rho_{TSP} + 3)$.

Proof. The result is obtained by comparing the approximation guarantees of both algorithms. Recall from the previous sections that there are the following bounds on the weight of a solution:

Algorithm 4.5 yields a solution of maximal weight $\rho_{TSP} \cdot \mathrm{OPT} + 2 \cdot c(A)$ (see Theorem 4.22 on page 101), while Algorithm 4.6 provides a solution of maximal weight $3 \cdot \mathrm{OPT} - 2 \cdot c(A)$ (see Theorem 4.26).

Factor ρ_{TSP} is a constant. Observe that if the quotient $c(A)/OPT$ is small, the first estimation provides the better result, while in case that $c(A)/OPT$ is large, the second algorithm is superior.

Let $\beta := 1/4 \cdot (3 - \rho_{TSP})$. If $c(A)/OPT \leq \beta$, then the first result is bounded from above as

$$\begin{aligned} \rho_{TSP} \cdot OPT + 2 \cdot c(A) &\leq (\rho_{TSP} + 2\beta) \cdot OPT \\ &= \left(\rho_{TSP} + 2 \cdot \frac{3 - \rho_{TSP}}{4}\right) OPT \\ &= \frac{\rho_{TSP} + 3}{2} \cdot OPT\,. \end{aligned}$$

Otherwise, if $c(A)/OPT > \beta$, then the second bound can be estimated as

$$\begin{aligned} 3 \cdot OPT - 2 \cdot c(A) &\leq (3 - 2\beta) \cdot OPT \\ &= \left(3 - 2 \cdot \frac{3 - \rho_{TSP}}{4}\right) OPT \\ &= \frac{\rho_{TSP} + 3}{2} \cdot OPT\,. \end{aligned}$$

Since the combined algorithm takes the best solution out of the two results, the final solution is bounded by $1/2 \cdot (\rho_{TSP} + 3) \cdot OPT$ in each case. □

Using Christofides' algorithm [Chr76] with $\rho_{TSP} = 3/2$ results in a performance guarantee of $9/4$ for the combined algorithm.

Theorem 4.29 (Approximability of SOURCE-DARP on general graphs)
There is an approximation algorithm for SOURCE-DARP *with performance*

$$\frac{1}{2}(\rho_{TSP} + 3) \leq \frac{9}{4}\,.$$

This algorithm can be implemented to run in time

$$O(\max\{n^3 + m_A m_E + m_A n \log n, m_E^2 \log n + m_E n \log^2 n\})\,.$$

Proof. The performance has already been proven. The running time of Algorithm 4.5 is dominated by that of Christofides' approximation algorithm for TRAVELLING SALESPERSON, which can be implemented to run in time $O(n^3)$, and the time needed for the addition of the paths in Step 9 which can be done in total time $O(m_A m_E + m_A n \log n)$. The running time of Algorithm 4.6 is dominated by the minimum cost flow computation which can be accomplished in time $O(m_E^2 \log n + m_E n \log^2 n)$ by using Orlin's enhanced capacity scaling algorithm [AMO93]. □

4.7 Improved Approximation for Source-Darp on Trees

Theorem 4.29 immediately implies a sharper approximation result on graph classes where TRAVELLING SALESPERSON can be approximated within factors better than 3/2. In particular, TRAVELLING SALESPERSON can be solved in polynomial time on trees, which means that $\rho_{\text{TSP}} = 1$ in that case. This implies the following result:

Corollary 4.30
There is an approximation algorithm for SOURCE-DARP *on trees with performance 2.* □

However, this factor can be improved by techniques analogously to the approach in the previous section. As before, we run two algorithms on the same instance. The first algorithm, namely Algorithm 4.5 which is based on the TSP approach, is employed without any changes here. The second algorithm, namely Algorithm 4.7, is a modified version of Algorithm 4.6 on page 104 from the previous section which is based on the set of last arcs.

We describe the modifications. The removal of non-essential nodes and the completion of G via shortest paths is deferred until after the balancing step. In contrast to the general case and similarly as for paths, even in the case of trees we can find a balancing set B which satisfies $B \sqsubset S^*$ as described in Lemma 4.13 on page 93. After the balancing we remove all nodes which are not incident to the arcs in $A + B$. At this point each node is in some sense "essential", i. e., it is incident with either an arc from A or an arc from B. From this situation on, the algorithm continues in the same manner as the algorithm from the previous section.

Input: A mixed graph $G = (V, E, A)$ where E is a tree,
a cost function c on E, an initial node $o \in V$, and a source order $\prec$

1 Compute a balancing multi-set $B \sqsubset E^{\rightleftarrows}$ such that $B \subseteq S^*$ for some optimal solution S^*
2 Complete the graph and remove all nodes not incident with arcs from $A + B$
3 Follow Steps 1 and 3 to 6 of Algorithm 4.2 to compute a set N of arcs and a $\prec$-respecting Eulerian cycle C with start o.
4 **return** the set $B + N$ and the cycle C

ALGORITHM 4.7: Algorithm for SOURCE-DARP on trees based on set of last arcs.

Theorem 4.31 (Performance of Algorithm Based on Set of Last Arcs)
Algorithm 4.7 finds a solution of cost at most $2 \cdot \mathrm{OPT} - 2 \cdot c(A)$.

Proof. We estimate the weight of the solution produced by Algorithm 4.7. Let $I = (G = (V, E, A), c, o, \prec)$ be the original instance such that $G[E]$ is a tree. Consider the instance $I' = (G = (V, E, A + B), c, o, \prec)$ which results from adding the balancing arcs B as new transportation jobs. Obviously I' is still an instance on a tree. By Lemma 4.13 it follows that any feasible solution to I will have to use the arcs from B anyway. Therefore we get that

$$\mathrm{OPT}(I) = \mathrm{OPT}(I') .$$

Denote by I'' the instance of SOURCE-DARP which is obtained from instance I' by completing G along shortest paths and removing non-essential nodes as in Step 2 of Algorithm 4.7. It is easy to see that

$$\mathrm{OPT}(I') = \mathrm{OPT}(I'') .$$

Instance I'' is no longer an instance on a tree, but it still obeys Assumption 4.11 for general graphs. Notice also that we can transform any feasible solution of I'' to a feasible solution of I' of the same weight: simply replace each arc which is not in $E^{\rightleftarrows}$ by the shortest path between its endpoints.

Denote by S^* an optimal solution for instance I. Define multi-set $Z := S^* \setminus B$. Similarly, let S'' be an optimal solution of I''. Since the multi-set $A + B$ consists of a collection of balanced strongly connected components and the instance is a complete graph obeying the triangle inequality, we can assume that S'' does not contain arcs with both endpoints in the same component. Consequently, $S'' \cap (A + B) = \varnothing$. Since

$$c(A + B) + c(Z) = \mathrm{OPT}(I) = \mathrm{OPT}(I'') = c(A + B) + c(S'') ,$$

we have that

$$c(Z) = c(S'') . \tag{4.5}$$

Let $A + B + N$ be the solution of instance I found by Algorithm 4.7. Then,

$$c(A + B + N) = c(A + B) + c(N) = c(S^*) - c(Z) + c(N) . \tag{4.6}$$

Now we follow the arguments of Lemma 4.25 on page 104 but take into account that in the current situation of instance I'', set S'' is the optimal augmenting set. Thus the mentioned result reads

$$c(N) \leq 2c(S'') \overset{(4.5)}{=} 2c(Z) \tag{4.7}$$

in the current context. Hence the right hand side of (4.6) can be further bounded from above by

$$\begin{aligned} c(S^*) - c(Z) + c(N) &\overset{(4.7)}{\leq} c(S^*) + c(Z) \\ &= OPT(I) - c(A) + OPT(I) - c(A+B) \\ &\leq 2 \cdot OPT(I) - 2 \cdot c(A)\,, \end{aligned}$$

and the claim is shown. □

Recall that from Theorem 4.22 on page 101 with the observation that $\rho_{TSP} = 1$ on trees we conclude that the first Algorithm 4.5 produces a solution of weight no more than

$$OPT + 2 \cdot c(A)$$

when restricted to tree instances. Running both algorithms and picking the best solution yields the following result similarly to the proof of Theorem 4.28:

Theorem 4.32 (Approximability of SOURCE-DARP on Trees)
There exists a polynomial time approximation algorithm for SOURCE-DARP *on trees with performance* 3/2. *This algorithm can be implemented to run in time* $O(nm_A + n^2 \log n)$.

Proof. If $c/OPT \leq 1/4$, then the cost of the solution produced by Algorithm 4.5 is bounded by

$$OPT + 2c(A) \leq OPT + \frac{1}{2} OPT = \frac{3}{2} OPT\,.$$

Otherwise, when $c/OPT > 1/4$, from Theorem 4.31 we know that the cost of the solution produced by Algorithm 4.7 is bounded by

$$2 \cdot OPT - 2 \cdot c(A) \leq 2 \cdot OPT - \frac{1}{2} OPT = \frac{3}{2} OPT\,.$$

This yields an overall performance of 3/2 as claimed.

The time bound for the algorithm is derived as follows: We can solve TRAVELLING SALESPERSON on the metric space induced by $G[E]$ in time $O(n)$. We then root the tree $G[E]$ at an arbitrary node. With $O(n)$ preprocessing time, the least common ancestor of any pair of nodes can be found in constant time (see [HT84, SV88]). Thus, we can implement Algorithm 4.5 in such a way that the invocations of Step 9 take total time $O(nm_A)$. This means that Algorithm 4.5 can be implemented to run in time $O(nm_A)$.

The balancing in Algorithm 4.7 can be accomplished in time $O(n + m_A)$ (see Lemma 4.13 on page 93). Completion of the graph by computing all-pairs shortest paths can be done in time $O(nm_E + n^2 \log n) \subseteq O(n^2 \log n)$ [CLR90, AMO93]. All other steps can be carried out in time $O(n^2)$ where again the algorithm from [Tar77] is employed for computing a minimum weight directed spanning tree. □

4.8 Source-Darp with Start and Stop Penalties

In this section we show how to extend our results to the case when there are start- and stop-penalties for the service vehicle. The approach with penalties makes the problem more realistic in view of applications. In elevator systems the time that the elevator needs to accelerate or decelerate in order to pick up or deliver its load can usually not be neglected. Thus, it is natural to penalize each stop and start of the server on its route.

Start- and stop-penalties do not introduce a completely new situation: the problem with penalties can be modeled as DARP on a slightly larger graph. However, the structure of the graph may be destroyed.

Particularly the penalties change the complexity of the problem when restricted to the simplest class of graphs. We prove that the problem with penalties becomes NP-hard even on paths without any precedence constraints, which contrasts with the polynomial solvability of the problem without penalties.

The problem SOURCE-DARP with penalties is denoted by PENALTY-SOURCE-DARP. An instance of PENALTY-SOURCE-DARP is specified by the same input as for SOURCE-DARP, and it defines additionally two nonnegative penalty functions p^+ and p^- on the set of nodes. Here, $p^+(v)$ is the time penalty for starting from a node and $p^-(v)$ is the penalty for stopping at the node. The objective is to find a closed walk serving all requests, such that the cost of the walk plus the cost of starting and stopping is minimized.

As also in the preceding sections we formulate PENALTY-SOURCE-DARP as a graph augmentation problem. In order to perform this in a meaningful way, we have to allow augmenting arcs from $V \times V$ and not just from $E^{\rightleftarrows}$, since each arc corresponds to a move and incurs a start and stop penalty. The cost function $c\colon E \to \mathbb{R}_0^+$ is extended by defining the cost of arc (v, w) to be the length of a shortest path from v to w in $G[E]$.

Definition 4.33 (Graph Augmentation Version of PENALTY-SOURCE-DARP) *An instance of problem* PENALTY-SOURCE-DARP *consists of the same input as for* SOURCE-DARP *(see Definition 4.5 on page 88) together with additional*

penalty functions $p^+, p^-\colon V \to \mathbb{R}_0^+$ *on the set* V *of nodes. The objective is to find a multi-set* S *of arcs,* $S \sqsubset V \times V$*, minimizing the weight*

$$c(A + S) + \sum_{u \in U^+} d^+(u)p^+(u) + \sum_{u \in U^-} d^-(u)p^-(u)$$

such that $G[A + S]$ *is* $\prec$*-Eulerian with start* o*. Here,* U^+ *is the set of sources of arcs in* $A + S$ *and* U^- *is the set of targets of arcs in* $A + S$*.*

In the sequel we show that problem PENALTY-SOURCE-DARP can be reduced to SOURCE-DARP. We emphasize that by the reduction the restriction on the shape of the underlying graph class may be violated. Therefore we cannot transfer the results from previous sections directly to the case with penalties.

Let $I = (G = (V, E, A), c, o, \prec, p^-, p^+)$ be an instance of PENALTY-SOURCE-DARP. We construct an instance $I' = (G' = (V', E', A'), c', o', \prec')$ of SOURCE-DARP on a slightly larger graph G'. For each node $v \in V$ we add both v and a new node $v^{(\pm)}$ to V'. Node $v^{(\pm)}$ is used to model starting or stopping at node v. The set E' consists of the edges in E and an additional edge e_v between v and $v^{(\pm)}$ for each node $v \in V$. The cost of the new edges is $c'(e_v) = (p^+(v)+p^-(v))/2$. The cost function c' coincides with c on the set E. For each arc $a = (u, v) \in A$ we add an arc $a' = (u^{(\pm)}, v^{(\pm)})$ to A' (the arcs in A are not contained in A'). The partial order on the set A' is induced naturally by that on A. Finally, the start node o' equals o.

Lemma 4.34 (Reducibility of PENALTY-SOURCE-DARP)
Let $I = (G, c, o, \prec, p^+, p^-)$ *be an instance of* PENALTY-SOURCE-DARP *and* $I' = (G', \prec, c', o')$ *be the instance of* DARP *constructed by the above method. Then,* I *and* I' *are equivalent in the following sense: Any feasible solution for* I' *can be transformed into a feasible solution for* I *of the same cost and vice versa. This transformation can be accomplished in polynomial time.*

Proof. Let S' be a valid solution for instance I' of SOURCE-DARP where S' is an augmenting set of arcs. Let C' be a $\prec$-respecting Eulerian cycle in $G'[A' + S']$ with start o'.

We first construct an auxiliary set M of arcs by traversing C' and replacing all chains of arcs from S' with a single arc from the start node of the chain to its end node. Notice that all endpoints of arcs in M are contained in $V' \setminus V$. We now construct a solution S by replacing each arc $(u^{(\pm)}, v^{(\pm)})$ by (u, v). It is easy to see that S is in fact a valid solution for I of cost equal to that of S'.

Conversely, let S be a feasible solution for I. We can construct a solution S' for I' with equal cost by adding for each arc (u, v) in S the arc $(u^{(\pm)}, v^{(\pm)})$ to S'.

The polynomial time bound is obvious from the construction. □

When drawing consequences from Lemma 4.34 must must take into account that the graphs $G[E]$ and $G'[E']$ used in the reduction do not necessarily belong to the same graph class. It follows from the construction that if $G[E]$ is a tree then $G'[E']$ is also a tree. Thus, Lemma 4.34 implies that approximation results for SOURCE-DARP on trees can be applied directly to PENALTY-SOURCE-DARP on trees. Similarly, approximation results for general graphs carry over immediately. Hence, we obtain the following result:

Corollary 4.35 (Approximability of PENALTY-SOURCE-DARP)
The problem PENALTY-SOURCE-DARP *can be approximated on trees with performance* $3/2$ *and on general graphs with performance* $9/4$. □

However, transforming an instance of PENALTY-SOURCE-DARP where $G[E]$ is a path yields an instance of SOURCE-DARP where $G'[E']$ is a caterpillar graph. In Theorem 4.36 we will show that SOURCE-DARP is NP-hard to solve on caterpillars. Unless we can give a better transformation, we can conjecture that even PENALTY-SOURCE-DARP on paths is NP-hard to solve. In the next section we will give a proof for this fact.

4.9 Hardness Results

Since SOURCE-DARP generalizes DARP, it follows from the hardness result in [FG93] that SOURCE-DARP is NP-hard even on trees. We show that this hardness continues to hold even if the source order $\prec$ is a total source order.

We can also strengthen the hardness result of [FG93] and show that DARP is hard on caterpillar graphs. This is in contrast to an application of DARP in the next section where caterpillar graphs naturally arise.

Recall the definition of a caterpillar graph (see page 5): A caterpillar graph consists of the *backbone*, which is a simple path, and the *feet*, which are additional leaves adjacent via hair edges to the backbone. We restrict the class of caterpillars further to those graphs where no two hairs are incident. Equivalently, the nodes on the backbone are of maximum degree 3.

Theorem 4.36 (Hardness of DARP)
DARP *on caterpillars is NP-hard to solve. This result continues to hold, if the transportation jobs are restricted to have sources and targets only in the feet of the caterpillar.*

Proof. The hardness is shown by a reduction from BIPARTITE STEINER TREE. An instance of BIPARTITE STEINER TREE consists of a bipartite graph $H = (X \cup Y, F)$ and a nonnegative number $k \leq |F|$. It is NP-complete to decide whether

there exists a subtree of H that spans all the nodes in Y and has at most k edges [GJ79, Problem ND12].

One can make two assumptions on H without loss of generality: First, each node in Y has degree at least two. Otherwise, for a node $y \in Y$ with degree one, there is no other choice besides including the unique edge incident on y in the Steiner tree. The second assumption is that H is connected. Otherwise, either there is a connected component containing Y, or there is no feasible solution at all.

Let $H = (X \cup Y, F)$ be an instance of BIPARTITE STEINER TREE. We create an instance $I = (G = (V, E, A), c, o)$ of DARP. The construction of graph G is illustrated in Figure 4.8. Start with a graph consisting of $2|F|$ nodes and $|F|$ pairwise non-incident edges. For each edge $[x, y] \in F$ where $x \in X$ and $y \in Y$, choose a yet unlabeled edge in G and label its endpoints by x and y, respectively. Now create the backbone of the caterpillar graph by inserting a path of $|F| - 1$ additional edges. These backbone edges are inserted between nodes with labels from X in such a way that for each $x \in X$ the graph induced by the nodes labeled x in G is a connected path, denoted by $P(x)$.

Set the weight of a backbone edge with two endpoints sharing the same label to 0, and the weight of backbone edges with two different labels to some large number $M := 2|F| + 1$. Set the weight of a hair to 1.

The arc set A is constructed as follows: For $y \in Y$ denote by $S(y)$ the set of foot nodes in G labeled with y. Then, for each $y \in Y$, choose a directed simple cycle connecting $S(y)$ and add its arc set to A. Finally, the origin o of the server is chosen to be the source of an arbitrary arc in A. Observe that by construction the graph $G[A]$ is degree balanced. It consists of the set of connected components $\{ S(y) \mid y \in Y \}$. Each of the components is strongly connected and Eulerian.

Let $C = \sum_{a \in A} c(a)$. We claim that H contains a Steiner tree with at most k edges if and only if there is a feasible solution to the instance I of DARP with cost at most $C + 2k$. This statement will be established in Claim 4.37 and Claim 4.38, and this completes the proof. □

Claim 4.37
Assume that graph H contains a Steiner tree with at most k edges. Then there is a feasible solution to the instance I with cost at most $C + 2k$.

Proof. Suppose that T is a Steiner tree in H with at most k edges connecting the nodes in Y. We construct a multi-set S of arcs, $S \sqsubset E^{\rightleftarrows}$, such that $G[A + S]$ is Eulerian and contains o and $c(A + S) \leq C + 2k$: For each $x \in X$ spanned by T, add all arcs from the set $P(x)^{\rightleftarrows}$ to S (these arcs are of cost 0). For each edge $[x, y] \in T$, add a pair of anti-parallel arcs between the endpoints of the

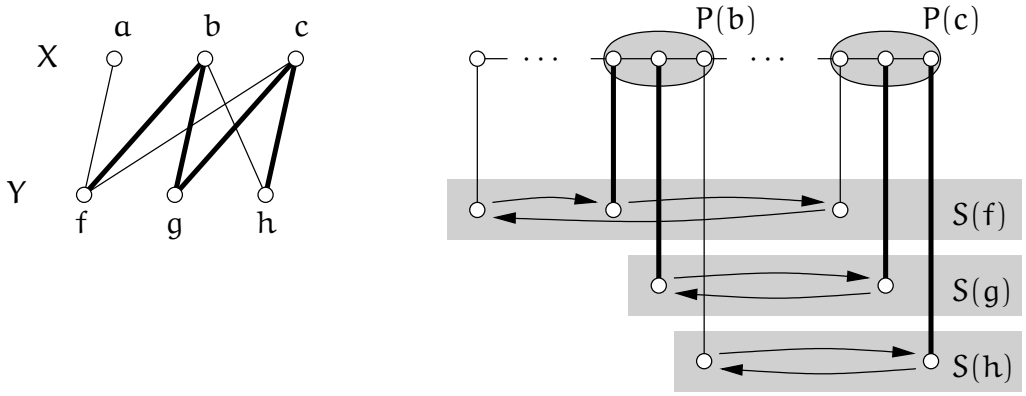

FIGURE 4.8: Transformation of BIPARTITE STEINER TREE into DARP on a caterpillar in Theorem 4.36. Thick lines represent a valid solution and its transformation.

unique hair labeled $[x, y]$ to multi-set S (these arcs are of cost 1 each). By this construction, graph $G[A + S]$ is degree balanced. Since T was spanning Y and connected, $G[A + S]$ is strongly connected and hence Eulerian. By construction, $c(A + S) \leq C + 2k$. □

Claim 4.38
Assume that there is a feasible solution to the instance I *with cost at most* $C+2k$. *Then graph* H *contains a Steiner tree with at most* k *edges.*

Proof. Assume that S, $S \sqsubset E^{\rightleftarrows}$, is a feasible solution for instance I, and its cost satisfies $c(A + S) \leq C + 2k$. Then $G[A + S]$ is Eulerian and contains o. The set S cannot contain any arc of cost M, since otherwise $c(A + S) = c(A) + c(S) \geq C + M = C + 2|F| + 1 > C + 2k$.

We now define a subgraph T of H as follows: For each $x \in X$, $y \in Y$, if S contains (at least one copy) of an arc between a node in $P(x)$ and $S(y)$ in arbitrary direction, then the edge $[x, y]$ is included in T. Since $G[A]$ and $G[A + S]$ are degree balanced and $S \cap A = \varnothing$, we can decompose S into arc disjoint cycles. Since $S \sqsubset E^{\rightleftarrows}$ and $G[E]$ is a tree it follows that $r \in S$ implies that the inverse arc r^{-1} must also be contained in S. Hence, T consists of at most $c(S)/2 = k$ edges.

It remains to show that T is connected and spans the nodes in Y. To this end, let y_1 and y_2 be two arbitrary nodes from Y. Since y_1, y_2 are incident with arcs from A, and $G[A + S]$ is strongly connected, there is a directed path $(r_1, \ldots, r_t)$ from a node labeled y_1 to one labeled y_2 in $G[A + S]$. The node labels along this path change only when r_i is of type $r_i = (x, y)$ or $r_i = (y, x)$ for suitable $x \in X$

and $y \in Y$. By construction, tree T contains edge $[x', y']$ in this case. Hence, T connects y_1 and y_2. □

Corollary 4.39 (Hardness of SOURCE-DARP)
All hardness results given in Theorem 4.36 for DARP *remain true for* SOURCE-DARP. *This is true even if the source order is restricted to be total.*

Proof. Observe that in the construction used above, $|A_v| \leq 1$ for all nodes v. So we can choose $\prec$ to be the empty relation and the claim immediately follows. □

We now turn over to problem PENALTY-SOURCE-DARP. The caterpillar constructed in the proof of Theorem 4.36 has the property that jobs have sources and targets only in the feet of the caterpillar. In fact, every instance of SOURCE-DARP on caterpillars with these properties can be transformed into an equivalent instance of PENALTY-SOURCE-DARP on a path: Let f be a foot and v be its unique adjacent node on the backbone. We replace all arcs from A which are incident with f by corresponding arcs with source or target v. We then remove foot f. The start- and stop-penalty on v is set to the length $c(f, v)$ of the hair between v and the foot f. It follows by arguments similar to those given in Lemma 4.34 that the constructed instance of PENALTY-SOURCE-DARP on the path (which corresponds to the former backbone) is in fact an equivalent instance to the instance of SOURCE-DARP on the caterpillar. Thus, we obtain the following result which contrasts with the polynomial solvability of SOURCE-DARP on paths:

Theorem 4.40 (Hardness of PENALTY-SOURCE-DARP)
PENALTY-SOURCE-DARP *is NP-hard even on paths.* □

4.10 Concluding Remarks

We have contributed several versions of a dial-a-ride problem with additional precedence constraints on the transportation jobs. Originally this research was motivated by the performance analysis of a large distribution center of one of the leading manufacturers of office products in Central Europe [AG+98].

We have shown that even in the presence of source order constraints for the transportation jobs the problem can be solved in polynomial time on paths which generalizes the result of [AK88]. On trees, however, the problem is NP-hard. The application to realistic elevator systems with non-negligible acceleration of the moving server motivates the formulation of a problem variant with start and stop penalties. Table 4.9 gives an overview on the results.

Graph class	DARP	SOURCE-DARP	PENALTY-SOURCE-DARP
paths	polynomial time solvable *[AK88]*	polynomial time solvable *(Theorem 4.21)*	NP-hard *(Theorem 4.40)*
			approximable within 5/3 *(Corollary 4.35)*
trees	NP-hard, even on caterpillars *(Theorem 4.36)*	NP-hard, even on caterpillars *(Theorem 4.36)*	NP-hard
	approximable within 5/4 *[FG93]*	approximable within 3/2 *(Theorem 4.32)*	approximable within 5/3 *(Corollary 4.35)*
general graphs	NP-hard *[FHK78]*	NP-hard	NP-hard
	approximable within 9/5 *[FHK78]*	approximable within 9/4 *(Theorem 4.29)*	approximable within 9/4 *(Corollary 4.35)*

TABLE 4.9: Complexity and approximation results for DARP and related problems.

Part II

Network Upgrade Problems

Chapter 5

Node Upgrade Problems

In the remaining part of the thesis we turn over to *network upgrade* problems as opposed to the network design problems examined in the first part. In this chapter we examine a network upgrade problem under a *node upgrade* model. The node upgrade model admits to distribute a budget among the nodes of the graph in order to upgrade a subset of the nodes. Upgrading a particular node results in decreasing the length of all incident links.

Our first problem can be characterized as a *bottleneck* problem. The final solution is (a subgraph of) a bottleneck graph which is constructed from a graph by deleting all links whose length exceeds the bottleneck value. By this operation it is guaranteed that each link length which appears in the final solution is bounded by a global constant. Problems of bottleneck type have applications e. g. in communication networks with electrical signal transmission: Given the fact that the resistance of cables is greater than zero, the quality of signals decreases along the cable. There is a reverse dependency between the signal quality and the length of a link. By this relation, a minimum requirement on the signal quality (which is itself determined by the parameters of amplifiers used and additional factors including maximal error rate) immediately induces a bound on the maximal length of a link.

The underlying network design problem is to find a *constrained forest* as introduced in [GW95]. The concept of constrained forests is a generalization of subgraph classes including spanning trees, Steiner trees, or s-t-paths.

The problem we concentrate on in this chapter is called NODE UPGRADE CONSTRAINED FOREST: The goal is to find a minimum cost set of nodes, such that the bottleneck graph which results from the upgrade contains a constrained forest. The constrained forest is implicitly specified by a proper function. We prove that the problem is NP-hard to solve. Moreover, we provide both logarithmic lower bound and logarithmic upper bound on the approximability of NODE UPGRADE CONSTRAINED FOREST in this chapter.

5.1 Preliminaries and Problem Formulation

5.1.1 Node Upgrade Model

At first we start with a description of the network upgrade model used in this chapter. Initially we are given an edge weighted graph and additionally a budget. The budget can be spent to improve the network in order to reduce the weight of some edges.

The upgrade model is *node based*. This means, that the upgrade operation takes places at the nodes of the graph. We use a *discrete* upgrade model: for each node we can either choose to upgrade it or to leave it in the original state. The effect of upgrading a particular node is that the weight of all edges incident with that node is reduced. The amount of reduction is computed as follows: The input instance specifies for each edge e three integer numbers

$$d_0(e) \geq d_1(e) \geq d_2(e) \geq 0\,.$$

Number $d_0(e)$ represents the initial weight of edge e. If one of the endpoints of e is upgraded, the weight falls to $d_1(e)$. If both endpoints are upgraded, the weight of the edge is determined by $d_2(e)$.

The cost of an upgrade strategy is determined by an additional integer valued function c on the set of nodes. For each node $v \in V$ the value $c(v)$ specifies how expensive it is to upgrade the node.

Any upgrade strategy is uniquely determined by specifying a subset $W \subseteq V$ of the set V of nodes which contains all upgraded nodes. The cost $c(W)$ of an upgrade strategy is defined in an obvious way as

$$c(W) := \sum_{v \in W} c(v)\,.$$

The edge weight function resulting from an upgrade strategy W is denoted by d_W. It is defined to be

$$d_W(e) := \begin{cases} d_0(e), & \text{if } |\{v,w\} \cap W| = 0 \\ d_1(e), & \text{if } |\{v,w\} \cap W| = 1 \\ d_2(e), & \text{if } |\{v,w\} \cap W| = 2 \end{cases}$$

for each edge $e \in E$, $e = (v,w)$.

5.1.2 Constrained Forest Problems

Constrained forests are used to characterize certain well known subgraph structures including the family of spanning trees, the familiy of Steiner trees (with

respect to a given set of terminals), or the family of s-t-paths (for a given pair s, t of nodes) in a unified way.

Constrained forest problems were introduced by Goemans and Williamson in [GG+94, GW95]. An instance specifies a graph and a family of cuts in the graph. A subgraph is a feasible solution for the problem if it intersects each of the cuts in the family, i. e., it contains at least on edge out of each cut. The goal is to find a minimum weight feasible subgraph. For the following observations we refer the reader also to [GW97].

Consider a graph $G = (V, E)$. Observe that each node subset $U \subset V$ which is neither the empty set nor the set V itself defines a cut

$$\delta(U) := \{ (v, w) \in E \mid v \in V, w \notin V \}$$

which contains all edges with exactly one endpoint in U. We can use a characteristic function $f\colon 2^V \to \{0, 1\}$ to define a family of cuts: $f(U) = 1$ if and only if $\delta(U)$ is a member of the family. A cut $\delta(U)$ where $f(U) = 1$ is called *active*.

A cut $\delta(U)$ is called *intersected* by an edge set $F \subseteq E$, if $\delta(U) \cap F \neq \varnothing$. Observe that there is the symmetry

$$\delta(U) = \delta(V \setminus U)$$

for all subsets $U \in 2^V \setminus \{\varnothing, V\}$. Moreover, for arbitrary disjoint subsets $U_1, U_2 \in 2^V \setminus \{\varnothing, V\}$, we have

$$\delta(U_1 \cup U_2) \subseteq \delta(U_1) \cup \delta(U_2)\,.$$

It is clear that whenever the cut $\delta(U_1 \cup U_2)$ is intersected by an edge set F, then at least one of the cuts $\delta(U_1)$ or $\delta(U_2)$ must also be intersected by F as well. These observations motivate the restriction of the characteristic functions to proper functions:

Definition 5.1 (Proper Function)
*A function $f\colon 2^V \to \{0, 1\}$ is called **proper**, if it satisfies:*

1. *Symmetry:*

$$f(U) = f(V \setminus U) \quad \textit{for all } U \in 2^V$$

2. *Disjointness: For any $U_1, U_2 \in 2^V \setminus \{\varnothing, V\}$,*

$$\big(U_1 \cap U_2 = \varnothing \ \wedge\ f(U_1) = f(U_2) = 0 \big) \implies f(U_1 \cup U_2) = 0\,.$$

A subgraph which intersects all active cuts as specified by a proper function is called a *constrained forest*. By this understanding one can use a proper function to define a family of subgraphs.

Definition 5.2 (Constrained Forest)
Let $G = (V, E)$ *be a graph,* $f\colon 2^V \to \{0, 1\}$ *be a proper function. Consider a subset* $F \subseteq E$ *such that*

$$|F \cap \delta(U)| \geq f(U), \quad \text{for all } U \in 2^V \setminus \{\varnothing, V\}.$$

Then the induced graph $G[F]$ *is termed a* ***constrained forest with respect to*** f.

Any vertex $v \in V$ such that $f(\{v\}) = 1$ is called a *terminal*. In the sequel we denote by $K := \{v \in V \mid f(\{v\}) = 1\}$ the set of terminals given by the proper function f.

Many interesting families of problems can be formulated as constrained forest problems with proper functions. We give three examples.

Example 5.3 (Spanning Tree). In the most basic case, the proper function f is defined by

$$f(U) = 1 \quad \text{for all } U \in 2^V \setminus \{\varnothing, V\}.$$

This means that the constrained forest must contain at least one edge of each cut in the graph. Therefore, the corresponding subgraph is connected. The inclusion-wise minimal constrained forests are the spanning trees of the input graph, while a weight minimal constrained forest is a minimum spanning tree.

Example 5.4 (s-t-Path). Assume that there are two distinct nodes $s, t \in V$ given. Define the proper function f to be

$$f(U) = 1 \quad \text{for all } U \in 2^V \text{ where } |U \cap \{s, t\}| = 1.$$

Obviously f makes exactly those cuts active which separate s from t. Hence each constrained forest contains a path between s and t, and a weight minimal constrained forest is a shortest path connecting the two nodes.

Example 5.5 (Steiner Tree). Another example of the use of proper functions is the application to STEINER TREE. Let $K \subseteq V$ be a set of terminals. The problem STEINER TREE is to find a minimum weight connected subgraph that spans all the terminals. The corresponding proper function f is defined by

$$f(U) = \begin{cases} 1 & \text{if } 0 < |U \cap K| < |K| \\ 0 & \text{otherwise.} \end{cases}$$

This choice of f means that each cut separating the set of terminals is an active cut. Therefore a constrained forest spans all terminals, and a weight minimal constrained forest is a solution to STEINER TREE.

Notice that the cardinality of the power set of V is of size exponential in $|V|$. If a proper function were given by explicitly quoting all values in a table, then the size of an input instance would no longer remain polynomial in the size $|V|$. This would imply that the complexity results could not be compared to results on other problems in a meaningful sense (see also the discussion on the encoding of input instances in Section 1.2).

A way out of this dilemma is to use an implicit specification of the proper function. The three examples above show how to use this implicit representation. In general it is sufficient that the proper function is specified in such a way that for each particular subset of the nodes we can *compute* its value in polynomial time.

Before continuing with the formulation of the main problem we report the following result which has been proven in [GW95].

Lemma 5.6
Let f be a proper function. If $f(U) = 0$ and $f(B) = 0$ for some $B \subseteq U$, then $f(U \setminus B) = 0$. □

5.1.3 Problem Formulation

As noted in the introduction to this chapter, we will investigate a bottleneck constrained forest problem under the node upgrade model introduced in Section 5.1.1. The bottleneck value is a number $D \in \mathbb{R}^+$ specified by the problem instance. Given a graph $G = (V, E)$ with edge weight function d, we define the *bottleneck graph*

$$\text{Bottleneck}(G, d_W, D)$$

to contain all edges $e \in E$ with $d_W(e) \leq D$.

Given a bottleneck value, we partition the set of edges into four sets. This partition is performed according to how many of the endpoints must be upgraded in order to decrease the delay of an edge below the threshold D:

- *uncritical edges:* An edge of delay $d_0(e) \leq D$ is called *uncritical* because it is part of the bottleneck graph regardless whether an upgrade happens or not.
- *1-critical edges:* An edge e is said to be *1-critical*, if $d_0(e) > D \geq d_1(e)$. Those edges need at least on endpoint to be selected for upgrading until they get included into the bottleneck graph.

- *2-critical edges:* Similarly, an edge e is called *2-critical*, if $d_1(e) > D \geq d_2(e)$. In this case it is necessary to upgrade both endpoints before the edge gets part of the bottleneck graph.
- *useless edges:* Finally, if $d_2(e) > D$, the edge e is called *useless*, since no upgrade strategy can cause those edges to be added to the bottleneck graph. Without loss of generality we assume that the graph does not contain any useless edges.

We are now ready to formulate the problem NODE UPGRADE CONSTRAINED FOREST under study.

Definition 5.7 (NODE UPGRADE CONSTRAINED FOREST Problem)
Given a graph $G = (V, E)$ *with nonnegative edge weights* $d_0 \geq d_1 \geq d_2$*, node weights* c *as before, a bound* D *and a proper function* f*, find a minimum cost set* $W \subseteq V$ *of nodes such that the resulting graph with edges weights given by* d_W *has a constrained forest (with respect to* f*) of bottleneck delay at most* D*.*

Notice that the condition just stated is equivalent to saying that after the upgrade operation the set of all edges of weight at most D forms a constrained forest. The canonical notation of the problem as a two-criteria optimization problem is

(MIN: NODE UPGRADE, CONSTRAINED FOREST BOTTLENECK WEIGHT) .

It will turn out in Section 5.5 that NODE UPGRADE CONSTRAINED FOREST is NP-hard to solve. This motivates the contruction of an approximation algorithm in the following sections.

Observe that given a vertex set $W \subseteq V$ it can be easily checked in polynomial time whether W is a valid upgrading set. This can be achieved by computing the bottleneck graph $H := \text{Bottleneck}(G, d_W, D)$ and evaluating the proper function for each connected component of H. In fact, we claim that W is valid if and only if f evaluates to zero on each component of H: Clearly, if $f(C) = 1$ for a connected component C of H then H can not contain a constrained forest, since any such forest must have an edge with exactly one endpoint in C. Assume conversely that $f(C) = 0$ for each connected component C of H. If H contained no constrained forest, there would be a set $U \in 2^V \setminus \{\varnothing, V\}$ with $f(U) = 1$ and $\delta(U) \cap H = \varnothing$. But then U was the disjoint union of components of H and from the disjointness property of f we obtain the contradiction that $f(U) = 0$.

5.2 Related Work

The problem NODE UPGRADE CONSTRAINED FOREST generalizes the problem of finding a node-weighted Steiner tree of minimum cost. An instance of NODE WEIGHTED STEINER TREE is given by a graph $G = (V, E)$ with edge weights l and node weights w. For a subset $K \subseteq V$ of terminals, the problem consists of finding a connected subgraph of G spanning all the terminals. The optimization objective is to minimize the total sum of node and edge weights of the subgraph.

Let an instance of NODE WEIGHTED STEINER TREE be given. Notice that without loss of generality we can assume that all edge weights are zero: If not, we can replace each edge (u, v) by two new edges (u, x) and (x, v), where x is a new vertex of weight $l(u, v)$. We can now construct an instance of NODE UPGRADE CONSTRAINED FOREST by taking the graph G specified in the NODE WEIGHTED STEINER TREE instance and defining edge weights

$$d_0(e) := d_1(e) := 2$$
$$d_2(e) := 1\,.$$

We set the bottleneck threshold D to be 1. The cost of upgrading a vertex v is set to $c(v) := w(v)$. The proper function f is defined as in Example 5.5 on page 122 to reflect Steiner trees for the terminal set K.

For each solution of NODE WEIGHTED STEINER TREE the vertices in the tree induce a upgrading set whose cost equals that of the tree. Moreover, all edges of the solution have both endpoints upgraded, hence the bottleneck graph contains the tree. Conversely, the bottleneck graph solution of NODE UPGRADE CONSTRAINED FOREST is also a solution of NODE WEIGHTED STEINER TREE, and the sum of the node weights in the latter does not exceed the upgrade cost of the first.

The problem of finding a node-weighted Steiner tree of minimum cost has been introduced in [Seg87]. By a reduction from MINIMUM SET COVER there is a lower bound of $\ln|K|$ for approximating a $|K|$-terminal instance, unless NP $\subseteq$ DTIME$(N^{O(\log\log N)})$. Klein and Ravi [KR95] obtained an approximation with performance $2\ln|K|$. This approximation result has been improved by Guha and Khuller [GK99b] to $1.35\ln|K|$ for weighted nodes and to $\ln|K|$ for nodes with unit weights.

The generalized problem of finding a minimum node weight constrained forest (specified by proper functions) has also been addressed by Guha and Khuller in [GK99b] where they provide an approximation with performance $1.6103\ln|K|$.

The concept of constrained forests is due to Goemans and Williamson [GW95, GW97]. They have shown that finding a minimum weight constrained forest in

an edge weighted graph is NP-hard, and gave a 2-approximation algorithm for the problem.

A simpler node upgrade model was suggested by Paik and Sahni [PS95]: Their model uses as input a graph with nonnegative edge weight function $d \geq 0$ and a global constant $\alpha \in \mathbb{R}$ with $0 < \alpha < 1$. The weight of an edge remains untouched if none of its endpoints are upgraded. If one or both of its endpoints are selected for upgrading, the weight $d(e)$ of an edge e falls down to $\alpha \cdot d(e)$ or $\alpha^2 \cdot d(e)$, respectively. It is easy to see that the model used in this thesis is a generalization of the Paik-Sahni model.

5.3 The Algorithm

In this section we provide the approximation algorithm for NODE UPGRADE CONSTRAINED FOREST. The analysis and proof of the performance will follow in the next section.

We first give a brief overview of our algorithm. A detailed description can be found in Algorithm 5.1 on the next page. Initially the set W of upgraded nodes is empty.

Our algorithm maintains the connected components of the bottleneck graph with respect to the current weight function d_W. Such a connected component C is called *active*, if $f(C) = 1$. The algorithm terminates with a feasible solution if no active component remains.

In each iteration the algorithm merges at least two active components. This is performed by upgrading some nodes in the network which actually adds new edges to the bottleneck graph. Notice that from the properties of proper functions it is not possible that there is a situation where exactly one component is active: This would interfere with the disjointness property in Definition 5.1.

The basic rule of which nodes to upgrade in an iteration is to select a set that gives the best improvement ratio. This ratio is measured by the quotient of the cost of the vertices and the decrease of the number of active components.

5.3.1 Quotient Costs

Consider the situation at the beginning of an iteration, where some components are active. By upgrading nodes, in fact we add edges to the bottleneck graph. As more and more nodes get upgraded, more and more edges are added to the bottleneck graph. At some point we obtain new paths which join some of the active components.

Input: A graph $G = (V, E)$ with a proper function $f\colon 2^V \to \{0, 1\}$,
three edge weight functions $d_0 \geq d_1 \geq d_2 \geq 0$,
a node weight function c, and a number D

1 $W \leftarrow \varnothing$
2 $G' \leftarrow \text{Bottleneck}(G, d_W, D)$
3 **while** G' contains at least one active connected component **do**
4 Assume that $\mathcal{C} = \{C_1 \ldots, C_q\}$ is the set of active components
5 **for all** $v \in V$, $C \in \mathcal{C}$ **do**
{ *Compute* $c^-(v, C)$ *and* $c^+(v, C)$ *as described in Section 5.3.2* }
6 $c^-(v, C) \leftarrow$ minimum upgrading cost to obtain a path from v to C of bottleneck delay at most D where v is not upgraded
7 $c^+(v, C) \leftarrow$ minimum upgrading cost to obtain a path from v to C of bottleneck delay at most D where v is upgrade (this cost does *not* include the upgrading cost of v)
{ *Observe that* $c^+(v, C) = c^-(v, C) = 0$ *if* $v \in C$ }
8 **end for**
9 Find a node $v \in V$ in the graph G with minimum quotient cost $q(v)$
{ *See Section 5.3.1 for definition and a description of how to accomplish this* }
Let U be the upgraded vertices on the paths from v to the clusters $C_1, \ldots, C_r$ chosen for optimal v
10 Let $W \leftarrow W \cup U$
11 Recompute the edge weights d_W,
the bottleneck graph $G' = \text{Bottleneck}(G, d_W, D)$,
and the connected components of G'
12 **end while**
Output: Upgrade set W

ALGORITHM 5.1: Node upgrading for constrained forests.

Assume that there is a distinguished node $v \in V$. Then we can count how many active components are connected to v by the current upgrade strategy. On the other hand we have the cost of the current ugprade set. The minimal ratio between these cost and the number of components joined to v is called the *quotient cost* of v. The decision for an upgrade strategy is based on minimizing the quotient cost over all nodes $v \in V$.

More formally, let $W \subseteq V$ be the current set of already upgraded nodes and $\mathcal{C} = \{C_1, \ldots, C_p\}$ be the current set of active components. Let $v \in V$ be a vertex. For technical reasons we perform a distinction of cases according to the fact whether v gets selected for upgrading in the current iteration.

We define $c^-(v, C_j)$ to be the minimum cost $c(U)$ of an upgrading set $U \subseteq V \setminus W \setminus \{v\}$ such that v and C_j are connected in the bottleneck graph. If no such upgrading set exists, we define $c^-(v, C_j) := +\infty$. Moreover, if $v \in C_j$, then $c^-(v, C_j) = 0$. Similarly, we define $c^+(v, C_j)$ to be the minimum cost $c(U \setminus \{v\})$ of an upgrading set $U \subseteq V \setminus W$ with $v \in U$ such that v and C_j are connected in the bottleneck graph. Notice that the cost $c(v)$ itself is not taken into account by the minimization. The computation of the values $c^+(v, C_j)$ and $c^-(v, C_j)$ can be performed in polynomial time, which will be shown in the next section.

With

$$q^+(v) := \min_{2 \le r \le p} \min_{\substack{\mathcal{C}' \subseteq \mathcal{C} \\ |\mathcal{C}'| = r}} \frac{c(v) + \sum_{C' \in \mathcal{C}'} c^+(v, C')}{r}$$

$$q^-(v) := \min_{2 \le r \le p} \min_{\substack{\mathcal{C}' \subseteq \mathcal{C} \\ |\mathcal{C}'| = r}} \frac{\sum_{C' \in \mathcal{C}'} c^-(v, C')}{r}$$

we are enabled to define the *quotient cost* of v by

$$q(v) := \min\{q^+(v), q^-(v)\}. \tag{5.1}$$

The quotient cost of a node can be computed easily, if all values $c^+(v, C_i)$ and $c^-(v, C_i)$ are already available. We explain how to calculate the value $q^+(v)$ efficiently. Assuming that the values $c^+(v, C_i)$ are given for the active components C_i, it is not necessary to check all $\sum_{r=2}^{p} \binom{p}{r}$ combinations for minimization. Renumber the active components such that

$$c^+(v, C_1) \le c^+(v, C_2) \le \cdots \le c^+(v, C_p).$$

Then a minimum over all r-element subsets of $\mathcal{C}$ is obviously attained by the subset $\{C_1, C_2, \ldots, C_r\}$. Hence for computing $q^+(v)$ it suffices to check p subsets

$$\{C_1, C_2, \ldots, C_r\}, \qquad \text{for } r = 1, \ldots, p.$$

The same approach works for computing the value $q^-(v)$. The additional time effort for this operation is in $O(p \log p)$ which is needed for sorting the $O(p)$ numbers.

5.3.2 Computing the Best Upgrading Paths

In this section we show that for each vertex v and active component C we can compute the values $c^-(v, C)$ and $c^+(v, C)$ in polynomial time. This is done by two single source shortest paths computations on an auxiliary graph H, where the length of a path is defined to be the node costs of the nodes on the path excluding the source vertex.

For each vertex $v \in V$ the auxiliary graph H contains two vertices v^+ and v^- representing the upgraded and the untouched version of v. The vertex set is augmented by one node for each active component. The cost of each vertex v^- is set to zero. The cost of each v^+ is set to zero for $v \in W$, and to $c(v)$ for $v \notin W$. Also, the cluster nodes have zero cost.

Informally the idea is as follows: For each edge (u, v) in G, the graph H contains the four edges with one endpoint in $\{u^+, u^-\}$ and the other endpoint in $\{v^+, v^-\}$. Therefore there is a one-to-one correspondence of paths in G and H as long as the marks "+" and "−" are ignored. By blocking edges incident with v^- in the auxiliary graph we can enforce that a path must use the corresponding node v^+ instead, which involves the upgrade cost $c(v)$. This mechanism is used to assure that critical edges are not considered as long as the necessary number of endpoints has been upgraded.

Formally, the edge set of H is determined as follows (the construction is illustrated in Figure 5.2): If $(u, v) \in E$ is uncritical, then H contains all four edges (u^+, v^+), (u^+, v^-), (u^-, v^+) and (u^-, v^-). If $(u, v) \in E$ is 1-critical, then H contains (u^+, v^+), (u^+, v^-), and (u^-, v^+). For 2-critical edge $(u, v) \in E$, graph H contains only (u^+, v^+). (Recall that we have assumed without loss of generality that there are no useless edges.) For all nodes $v \in W$, i. e., the nodes which are already upgraded, we remove the corresponding vertices v^- and incident edges from H. Finally, each active cluster C is joined to all the vertices v^+ and v^- where $v \in C$.

For a vertex v and an active cluster C let $c(v^-, C)$ and $c(v^+, C)$ denote the length of shortest paths with respect to node weights from v^- and v^+ to C in H, respectively, not including the cost of the source vertex. Thus, the cost $c(v^+, C)$ does *not* contain the cost of v.

Lemma 5.8 (Determining the Best Upgrading Set)
For each vertex $v \in V$ and each active cluster C the minimum cost $c^-(v, C)$ of an upgrading set not containing v such that the resulting bottleneck graph has a path from v to a node in C equals the node weighted distance $c(v^-, C)$ in the auxiliary graph H.

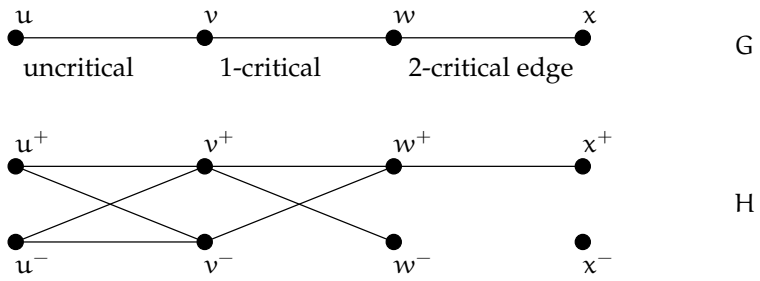

FIGURE 5.2: A graph G and the constructed auxiliary graph H.

Proof. Let S be an upgrading set of minimum cost such that the upgraded graph contains a bottleneck path $(v_0 = v, v_1, \ldots, v_t)$ from v to some node $v_t \in C$. Clearly, $S \subseteq \{v_1, \ldots, v_t\}$.

Then, the path $(v^-, v_1^{\sigma_1}, \ldots, v_t^{\sigma_t}, C)$ where we define $\sigma_i := +$ whenever $v_i \in S$ and $\sigma_i := -$ otherwise, is a path in H from v^- to C of cost at most $c(S)$. Thus, $c(v^-, C) \leq c(S)$.

We will now argue that $c(S) \leq c(v^-, C)$ proving the claim of the lemma. To this end let $(v^-, v_1^{\sigma_1}, \ldots, v_t^{\sigma_t}, C)$ be a least cost path in H. By upgrading the vertices v_i where $\sigma_i = +$, we obtain a bottleneck path in G. □

The following lemma can be proven similarly.

Lemma 5.9
For each $v \in V$ and each active cluster, $c^+(v, C) = c(v^+, C)$. □

5.3.3 Running Time

We briefly argue that our algorithm can be implemented to run in polynomial time. Notice that for a terminal set K the algorithm performs at most $|K|$ iterations. In each iteration we must solve $O(n)$ single-source shortest-path problems to compute the best upgrading paths. Each of these shortest-path trees can be computed by Dijkstra's algorithm in time $O(n \log n + m)$. For a fixed node its quotient cost can then be determined in $O(n \log n)$ time. This leads to a total time of $O(n^2 \log n)$ per iteration neglecting the time needed to update the weights and the bottleneck graph.

The latter task needs total time $O(m)$ over all iterations, since each edge weight is updated at most twice. Thus, the algorithm can be implemented to run in time $O(|K| n^2 \log n)$.

5.4 Performance Guarantee

The proof of the performance guarantee uses the notion of a spider covering, which extends the definitions given in [KR95]. Informally a spider covering is a partition of the graph into subgraphs, where the shape of each subgraph is similar to a spider. Since the structure used for computing the quotient cost is similar in appearance like a spider, we can use the existence of a spider covering in connection with an averaging argument to find a lower bound on the quotient cost of the upgrade strategy chosen in an iteration.

5.4.1 Spider Decompositions and Coverings

We first recall the definition of a spider and a spider decomposition. Although the definition is essentially taken from [KR95] we point out that the notion of a foot is used slightly different in our setting.

Definition 5.10 (Spider)
*A **spider** is a tree with at most one node of degree greater than two. A **center** of a spider is a node from which there are edge-disjoint paths to the leaves of the spider. If a spider has at least three leaves, then its center is unique. A **foot** is a leaf, and the path from the center to a non-center foot is called a **leg** of the spider. A **nontrivial spider** is a spider with at least two leaves.*

Definition 5.11 (Spider Decomposition)
*Let $G = (V, E)$ be a graph and $M \subseteq V$. A **spider decomposition of** M **in** G is a set of node-disjoint nontrivial spiders which are all subgraphs of G such that the union of the feet and the centers of the spiders contains M.*

For illustration we give an example of a spider decomposition in Figure 5.3 on the following page. Klein and Ravi proved that a spider decomposition exists almost always (with the exception of the pathological case $|V| = 1$ which per definition does not admit a nontrivial spider as a subgraph) [KR95].

Lemma 5.12 (Existence of a Spider Decomposition)
Let $G = (V, E)$ be a connected graph with $|V| > 1$, and let $M \subseteq V$ be an arbitrary nonempty subset of its nodes. Then G contains a spider decomposition of M. □

We now introduce the notion of a spider covering which will be used in the sequel. While a spider decomposition requires a collection of nodes to be overlapped by the spiders, a spider covering assumes that a collection of node subsets are covered by the spiders. An example of a spider covering is displayed in Figure 5.4.

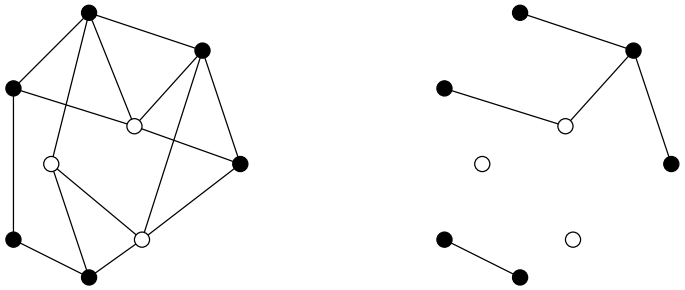

FIGURE 5.3: Example of a spider decomposition. Graph with marked nodes (left), valid spider decomposition (right).

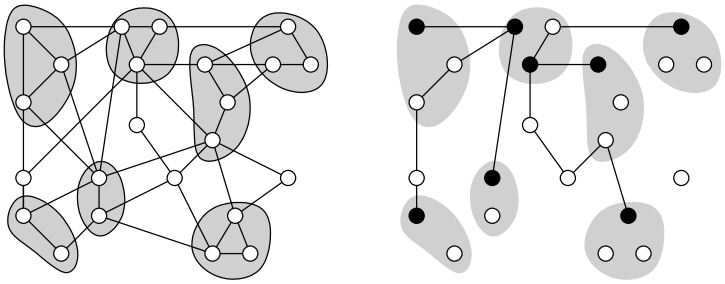

FIGURE 5.4: Example of a spider covering. Graph with marked node sets (left), valid spider covering (right).

Definition 5.13 (Spider Covering)
Let G *be a connected graph, and let there be a collection* $\{M_1, \ldots, M_p\}$ *of disjoint node sets, where each* M_i *induces a connected subgraph of* G*. A* ***spider covering of*** $\{M_1, \ldots, M_p\}$ ***in*** G *is a collection of node disjoint nontrivial spiders which are all subgraphs of* G *such that:*

1. *each set* M_i *contains a foot or a center of a spider, and*
2. *if a set* M_i *contains a foot, then* M_i *does not contain any other foot or center.*

We now show that each graph and collection of node sets admits at least one valid spider covering.

Theorem 5.14 (Existence of a Spider Covering)
Let $G = (V, E)$ *be a connected graph with* $|V| > 1$ *and* $\{M_1, \ldots, M_p\}$ *be a collection of disjoint node sets each inducing a connected subgraph in* G*. Then there is a spider covering of* $\{M_1, \ldots, M_p\}$ *in* G*.*

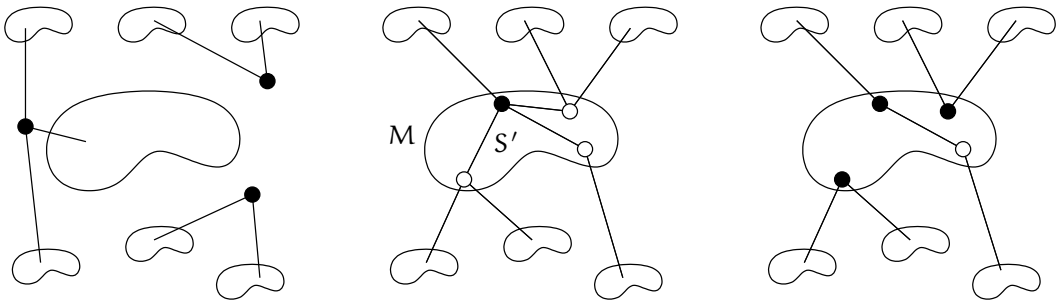

FIGURE 5.5: Illustration on the proof of Lemma 5.14. (a) No super node appearing as a body. (b) One super node as body. Solid lines represent second spider decomposition. (c) Resulting set of spiders.

Proof. If $p = 1$, the claim is trivial. So we assume that $p > 1$. Let $\tilde{G} = (\tilde{V}, \tilde{E})$ be the graph created from G by aggregating each node set M_i to a super node M_i. By Lemma 5.12 there is a spider decomposition of the set $\{M_1, \ldots, M_p\}$ in $\tilde{G}$.

All super nodes appear as feet or centers in the spider decomposition. Let us first assume that none of the super nodes is a center of a spider (see Figure 5.5–a). Then, unfold each super node M_i and choose a node $m_i \in M_i$ to connect as a foot to the corresponding leg of the spider in G. The modified spider decomposition then forms a collection of spiders with the desired properties.

Now we consider the case that there is a super node M which is the center of a spider S in the decomposition. Unfold the super node M and replace it by the corresponding subgraph. Denote by M' the set of nodes in the subgraph $G[M]$ in which the legs of spider S are rooted. Then, perform a second spider decomposition of M' in $G[M]$ (see Figure 5.5–b).

Let S' be one of the spiders of that decomposition. For each foot $m' \in M'$ of S' in which paths to two or more super nodes are rooted, disconnect m' from S', and declare m' as the body of a new spider. After this procedure, we are left over with the body of S' and a set of feet each rooting the path to exactly one super node.

If this remaining part of S' connects zero or more than one super nodes, then it can be discarded or it forms a nontrivial spider, respectively. Otherwise, the single remaining super node can be connected through edges of S' to any of the just constructed new spiders (Figure 5.5–c).

The given construction can be performed for each spider in M and again for each super node appearing as a center of a spider in the first spider decomposition. The resulting set of spiders is then a spider covering as desired. □

5.4.2 An Averaging Lemma

We now claim that in each iteration of the algorithm we choose a node whose quotient cost is bounded from above by the total upgrade cost of an optimal solution divided by the number of active components at the beginning of the iteration. This averaging result will be used in the next section to establish the result on the performance guarantee of our algorithm.

Lemma 5.15 (Estimation of Average Costs)
Let v be a node chosen in Step 9 of Algorithm 5.1 on page 127 and let $c(U)$ denote the total cost of the nodes added to the solution set W in this iteration. Let there be p active clusters before v is chosen and assume that in this iteration r clusters are merged. Then

$$\frac{c(U)}{r} \leq \frac{OPT}{p}.$$

Proof. Let W^* be an optimal upgrading set of cost $OPT := c(W^*)$ and F^* be a constrained forest of bottleneck delay at most D after the upgrade of the vertices in W^*. Let $C_1, \ldots, C_p$ be the active components at the beginning of the iteration. Notice that from the symmetry of f it follows that $p \geq 2$. Also, let W be the upgrading set constructed by the algorithm so far and $F \subset E$ be the set of edges whose delay has already been decreased to be at most D.

Assume in the first case that the graph F' consisting of the edges of $F^* \cup F$ is connected.

We now apply Lemma 5.14 to the graph F' with $M := \{C_1, \ldots, C_p\}$ to obtain a spider covering of M in F'. Let $P_1, \ldots, P_k$ be the spiders in the decomposition. We define the cost $c(P_i)$ of spider P_i to be the sum of the cost of the vertices from $W^* \setminus W$ that are contained in P_i, i. e., the cost of the vertices from the optimum solution that have not been upgraded yet. Since the spiders are node disjoint we have

$$\sum_{i=1}^{k} c(P_i) \leq c(W^*) - c(W) \leq OPT. \tag{5.2}$$

Let $M' \subset M$ denote those clusters which are not covered by the feet of the spiders. Notice that for each such cluster $C \in M'$ we have at least one spider that contains a node from M' as a center.

Denote the number of feet in spider P_i by f_i. Then by Lemma 5.14 we have

$$|M'| + \sum_{i=1}^{k} f_i \geq p. \tag{5.3}$$

We will now show the following: If v_i is the center of spider P_i and is contained in an active component C which is not covered by the feet of the spiders in the cover, then the quotient cost of v_i is at most $c(P_i)/(f_i+1)$. Otherwise we show the slightly weaker estimate that the quotient cost is bounded by $c(P_i)/f_i$.

Let $C_1, \ldots, C_{f_i}$ be the active clusters covered by the feet of the spider centered at v_i.

In the first case we have $v_i \notin W^* \setminus W$. Then, the upgraded vertices from $W^* \setminus W$ on the path from v_i to the foot covering C_j are an upgrading set resulting in a bottleneck path of delay at most D from v_i to some node in C_j. Thus, their costs are at least $c^-(v, C_j)$. Since the legs are node disjoint (except for the center v_i which by assumption does not belong to $W^* \setminus W$), we get that

$$\sum_{j=1}^{f_i} c^-(v_i, C_j) \leq c(P_i). \tag{5.4}$$

Since the quotient cost of v_i is at most $\sum_{j=1}^{f_i} c(v_i^-, C_j)/f_i$, this implies that the quotient cost of v_i is bounded from above by $c(P_i)/f_i$. Moreover, if $v_i \in C$ and C is not covered by the feet of the spider in our collection, then in particular C does not occur in the sum on the left hand side of (5.4). Since $c^-(v_i, C) = 0$ we get

$$c^-(v_i, C) + \sum_{j=1}^{f_i} c^-(v_i, C_j) \leq c(P_i)$$

and, consequently, the quotient cost of v_i is at most $c(P_i)/(f_i+1)$.

In the second case the vertex v_i is contained in $W^* \setminus W$. In this case, the upgrading vertices from $W^* \setminus W$ on the leg to C_j excluding v_i have cost at least $c^+(v_i, C_j)$. Again, by the node disjointness of the legs we get that

$$c(v_i) + \sum_{j=1}^{f_i} c^+(v_i, C_j) \leq c(P_i).$$

Since the quotient cost of v_i is also at most $c(v_i) + \sum_{j=1}^{f_i} c^+(v_i, C_j)$ divided by f_i, we obtain again that the quotient cost of v_i is at most $c(P_i)/f_i$.

The same arguments as above show that if $v_i \in C$ and C is not covered by the feet of the spiders then the quotient cost of v_i can be bounded by $c(P_i)/(f_i+1)$.

Let v be the node chosen in Step 9 in the current iteration. Then the quotient cost $q(v)$ of v satisfies $q(v) \leq q(v_i)$ for $i = 1, \ldots, k$. Thus we get

$$q(v^*) \cdot f_i' \leq c(P_i), \quad \text{for } i = 1, \ldots, k, \tag{5.5}$$

where $f'_i \in \{f_i, f_i + 1\}$ is chosen as above such that the quotient cost of the center v_i of spider P_i is bounded by $c(P_i)/f'_i$.

Summing up the inequalities in (5.5) and using (5.2) and (5.3) the claim of the lemma follows in this case.

It remains to consider the case that the graph F' consisting of the edges from $F^* \cup F$ is not connected. Notice that if we show that each connected component of F' contains either none or at least two active clusters, we can apply our arguments from above to each of the connected components and the claim of the lemma will follow by summing up over those components that contain active clusters.

Let $C_1, \ldots, C_p$ be the active components and $Z_1, \ldots, Z_t$ be the inactive components at the beginning of the iteration. Notice that each connected component of F' is the disjoint union of some components C_i and Z_j. Assume for the sake of a contradiction that component Z of F' contains exactly one active cluster, say C_1. As noted above, F' can be written as the disjoint union of the connected components at the beginning of the current iteration, so $F' = C_1 \cup Z_1 \cup \cdots \cup Z_{t'}$ for some inactive components Z_j.

Clearly $f(Z) = 0$, since otherwise one connected component of F^* (and thus of F') would contain vertices from Z as well as from $V \setminus Z$ which is not possible. Moreover, $f(Z_j) = 0$, by definition of an inactive component. By the disjointness of the Z_j we have for $B := Z_1 \cup \cdots \cup Z_{t'}$ that $f(B) = 0$. Now applying Lemma 5.6 for $U := Z$ and B as defined above yields that $f(C) = 0$ which contradicts the fact that C was an active component. □

5.4.3 Potential Function Argument

We can now use a proof technique based on a potential function [KR95] to prove the main result on the performance of our approximation algorithm.

Theorem 5.16 (Approximability of NODE UPGRADE CONSTRAINED FOREST) *Algorithm 5.1 on page 127 is an approximation algorithm for* NODE UPGRADE CONSTRAINED FOREST *with a performance guarantee of* $(2\ln(\sqrt{e}/2 \cdot |K|), 1)$. *Here*

$$K := \{ v \mid f(\{v\}) = 1 \}$$

is the set of terminals given by the proper function f.

Proof. Let the algorithm use l iterations. Notice that $l \leq n$. We let the potential function φ_j denote the number of active components at the end of iteration j.

Then $\varphi_{l-1} \geq 2$ since the algorithm does not terminate before iteration l and $\varphi_l = 0$. Now let r_j denote the number of active components merged during iteration j, and let c_j be the cost spent in that iteration. Then, the statement of Lemma 5.15 reads

$$r_j \geq \frac{c_j \cdot \varphi_{j-1}}{\mathrm{OPT}}. \tag{5.6}$$

Hence,

$$\varphi_j \leq \varphi_{j-1} - (r_j - 1) \leq \varphi_{j-1} - \frac{1}{2} r_j \overset{(5.6)}{\leq} \varphi_{j-1} \left(1 - \frac{c_j}{2\,\mathrm{OPT}}\right).$$

Solving this recurrence yields

$$\varphi_{l-1} \leq \varphi_0 \prod_{j=1}^{l-1} \left(1 - \frac{c_j}{2\,\mathrm{OPT}}\right).$$

Taking natural logarithms and using the estimate $\ln(1-\tau) \leq -\tau$, we get

$$\sum_{j=1}^{l-1} c_j \leq 2\,\mathrm{OPT} \cdot \ln \frac{\varphi_0}{\varphi_{l-1}} \leq 2\,\mathrm{OPT} \cdot \ln(|K|/2) \tag{5.7}$$

as a bound for the cost of the upgraded vertices in all but the last iteration. Also, by Lemma 5.15 the cost of the vertices upgraded in the last iteration is at most

$$\frac{\mathrm{OPT}}{\varphi_{l-1} - \varphi_l} \cdot \varphi_{l-1} = \mathrm{OPT} \tag{5.8}$$

(recall that $\varphi_l = 0$ per definition of l). Summing up the right hand sides of (5.7) and (5.8) provides an upper bound on the total cost which is

$$\mathrm{OPT} \cdot \left(2\ln(|K|/2) + 1\right) = \mathrm{OPT} \cdot \left(2\ln\left(\sqrt{e}/2 \cdot |K|\right)\right)$$

as claimed. □

5.5 Hardness Results

This section contains our hardness results for the node upgrading problem under study.

Theorem 5.17 (Hardness of NODE UPGRADE CONSTRAINED FOREST)
Let $\varepsilon > 0$ be arbitrary and the proper function f specify Steiner trees. Unless NP $\subseteq$ DTIME$(N^{O(\log\log N)})$, there is no approximation algorithm for NODE UPGRADE CONSTRAINED FOREST *(with proper function f) with performance $((1-\varepsilon)\ln|K|, 1)$, where K is the set of terminals. This result continues to hold even if $c(v) = 1$ for all vertices $v \in V$.*

Proof. We perform a reduction from MINIMUM SET COVER. Given an instance with element set Q and subset collection $R \subseteq 2^Q$, we set up a bipartite graph with node set $Q \cup R$. For $Q' \in R$ and $q \in Q'$, we add an edge e between node q and Q' of weight $d_0(e) = 2$ and $d_1(e) = d_2(e) = 1$. We add a root node connected to all set nodes through edges e of weight $d_0(e) = d_1(e) = d_2(e) = 1$. The proper function f is chosen to reflect a Steiner tree with terminal set Q. The bottleneck constraint is chosen to be 1. All nodes have equal upgrade cost 1.

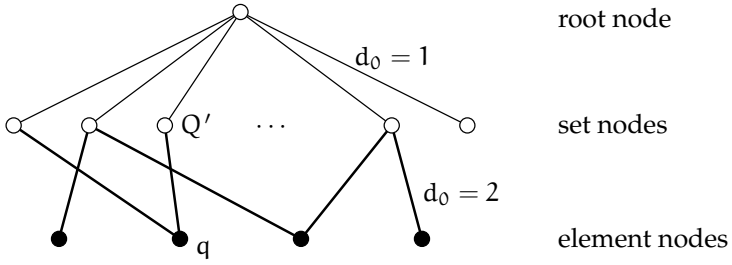

FIGURE 5.6: Illustration on the reduction from MINIMUM SET COVER.

It is easy to see that a set cover of some size implies a valid upgrade set of the same cost. Conversely, notice that there is no advantage in prefering the root node or element nodes for upgrading rather than adjacent set nodes. Hence we can assume that a minimum cost upgrade set consists only of set nodes. Consequently, it implies a set cover of size equal the upgrade costs.

Since the reduction is approximation preserving, we can apply the non-approximability result for MINIMUM SET COVER (see Theorem 1.6 on page 17) and the claim follows. □

The preceding result was formulated with a performance guarantee of 1 for the bottleneck delay. Indeed, the result holds even if we allow the contraint on the bottleneck delay to be violated by any factor $f(n)$, where f is an arbitrary polynomial time computable function. This follows immediately from the proof of Theorem 5.17 where the definition $d_0(e) := 2$ is replaced by $d_0(e) := f(n) + 1$.

Corollary 5.18 (Hardness II)
For any polynomial time computable function $f(n)$, the hardness result from Theorem 5.17 continues to hold if performance guarantee $((1 - \varepsilon) \ln |K|, 1)$ is weakened to $((1 - \varepsilon) \ln |K|, f(|V|))$. □

5.6 Concluding Remarks

We have argued that NODE UPGRADE CONSTRAINED FOREST is a generalization of NODE WEIGHTED STEINER TREE. Moreover, we have justified the ap-

proximability of NODE UPGRADE CONSTRAINED FOREST by proving the following bounds:

lower bound	$(\ln \lvert K\rvert, 1)$	(*Theorem 5.17*)
	$(\ln \lvert K\rvert, f(\lvert V\rvert))$	(*Corollary 5.18*)
upper bound	$(2\ln(\sqrt{e}/2 \cdot \lvert K\rvert), 1)$	(*Theorem 5.16*)

Here, K is the set of terminals induced by a proper function specifying Steiner trees, and f is any polynomial time computable function. The lower bound is valid under the assumption $NP \not\subseteq DTIME(N^{O(\log\log N)})$.

Chapter 6

Arc Upgrade Problems

The concept of a *flow* in a graph is a very natural way to model transportation and logistics problems where a special focus lies on the simultaneous flow of multiple goods. To this end, goods which are to be transported through the network are represented by items of unit size. The flow properties (see definition on page 5) guarantee that the flow is conserved and there are no sources or sinks where goods are created or vanish.

Flows can be formulated and investigated on undirected graphs as well as on directed graphs. We are going to assume that all graphs are *directed* in this chapter. Nevertheless, most of the results are also valid on undirected graphs. This is mainly a consequence of the fact that the graph structure which is most relevant in this chapter, namely the class of series-parallel graphs, can be handled with the same effort both on directed and undirected graphs.

In the flow model all items are treated equally, i. e., one can not distinguish between the several items. Problems which involve different types of goods—where each type has its own demands and constraints—-are known as *multi commodity* flow problems in literature. This kind of problems are not investigated in this thesis. We refer the reader e. g. to [GCF99] for a survey on multi commodity flows.

When looking at practical applications one observes that often there is for each link an upper bound on the amount of flow which can be transported along this link in one unit of time. This fact is incorporated into the model by furnishing the arcs with numbers which represent the upper *capacity* of the particular arc.

More complex applications come up with questions on the *cost* of transportation of some goods through the network. For this sake one introduces a cost function for each arc. The cost function maps the amount of flow to the transportation cost emerging from the flow. There are several types of cost functions whose properties are compiled in Section 6.1.1.

Capacities and costs as described above are *statically* given numbers. A main focus of this chapter lies on problems under a model where those numbers are subject to change. We use an *arc upgrade model*. The upgrade of a particular arc is performed by investing a budget on that arc. As a result, the available capacity increases, or the cost per unit of flow decreases, or both happens at the same time.

6.1 Preliminaries and Problem Formulation

Throughout this chapter we assume that the underlying graph is a *directed* graph.

6.1.1 Flow Cost Functions

For cost flow problems each arc $r \in R$ is furnished with a cost function

$$c_r \colon t \mapsto c_r(t) \,.$$

The value $c_r(t)$ of the cost function on arc r specifies the transportation cost emerging from the transportation of t units of goods along arc r. Thus, if $x \colon R \to \mathbb{R}_0^+$ is a flow then the total cost of flow x is given by

$$\sum_{r \in R} c_r(x(r)) \,.$$

We make two assumptions on cost functions:

1. $c_r(0) = 0$
2. c_r is non-decreasing

The first assumption is made without loss of generality: If it is not satisfied, we replace cost function c_r by the function c'_r defined by $c'_r(t) := c_r(t) - c_r(0)$. Since cost $c_r(0)$ incurs anyway, we can subtract it from the available flow cost budget a priori. Albeit the second assumption may be considered as a restriction, we remark that it is strongly motivated from practical applications.

We give a brief overview on several special types of cost functions which are common in literature and also utilized in this chapter (confer Figure 6.1): The first type of cost function is defined by

$$c_r(t) := \begin{cases} 0, & \text{if } t = 0, \\ c_r, & \text{if } t > 0 \end{cases}$$

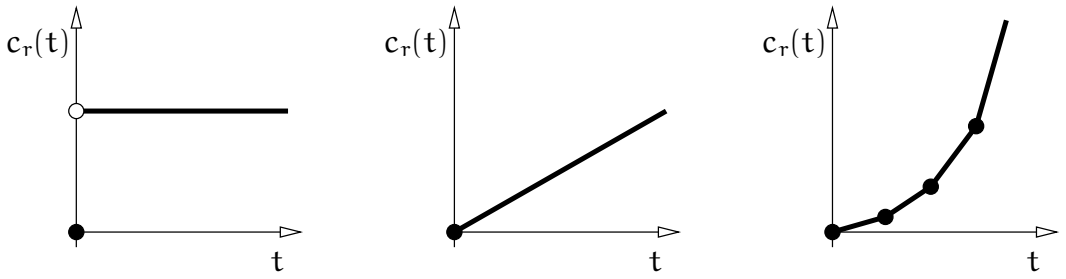

FIGURE 6.1: Typical cost functions: fixed charge (left), linear (center), piecewise linear convex (right).

for a constant $c_r \in \mathbb{R}^+$ and called *fixed charge* arc cost function. A cost function of this type is a good approximation to the case where the costs for setting up a link dominate the costs for the actual transportation. In such a scenario one is charged the activation costs of a link whenever there is a positive flow on the link regardless of the actual amount of flow.

Another well-motivated type of cost functions are *linear* transportation costs which are defined by

$$c_r(t) := c_r \cdot t$$

for a constant $c_r \in \mathbb{R}^+$. Functions of this type deal with the fact that several costs are directly proportional to the amount of flow on an arc: Here one may consider a transportation scenario where cargo is measured in units of trucks. If c_r is the cost needed for moving one truck along the road, then the resulting transportation costs are $c_r t$ as a first approximation.

More complicated cost functions include combinations of functions of the above mentioned types. Of particular interest are continuous functions which are *piecewise linear* and *convex*. These can be decomposed into a finite collection of linear parts with increasing slope. Cost of this type can arise in networks with constrained resources: as long as the supply exceeds the demand, the costs will be low. On the other hand, if the workflow is near the limit of the network (or beyond that, i. e., the demand exceeds the supply), the costs will increase disproportionally.

6.1.2 Improvement of Capacity

Upgrade Model

In this section we describe the first arc upgrade model applied in this chapter. It is assumed that a capacitated graph is given. The arc capacities are subject to change: The model admits to distribute a given budget among the arcs in order to increase the capacity of several arcs.

From the practical point of view the capacities can not be increased to infinity in realistic applications. This is reflected in the model by the fact that each arc $r \in R$ is furnished with two numbers, $u_r, U_r \in \mathbb{Q}_0^+$, where $U_r \geq u_r$ for each $r \in R$. Number u_r denotes the *initial capacity* of arc r. This capacity may be increased up to U_r, which is the *maximal capacity* beyond which no further upgrade is allowed.

The actual amount of upgrade is notated by a function $i\colon R \to \mathbb{Q}_0^+$, which is called the *upgrade strategy* function. For each arc $r \in R$, the value $i(r)$ denotes the amount of capacity increase on that arc, i. e., the resulting capacity is given by $u_r + i(r)$. Notice that a *valid* upgrade strategy must satisfy

$$u_r + i(r) \leq U_r$$

for each arc $r \in R$.

Although there may be an arbitrary relation between the amount of budget invested on an arc and the resulting increase of capacity, we restrict this relation to *linear* functions. Therefore the relation can be described by one numerical parameter per arc: We use $b_r \geq 0$ to denote the slope of the upgrade cost function for arc r.

The budget consumed by a valid upgrade strategy i is denoted by $B(i)$. It is computed by summing up the spent budget over all arcs of the graph. Thus,

$$B(i) := \sum_{r \in R} b_r \cdot i(r)$$

is the total amount of budget used for upgrade strategy i.

Discussion of Boundary Conditions

At first we point out that one can assume without loss of generality that $b_r > 0$ holds for all arcs $r \in R$: Assume there was an arc r with zero upgrade cost, then the situation does not change when this arc is upgraded to its maximal capacity U_r before starting any further task.

A second remark is on applications where from the outset there is no bound on the amount of upgrading the capacities. This would imply that $U_r = \infty$ on

some arcs. Usually we have $b_r > 0$ for arcs of this type. Therefore the available upgrade budget is actually a limiting factor. If B denotes the total budget, it is sufficient to set $U_r := u_r + B/b_r$ to handle this unlimited case.

If there is a graph where only a (proper) subset of the arcs is eligible for upgrading, one can choose $U_r := u_r$ for the remaining arcs and thus prevent to upgrade those fixed capacity arcs.

We do not prohibit the case $u_r = 0$. This means that an arc is actually not present in the initial graph, but it may be included by the upgrade procedure.

We distinguish further according to the granularity of upgrade strategies. A *continuous* upgrade strategy allows function i to take on any real (or any rational) value. An *integer* strategy restricts the values to integer numbers. Last, a *zero-one* strategy requires value $i(r)$ to be chosen from the two element set $\{0, U_r - u_r\}$, i. e., for each arc one can only decide between upgrading the arc in full or leaving it in the initial state.

Problem Formulation

We formulate now the first flow problem investigated in this chapter. The problem utilizes the arc upgrade model described above.

Definition 6.1 (MAXFL-CPI Problem)
An instance of MAXFL-CPI *specifies:*

- *a graph* $G = (V, R)$ *with source* $s \in V$ *and sink* $t \in V$*,*
- *nonnegative arc function* u_r *(initial capacities),*
- *arc function* $U_r \geq u_r$ *(maximal capacities),*
- *positive arc function* b_r *(upgrade costs),*
- *an integer* B *(budget).*

The goal is to find a valid improvement strategy $i\colon E \to \mathbb{Q}_0^+$ *of bounded total cost* $\sum_{r\in R} b_r \cdot i(r) \leq B$*, such that the flow from* s *to* t *in the upgraded graph is maximized.*

The canonical notation of this two-criteria optimization problem is

$$(\text{MAX: FLOW, CAPACITY IMPROVEMENT COST})\,.$$

The dual of the problem which is generated by interchanging the two optimization objectives is denoted by MINCPI-FL and has the canonical notation

$$(\text{MIN: CAPACITY IMPROVEMENT COST, FLOW VALUE})\,.$$

It is defined as follows:

Definition 6.2 (MINCPI-FL Problem)
An instance of MINCPI-FL *specifies:*

- *a graph* $G = (V, R)$ *with source* $s \in V$ *and sink* $t \in V$*,*
- *nonnegative arc function* u_r *(initial capacities),*
- *arc function* $U_r \geq u_r$ *(maximal capacities),*
- *positive arc function* b_r *(upgrade costs),*
- *an integer* F *(flow value).*

The goal is to find a valid improvement strategy $i\colon E \to \mathbb{Q}_0^+$ *of minimum total cost* $\sum_{r\in R} b_r \cdot i(r)$*, such that the flow from* s *to* t *in the upgraded graph is at least* F*.*

We further distinguish the problem MAXFL-CPI according to granularity of the improvement strategy as described above. Problem variants CONTINUOUS-MAXFL-CPI and INTEGER-MAXFL-CPI (continuous and integer upgrade strategy) turn out to be solvable in polynomial time which is written down in Section 6.3. Problem 0/1-MAXFL-CPI under the zero-one upgrade strategy turns out to be intractable. This variant is investigated in Section 6.4.

Fixed Arc Charge

The following pair of dual problems is from the area of network design problems, i. e., they are no upgrade problems. Nevertheless it will turn out that they are closely related to the above stated problem. The problems are formulated using a fixed arc cost function. This means that an arc is charged a fixed budget as soon as there is a positive flow on it.

Definition 6.3 (MAXFL-ARCWT Problem)
An instance of MAXFL-ARCWT *specifies:*

- *a graph* $G = (V, R)$ *with source* $s \in V$ *and sink* $t \in V$*,*
- *nonnegative arc function* u_r *(capacities),*
- *nonnegative arc function* c_r *(costs),*
- *an integer* B *(budget).*

The goal is to determine an arc subset $A \subseteq R$ *of total cost* $\sum_{r\in A} c_r \leq B$*, such that in* $G[A]$ *the flow from the source* s *to the sink* t *is maximized.*

The canonical notation of MAXFL-ARCWT is

(MAX: FLOW, TOTAL ARC WEIGHT).

The dual problem, denoted by MINARCWT-FL, and with canonical notation

(MIN: TOTAL ARC WEIGHT, FLOW VALUE),

is defined as follows:

Definition 6.4 (MINARCWT-FL Problem)
An instance of MINARCWT-FL *specifies:*

- *a graph* $G = (V, R)$ *with source* $s \in V$ *and sink* $t \in V$,
- *nonnegative arc function* u_r *(capacities),*
- *nonnegative arc function* c_r *(costs),*
- *an integer* F *(flow value).*

The goal is to find a minimum cost subset $A \subseteq E$ *of the arcs such that the flow in* $G[A]$ *from the source* s *to the sink* t *is at least* F.

As stated before, the above problems are actually not arc upgrade problems. The reason why they are included here is that they are equivalent to arc upgrade problems under a zero-one upgrade strategy as shown below. Moreover, for the sake of clarity we use the fixed cost variants to design the approximation algorithm in Section 6.4.

The equivalence between 0/1-MINCPI-FL and MINARCWT-FL is easy to see. To reduce 0/1-MINCPI-FL to problem MINARCWT-FL, we use the following idea [AMO93]: Each arc $r \in R$ with initial capacity u_r, maximal capacity U_r and upgrade cost b_r is replaced by two parallel arcs. The first of them has capacity u_r and zero flow cost. The second supplies the remaining capacity $U_r - c_r$. Its fixed flow cost is set to $b_r \cdot (U_r - u_r)$.

We describe the reduction in the reverse direction: Each arc with zero flow cost is included in the instance of 0/1-MINCPI-FL with retaining its initial capacity and preventing any upgrade possibility. If an arc r has positive flow cost b_r and capacity u, we include the arc in the instance by setting its initial capacity $u_r := 0$, the maximum capacity to u, and the upgrade cost to b_r/u.

Since a zero-one upgrade strategy is employed here, the decision for *upgrading* an arc in 0/1-MINCPI-FL is equivalent to the decision for *using* the corresponding arc in MINARCWT-FL. Obviously the construction can be performed in polynomial time. Thus the stated equivalence is immediate.

Theorem 6.5 (Equivalence of Problems)
0/1-MINCPI-FL *is equivalent to* MINARCWT-FL. □

6.1.3 Improvement of Unit Flow Cost

Upgrade Model

In the sequel we describe the second upgrade model utilized by problems investigated in the current chapter. In this model there is a graph with arc capacities and unit flow costs as usually used for modeling cost flow problems. In addition, the unit flow cost on each arc is subject to change: By investing (a part of) the given budget on an arc, the unit flow costs can be decreased.

The parameters of the model are justified by the following values for each arc r: Value $c(r)$ is the *initial cost* for a unit of flow. Value $p(r)$ specifies the *price* for purchasing an "upgrade unit", and value $d(r)$ specifies the *discount* of each upgrade unit, i. e., the amount of decrease of unit flow costs per upgrade unit. As usual, parameter $u(r)$ specifies the capacity of the arc which is fixed in the current model.

An upgrade strategy is described by an integer function y on the arcs. A value $y(r) > 0$ denotes the fact that the arc r is upgraded by $y(r)$ upgrade units. Assume that there is a flow of value x on arc r. The effect of upgrading the arc by $y(r)$ units is, that the flow costs reduce from initially $c(r) \cdot x$ to

$$\big(c(r) - y(r)d(r)\big) \cdot x$$

after the upgrade. The cost of upgrading this arc is given by $y(r)p(r)$. An additional arc function c_{min} specifies a lower bound on the flow costs to deal with the fact that in realistic applications it is unlikely that unit flow costs can be made to vanish.

Discussion of Boundary Conditions

We briefly discuss some assumptions on the parameters and argue that they can be made without loss of generality.

- At first we assume $u(r) > 0$ for all arcs r. This is permitted since arc capacities can not be changed in the model and an arc with capacity zero is useless.
- We also assume $c(r) > 0$. The case $c(r) = 0$ means that the flow cost of an arc vanishes. This can be modeled by setting $c(r) := 1$, $d(r) := 1$ and $p(r) := 0$ which means that the initial non-zero unit flow cost can be reduced to zero for free.

- Next we assume $c(r) > c_{\min}(r)$: equality would mean that the flow cost is not subject to change. This can be handled by choosing a very large price $p(r)$ which exceeds the given budget.
- Assumption $d(r) > 0$ on the discount can be supposed by the same arguments.
- Last we assume that $d(r) \leq c(r) - c_{\min}(r)$. This can be achieved by assigning $d(r)$ to $\max\{d(r), c(r) - c_{\min}(r)\}$ since a reduction of the unit flow cost below $c_{\min}(r)$ is not allowed anyway.

We further assume that all parameters are integer numbers. This can be achieved by multiplying rational numbers by a suitable common denominator. As noted in the introduction in the discussion of the complexity of computations (Section 1.2 on page 5), this normalizing is permissible without affecting the estimations on the running time.

For simplicity we would like to have another step of normalizing which would make one of the parameters p and d to be equal 1. Unfortunately it is not possible to guarantee this normalizing and the integer assumption at the same time, so we do not make further assumptions on the price and discount values.

Problem Formulation

We now describe the problem formulated under the current arc upgrade model. It is best described as a problem where the flow value is maximized while the costs shall remain below given limits.

Notice that we must now distinguish carefully between the budget available for upgrading the arcs and the budget from which we pay the actual flow in the upgraded network. A solution $\sigma = (x, y)$ specifies a flow x and an upgrade strategy y on the graph. By

$$B(\sigma) := \sum_{r \in R} y(r)p(r)$$

we denote the total upgrade cost needed for the upgrade strategy, while

$$C(\sigma) := \sum_{r \in R} \big(c(r) - y(r)d(r)\big)x(r)$$

denotes the resulting flow cost in the network. With $F(\sigma)$ we term the net flow through the network from s to t.

With these definitions we are ready to formulate the next problem:

Definition 6.6 (MAXFL-COI-FC Problem)
An instance of MAXFL-COI-FC *specifies:*

- *a graph* $G = (V, R)$ *with source* $s \in V$ *and sink* $t \in V$,
- *positive arc function* u *(capacities),*
- *nonnegative arc functions* c *and* c_{min} *with* $c > c_{min} \geq 0$ *(costs),*
- *nonnegative arc function* p *(prices),*
- *positive arc function* d *(discount),*
- *integers* B *(budget) and* C *(flow budget)*

The goal is to find a solution $\sigma = (x, y)$*, specified by a feasible flow* x *and an upgrade strategy* y *on the arcs, such that*

1. $B(\sigma) \leq B$
2. $C(\sigma) \leq C$
3. $F(\sigma)$ *is maximized.*

The canonical notation of this problem is

$$(\text{MAX: FLOW, COST IMPROVEMENT, TOTAL FLOW COST}) .$$

6.1.4 Improvement of Both Capacity and Unit Flow Cost

Upgrade Model

The two upgrade models described so far suggest to set up a combined model where both arc capacities and unit flow costs can be improved by investing the upgrade budget on the graph. In this section we describe one type of such a combined model.

As before, the *upgrade in unit flow costs* is performed by purchasing upgrade units. The effect of an upgrade unit is described by the *discount* d. Its *price* is denoted by p_d. The first part of the upgrade strategy is denoted by value y_d.

Similarly, the *improvement of capacity* is performed in upgrade units: one upgrade unit increases the capacity of the arc by the *augmentation* a and incurs the *price* p_a. The amount of upgrade units for this part of the upgrade strategy is denoted by y_a. A value u_{max} denotes the maximal capacity beyond which no further upgrade is allowed.

We illustrate this model with an example. Assume that there is a flow of value x on arc r. The effect of upgrading the arc by $y_a(r) + y_d(r)$ units is, that for each arc $r \in R$ the flow costs reduce from initially $c(r) \cdot x$ to

$$\big(c(r) - y(r)d(r)\big) \cdot x$$

while the arc capacity increases from initially $u(r)$ to

$$u(r) + a(r) \cdot y_a(r)$$

after the upgrade. The term

$$y_a(r)p_a(r) + y_d(r)p_d(r)$$

specifies the cost of upgrading this particular arc.

This model makes no qualitative distinction between capacity and flow cost improvement. It uses only one combined budget for both kinds of upgrade.

Further it is presumed that the two types of upgrade do not interfere with each other. This means, that the model assumes that the effort for lowering the unit flow costs is independent from the capacity improvement on the particular arc. This may be considered as a unnatural restriction but it helps to keep the model smooth enough to be capable within the framework.

Problem Formulation

We are now ready to describe the problem which is formulated under the combined arc upgrade model. Similar to the problem suggested in the previous section, the goal is to maximize the flow while the upgrade costs and flow costs are limited.

A solution $\sigma = (x, y_a, y_d)$ of the problem is given by three arc functions, where x denotes the flow, y_a the improvement of capacity and y_d the improvement of unit flow costs. The total upgrade costs needed by σ are given by

$$B(\sigma) := \sum_{r \in R} y_a(r)p_a(r) + y_d(r)p_d(r)\,.$$

The resulting flow costs are

$$C(\sigma) := \sum_{r \in R} \big(c(r) - y_d(r)d(r)\big)x(r)$$

The flow is valid, if the capacity constraints

$$0 \leq x(r) \leq u(r) + y_a(r)a(r) \leq u_{max}(r)$$

hold for each arc $r \in R$ in the network. As before, with $F(\sigma)$ we term the net flow through the network from s to t.

With these definitions we are ready to formulate the next problem:

Definition 6.7 (MAXFL-COCPI-FC Problem)
An instance of MAXFL-COCPI-FC *specifies:*

- *a graph* $G = (V, R)$ *with source* $s \in V$ *and sink* $t \in V$,
- *positive arc function* u *and* u_{max} *with* $u_{max} > u$ *(capacities),*
- *nonnegative arc functions* c *and* c_{min} *with* $c > c_{min} \geq 0$ *(costs),*
- *nonnegative arc function* p_d *(prices for cost improvement),*
- *positive arc function* d *(discount),*
- *nonnegative arc function* p_a *(prices for capacity augmentation),*
- *positive arc function* a *(augmentation),*
- *integers* B *(upgrade budget) and* C *(flow cost budget)*

The goal is to find a solution $\sigma = (x, y_a, y_d)$*, specified by a feasible flow* x*, a capacity upgrade strategy* y_a *and a cost upgrade strategy* y_d *on the arcs, such that*

1. $B(\sigma) \leq B$
2. $C(\sigma) \leq C$
3. $F(\sigma)$ *is maximized.*

The canonical notation of this problem is

(MAX: FLOW, COST AND CAPACITY IMPROVEMENT, TOTAL FLOW COST) .

6.2 Related Work

Series-parallel graphs are a special case of decomposable graphs [BLW87]. Initially series-parallel graphs were motivated from electrical networks consisting of resistors [Duf65]. If the terminals are not pre-determined, then the class of undirected series-parallel graphs includes the class of outer-planar graphs [Duf65].

A series-parallel graph can be described by a *decomposition tree* which is of size linear in the number of edges or arcs. Algorithms for recognizing series-parallel networks and constructing a decomposition tree in linear time (with respect to the size of the edge or arc set) have been described in [VTL82, Sch95].

In [Woe99] there is introduced the notion of *benevolent* problems which describes a further characterized class of problems which are solvable by a dynamic programming approach. It is shown that there exists an FPAS for all benevolent problems.

There are other applications of edge upgrade models in literature. In [KN+98] the authors use a model where edge weights can be reduced by the upgrade. They give among other results approximation algorithms for the problem of minimizing the weight of a spanning tree (while the available budget is constrained) with a performance of $(1+\varepsilon, 1+2/\varepsilon)$. A related problem where the diameter of the spanning tree is to be minimized is approximated with performance $(O(\log n), O(\log n))$.

The problem of minimizing the bottleneck delay of a spanning tree in the upgraded graph is solvable in polynomial time [KM+98a]. An algebraic framework which generalizes budget constrained bottleneck capacity upgrade problems and provides polynomial time algorithms has been developed in [BKZ00].

The problem of minimizing the shortest path tree weight of a given tree by decreasing the edge weights has been shown to be polynomially solvable in [Ber92].

An opposite upgrade model has been considered e. g. in [FSO96]. Here the model admits edge weights to be increased with linear costs. The authors give a polynomial time algorithm to the problem of finding an upgrade strategy such that the weight of a minimum spanning tree in the resulting graph is maximized.

A problem where upgrading an edge means decreasing its capacity is the *network inhibition* problem [Phi93]. Here the goal is to find an upgrade strategy such that the total flow through the upgraded graph is minimized. There is an FPAS on planar graphs [Phi93] as well as an approximation algorithm on general graphs [KM+00].

6.3 Solving Capacity Improvement Problems

In this section we describe those variants of MAXFL-CPI which can be solved optimally in polynomial time.

6.3.1 Continuous Upgrade Strategy

Problem CONTINUOUS-MAXFL-CPI can be solved optimally in polynomial time with a linear programming approach. The linear program depicted in Figure 6.2 on the following page is a formulation of CONTINUOUS-MAXFL-CPI. It is an easy extension of the well known linear programming formulation for the MAXIMUM FLOW problem.

Theorem 6.8 (Solution of CONTINUOUS-MAXFL-CPI)
CONTINUOUS-MAXFL-CPI *can be solved in polynomial time.* □

$$
\begin{aligned}
&\text{maximize } F \\
&\text{subject to} \\
&\sum_{w:(s,w)\in R} x(s,w) - \sum_{u:(u,s)\in R} x(u,s) = +F \\
&\sum_{w:(v,w)\in R} x(v,w) - \sum_{(u,v)\in R} x(u,v) = 0 \qquad \text{for all } v \in V \setminus \{s,t\} \\
&\sum_{w:(t,w)\in R} x(t,w) - \sum_{u:(u,t)\in R} x(u,t) = -F \\
&0 \le x(r) \le u_r + i(r) \qquad \text{for all } r \in R \\
&0 \le u_r + i(r) \le U_r \qquad \text{for all } r \in R \\
&\sum_{r\in R} b_r \cdot i(r) \le B\,.
\end{aligned}
$$

FIGURE 6.2: Linear programming formulation of CONTINUOUS-MAXFL-CPI.

Notice that this approach does not work for integral or zero-one upgrade strategies, since with these additional restrictions one cannot expect that the resulting linear program can be solved in polynomial time.

6.3.2 Integer Upgrade Strategy

The main contribution of this section is to show that MAXFL-CPI can also be solved optimally in polynomial time even if we require the improvement strategy to be integral.

The main idea is a transformation of INTEGER-MAXFL-CPI into a minimum cost flow problem with additional budget constraint. Subsequently we give an algorithm to the latter problem.

Consider an instance of INTEGER-MAXFL-CPI. Assume that i^* is an optimal improvement strategy and x^* is a corresponding maximal flow. Since the upgrade strategy does not waste money, each arc is not upgraded unless the flow on it exceeds the initial capacity. This means, we have $i^*(r) = \max\{0, x^*(r) - u_r\}$ for each arc r.

This behavior can be modeled by a flow cost function c_r defined as follows. As long as one sends flow along an arc r of value at most u_r, there are no costs. If one wants to send more flow along this arc, one has to pay b_r units of money for

each unit of flow exceeding the initial capacity u_r. This results in the following flow cost function c_r:

$$c_r(x) = \begin{cases} 0 & \text{if } 0 \le x \le u_r \\ b_r \cdot (x - u_r) & \text{if } u_r < x \le U_r \end{cases}$$

Notice that the flow cost functions are piecewise-linear and convex.

Consider a graph with flow cost functions as described before and upper capacities U_r. It is easy to see that a minimum cost flow of flow value F and flow cost B corresponds to a flow of the same flow value and upgrade cost B: simply use for each arc the same flow value in both instances.

To apply well known algorithms for the minimum cost flow problem, the cost functions must be linear. This can easily be achieved by a method described in [AMO93]: each arc $r \in R$ is replaced by two parallel arcs r_0 and r_1. Arc r_0 has capacity u_r and zero flow cost, while arc r_1 gets the remaining capacity $U_r - u_r$ and flow cost b_r. The validity of the transformation follows easily from the convexity of the flow cost functions c_r.

To solve the budget constrained maximum flow problem one seeks the maximal flow value F^* which can be achieved with a flow of cost B. This can be determined by a binary search for F^* on the integer interval $I := [0, F_{max}]$, where F_{max} is an upper bound on the total flow value. A suitable value for this maximal value is $F_{max} := n \cdot U_{max}$, where $U_{max} := \max_{r \in R} u_r$ is the maximal capacity of an arc.

For a test value $F \in I$ one computes a solution to the classical minimum cost flow problem using well known efficient algorithms (see [AMO93]). After that, one can decide whether the resulting flow cost exceeds the budget B and continue with the binary search.

The above discussion shows that problem INTEGER-MAXFL-CPI (on a graph with m arcs) can be reduced to a budget constrained minimum cost flow problem (on a graph with at most $2m$ arcs). This problem in turn can be solved by $O(\log(nU_{max}))$ minimum cost flow computations.

Theorem 6.9 (Solution of INTEGER-MAXFL-CPI)
INTEGER-MAXFL-CPI *can be solved in polynomial time. This can be achieved by* $O(\log(nU_{max}))$ *minimum cost flow computations on a graph which consists of* $2m$ *arcs.* □

Approximate Solution

The algorithm for solving the budget constrained minimum cost flow problem described in the previous section can be speeded up for the price of accuracy

by a general technique. Instead of performing a binary search on the interval $[0, nU_{max}]$ we can search the interval only in multiplicative steps of $1+\varepsilon$, where $\varepsilon > 0$ is a fixed accuracy parameter. More formally, we find the largest value

$$F' \in \{ (1+\varepsilon)^i \mid i = 0, 1, \ldots \lceil \log_{1+\varepsilon}(nU_{max}) \rceil \}$$

such that there exists flow of value F' of cost at most B. If OPT denotes the optimal flow value, the value F' found by this modified binary search satisfies $F' \geq \text{OPT}/(1+\varepsilon)$. Thus, the modified algorithm computes a solution with performance $1+\varepsilon$.

Theorem 6.10 (Approximation of INTEGER-MAXFL-CPI)
For any fixed $\varepsilon > 0$, there is a $(1+\varepsilon)$-approximation algorithm for INTEGER-MAXFL-CPI. *This algorithm needs the time of $O(\log \log_{1+\varepsilon}(nU_{max}))$ minimum cost flow computations on a graph with $2m$ arcs.* □

Nonlinear Cost Functions

It should be noted that the techniques presented above can be extended to the case when the original cost functions given in the specification of MAXFL-CPI are piecewise-linear convex functions instead of linear functions. If there are g linear pieces and $g-1$ breakpoints in a cost function of a certain arc, then this arc can be replaced by g parallel arcs where each arc is furnished with a linear cost function. Thus, Theorem 6.9 and 6.10 carry over to the more general case of piecewise-linear convex cost functions. However, each of the minimum cost flow computations must now be performed on a graph with up to $g_{max} \cdot m$ arcs, where g_{max} is the maximum number of breakpoints occurring in the piecewise-linear cost functions.

6.4 Approximating Capacity Improvement Problems

In Section 6.3 we provided polynomial time solutions to MAXFL-CPI for continuous and integer upgrade strategies. In the current section we investigate the remaining upgrade strategy, namely the zero-one strategy. This problem variant is denoted by 0/1-MAXFL-CPI.

For the sake of an easier presentation of the approximation algorithms we switch now to the dual problem, 0/1-MINCPI-FL, and recall from Theorem 6.5 that this problem is in fact equivalent to problem MINARCWT-FL. So we will actually deal with problem MINARCWT-FL in this section.

It will turn out in Section 6.5 that MINARCWT-FL is NP-hard even on series-parallel graphs. Therefore we develop in the sequel an approximation algorithm on series-parallel graphs. The current section is concluded with a suggestion on an approach approximating the problem on general graphs.

6.4.1 An mFPAS on Series-Parallel Graphs

We assume without loss of generality that the terminals of a series-parallel graph coincide with the pair $\{s, t\}$ of source and target. This assumption is admissible due to the following observation: If there is a series-parallel graph and two arbitrary nodes s, t in that graph where t can be reached from s, then the subgraph induced by the set of those nodes which can both be reached from s and reach t is itself series-parallel (with terminals s and t).

Let $I = (G, u_r, c_r, F)$ be an instance of MINARCWT-FL. We denote by

$$C_{max} := \max_{r \in R} c_r$$

the maximal arc cost appearing in the input instance.

Auxiliary Algorithm

We continue with developing an auxiliary algorithm. This algorithm solves the dual problem, namely MAXFL-ARCWT, and is employed afterwards within a binary search to solve the first problem. A detailed description can be found in Algorithm 6.3 on the next page.

The idea of Algorithm 6.3 is as follows: For an arbitrary series-parallel graph H with two terminals we define

$$F_H(\beta)$$

to be the maximal flow in graph H between the terminals after upgrading the graph with budget β. Thus, number $F_G(B)$ is the value of an optimal solution for MAXFL-ARCWT.

We can now exploit the fact that series-parallel graphs are decomposable. Starting with single arcs we evolve a dynamic programming scheme to compute the set $\{ F_H(\beta) \mid \beta = 1, \ldots, B \}$ for all subgraphs H of the decomposition of G.

Clearly, for a single arc r, the value $F_r(\beta)$ attains only two values: as long as $\beta < c_r$, we cannot afford to buy the arc, and hence $F_r(\beta) = 0$. Otherwise, if $\beta \geq c_r$, we have $F_r(\beta) = u_r$.

Now consider the case that H is the series composition of H_1 and H_2, and that the sets $\{F_{H_1}(\beta)\}$ and $\{F_{H_2}(\beta)\}$ are already computed. In order to compute

Input: A series-parallel graph G with terminals s, t,
arc capacities u_r,
arc costs c_r,
budget constraint B

if G is a single arc r **then**
  for $\beta = 1, \ldots, B$ **do**
    Assign $F_G(\beta) := \begin{cases} 0, & \text{if } \beta < c_r \\ u_r, & \text{otherwise} \end{cases}$
  end for
else
  G decomposes into G_1 and G_2
  Call algorithm recursively on G_1 and G_2
  if G is series composition of G_1 and G_2 **then**
    for $\beta = 1, \ldots, B$ **do**
      assign $F_G(\beta) := \max_{0 \le i \le \beta} \min\{F_{G_1}(\beta), F_{G_2}(\beta - i)\}$
    end for
  else
    { G *is parallel composition of* G_1 *and* G_2 }
    for $\beta = 1, \ldots, B$ **do**
      assign $F_G(\beta) := \max_{0 \le i \le \beta} F_{G_1}(\beta) + C_{G_2}(\beta - i)$
    end for
  end if
end if

ALGORITHM 6.3: Auxiliary algorithm solving the dual problem MAXFL-ARCWT.

value $F_H(\beta)$, it suffices to try all possible distributions of budget β on the two graphs H_1 and H_2 and respect the fact that the resulting flow must not exceed the capacity constraints of both graphs. Therefore, by

$$F_H(\beta) := \max_{0 \le i \le \beta} \min\{F_{H_1}(i), F_{H_2}(\beta - i)\} \tag{6.1}$$

the value $F_H(\beta)$ can be computed. Similarly, if H is the parallel composition of H_1 and H_2, then by

$$F_H(\beta) = \max_{0 \le i \le \beta} F_{H_1}(i) + F_{H_2}(\beta - i) \tag{6.2}$$

the value $F_H(\beta)$ can be computed.

Assuming that all numbers for a decomposition are already computed, the running time for calculating one value $F_H(\beta)$ is in $O(\beta) \subseteq O(B)$. Hence the set $\{F_H(\beta) \mid \beta = 1, \dots, B\}$ can be computed in time $O(B^2)$ per arc. Since the size of the decomposition tree is $O(m)$, the total running time of Algorithm 6.3 is in $O(mB^2)$. Notice that the algorithm can also keep track of the respective arc sets, such that it can also output the arc set of the optimal solution.

Scaling

The bound on the running time of Algorithm 6.3 shows that the algorithm is *pseudopolynomial*. Therefore we scale down the time-limiting budget value in order to get a polynomial running time. Since we want to apply the algorithm unchanged, the costs of the arcs should be scaled down by the same ratio.

The scale factor should be a number in the same order of magnitude than budget constraint B. This constraint is not known in advance. We can only figure out that B must be a number within the interval $[1, mC_{max}]$. To this end, we choose an arbitrary test parameter $M \in [1, mC_{max}]$. We will show later how to determine the correct parameter.

Let $\varepsilon > 0$ be a given accuracy requirement. Now we construct an instance I' by scaling the budget constraint and all arc costs in the graph by the same factor $M\varepsilon/m$. The resulting scaled costs are defined to be

$$c_r^M := \left\lceil \frac{c_r}{\frac{M\varepsilon}{m}} \right\rceil = \left\lceil \frac{c_r m}{M\varepsilon} \right\rceil .$$

We now use Algorithm 6.3 to perform a test whether parameter M is a good guess value for the budget constraint. The algorithm, when run on I' with budget constraint $(1 + 1/\varepsilon)m$, has a polynomial running time of $O(m^3(1 + 1/\varepsilon)^2)$. We call the test *successful*, if the computed arc set A' admits a flow of value at

least F. The main idea is that we simply use arc set A' as a solution on the unscaled instance I.

Denote by A^* the arc set of an optimal solution to MINARCWT-FL on the unscaled instance I, and let $OPT := c(A^*)$ be its cost. We now show for which cases the above test is successful:

Lemma 6.11 (Test Success)
The test is successful if $M \geq OPT$.

Proof. Consider the optimal arc set A^*. We argue that A^* is a feasible solution on the scaled instance: Its total cost is given by

$$\begin{aligned}\sum_{r\in A^*} c_r^M &\leq \sum_{r\in A^*} \left(\frac{m c_r}{M\varepsilon} + 1\right) \leq \frac{m}{M\varepsilon} \sum_{r\in A^*} c_r + |A^*| \\ &= \frac{m}{\varepsilon} \cdot \frac{OPT}{M} + |A^*| \leq \frac{m}{\varepsilon} + |A^*| \\ &\leq (1 + 1/\varepsilon) \cdot m\,.\end{aligned}$$

Thus the arc set A^* is a feasible solution for the scaled instance. By definition, A^* admits a flow of value F. Since Algorithm 6.3 computes a feasible solution with maximal flow, this flow is at least F and hence the test is successful. □

We now use a binary search to find the smallest integer $M' \in [0, mC_{max}]$ such that the test described above succeeds. From Lemma 6.11 we conclude that $M' \leq OPT$. Let A' be the corresponding arc set found on scaled instance I'. We now estimate the quality of arc set A', when used as a solution on the original instance.

Lemma 6.12 (Cost Bound)
The cost $c(A')$ *is bounded by* $(1+\varepsilon)OPT$.

Proof. The proof uses an easy calculation:

$$\begin{aligned}\sum_{r\in A'} c_r &\leq \frac{M'\varepsilon}{m} \sum_{r\in A'} c_r^{M'} \\ &\leq \frac{M'\varepsilon}{m} \cdot (1 + 1/\varepsilon) \cdot m \\ &\leq (1+\varepsilon) \cdot OPT\,.\end{aligned}$$

This shows the claim. □

Input: A series-parallel graph G with terminals s, t,
arc capacities u_r,
arc costs c_r,
minimal flow value F

Let $C_{max} := \max_{r \in R} c_r$
for test value $M \in [1, mC_{max}]$ with binary search **do**
  Compute scaled costs c^M
  Call Algorithm 6.3 on the scaled instance
  Decide whether resulting flow is at least F
end for
Let M' be the smallest test value with successful test
Let A' be the corresponding set of arcs
return A' as a solution set

ALGORITHM 6.4: Algorithm for approximating MINARCWT-FL.

The complete algorithm is depicted in Algorithm 6.4. The running time of Algorithm 6.4 can be bounded as follows: We run $O(\log mC_{max})$ tests on scaled instances, each of which needs $O(m^3(1 + 1/\varepsilon))$ time. Thus, the total running time is $O(m^3(1 + 1/\varepsilon) \log mC_{max})$. This is a polynomial both in the input size and $1/\varepsilon$. Hence we have constructed an FPAS.

Theorem 6.13 (Approximation of MINARCWT-FL)
For any $\varepsilon > 0$, MINARCWT-FL *can be approximated with performance $(1+\varepsilon, 1)$ on series-parallel graphs. The corresponding uni-criterion problem admits an FPAS.* □

With the help of Theorem 1.3 on page 14 one can obtain a similar result for the dual problem:

Corollary 6.14 (Approximation of MAXFL-ARCWT)
There is an mFPAS for the problems MAXFL-ARCWT *and* 0/1-MAXFL-CPI *when restricted to series-parallel graphs.* □

6.4.2 Towards an Approximation on General Graphs

In the sequel we present an approach to approximating MINARCWT-FL on general graphs. This algorithm works both for the directed as well as for undirected graphs. We stress that the result is not very strong, since we can only show a performance guarantee of F for the cost, where F is the flow value to achieve.

We describe the algorithm. Given an instance $I = (G, u_r, c_r, F)$ of problem MINARCWT-FL, we construct an instance I' of MINIMUM COST FLOW: The capacities u'_r are set in instance I' equal to those of instance I. The unit flow cost on each arc r are set to $c'_r := c_r/u_r$. Now compute an integral minimum cost flow with value F on instance I'. Let x' be the resulting flow.

At this point, define a solution A to MINARCWT-FL. An arc r is a member of the solution, if it has positive flow on instance I'. Formally, we define

$$A := \{ r \in R \mid x(r) > 0 \}.$$

Obviously, the set A admits a flow of value at least F and is therefore a feasible solution.

Notice that the total cost $C(x)$ of the flow x on instance I' which is given by

$$C(x) = \sum_{r \in R} \frac{c_r}{u_r} x(r)$$

is bounded from above by OPT. Here, OPT denotes the cost of an optimal solution on instance I. This is true due to the fact that this optimal solution is also a feasible solution on I'.

It remains to estimate the cost of the arc set A. The total cost are given by

$$\begin{aligned} \sum_{r \in A} c_r &= \sum_{r \in A} u_r \frac{c_r}{u_r} \leq F \cdot \sum_{r \in A} \frac{c_r}{u_r} \\ &\leq F \cdot \sum_{r \in A} \frac{c_r}{u_r} \cdot x(r) \\ &= F \cdot C(x) \leq F \cdot \mathrm{OPT}. \end{aligned}$$

Thus, the set A has total cost at most $F \cdot \mathrm{OPT}$.

Theorem 6.15 (Approximation of MINARCWT-FL)
Problem MINARCWT-FL *can be approximated with performance* $(F, 1)$ *on general graphs where* F *is the flow value to be achieved.* □

The approximation can be performed in time $O(m + T_{MCF})$, where T_{MCF} is the time needed to compute a minimum cost flow of value F in the input graph with capacities u_e and per unit flow costs of c_e/u_e.

6.5 Hardness of Capacity Improvement Problems

It is already known in literature that the decision version of the dual problem, namely MAXFL-ARCWT, is NP-complete [GJ79, Problem ND32]. By duality, this hardness result holds even for MINARCWT-FL. We will give a proof which shows the hardness even for the restriction to series-parallel graphs.

Theorem 6.16 (Hardness of MINARCWT-FL)
MINARCWT-FL *is NP-hard even on series-parallel graphs.*

Proof. We show the claim by a reduction from KNAPSACK. An instance of KNAPSACK is given by a finite set $A = \{a_1, \ldots, a_k\}$ of items, each item with weight $w(a_i) \geq 0$ and value $l(a_i) \geq 0$, and two integers W and L. It is NP-complete to decide whether there is a subset $A' \subseteq A$ such that $w(A') \leq W$ and $l(A') \geq L$ [GJ79, Problem MP9].

Given an instance of KNAPSACK, we construct a graph with vertex set $\{s, t\}$ joined by $|A|$ parallel arcs. For item a_i, arc r_i has cost $w(a_i)$ and capacity $l(a_i)$. We set the flow constraint to the minimum value L of the knapsack.

It is easy to see that the instance of KNAPSACK has a solution if and only if there is a solution of MINARCWT-FL by a selection of arcs of cost at most W. □

The above proof says nothing on the hardness of approximating the problem MINARCWT-FL. At least for bipartite graphs we can provide a nontrivial lower bound:

Theorem 6.17 (Hardness of MINARCWT-FL)
MINARCWT-FL *is strongly NP-hard even on bipartite graphs. Unless NP* $\subseteq$ *DTIME*$(N^{O(\log\log N)})$*, for any* $\varepsilon > 0$ *there is no approximation algorithm for* MINARCWT-FL *on bipartite graphs with performance of* $((1-\varepsilon)\ln F, 1)$*, where* F *is the given flow value to be achieved.*

Proof. We show the theorem by providing an approximation preserving reduction from MINIMUM SET COVER. Given an instance of MINIMUM SET COVER with ground set $Q = \{q_1, \ldots, q_k\}$ and family of subsets $\{Q_1, \ldots, Q_l\}$, we first construct the natural bipartite graph, one side of the partition for set nodes and the other for element nodes. We insert an arc (Q_i, q_j) if and only if $q_j \in Q_i$. All these arcs have capacity 1 and zero costs.

We now add a source node s and a sink node t to the graph. The source node is joined to all the set nodes via arcs (s, Q_i) of capacity $|M|$ and cost 1. For each element $q_j \in Q$ there is an arc (q_j, t) from q to the sink with capacity 1 and cost 0. Let us denote the resulting graph by G. Finally, we set the flow value F to be the size $|M|$ of the ground set.

Since the cost of any selection A' of arcs is exactly the number of arcs in E' emanating from the source, we can assume without loss of generality that each such set A' contains all zero-cost arcs between sets and elements and the elements and the sink.

It is now easy to see that the sets $\{Q_i \mid (s, Q_i) \in A'\}$ form a cover of M if and only if the flow in $G[A']$ has value (at least) $|M|$. Thus, a set cover of

size K transforms into a feasible solution for MINARCWT-FL of the same cost and vice versa. By applying the hardness result of MINIMUM SET COVER (see Theorem 1.6 on page 17) the claim follows. □

6.6 Approximating Unit Flow Cost Improvement Problems

We now turn over to the second arc upgrade model defined in Section 6.1.3. The problem to investigate is (MAX: FLOW, COST IMPROVEMENT, TOTAL FLOW COST), denoted by MAXFL-COI-FC for short. We briefly recall the definition of the problem (confer Definition 6.6 on page 150): An upgrade budget can be spent to reduce the unit flow cost on the graph. The total flow cost is limited by a budget. The goal is to maximize the flow through the graph.

In the current section, we give an approximation of the problem on series-parallel graphs. The technique used for the approximation is at a first glance identical to the approach in Section 6.4, but in the current situation one must attach more importance to correctly rounding the fractional values. The hardness of the problem is shown in Section 6.7.

6.6.1 An mFPAS on Series-Parallel Graphs

We start with some notational conventions. The instance I of MAXFL-COI-FC is given by a series-parallel graph G with two terminals and arc attributes as specified in Definition 6.6: each arc is furnished with capacity u, unit flow cost c, minimum unit flow cost c_{min}, discount d and price p. Additionally there is a bound B on the upgrade budget and a bound C on the total flow cost.

We assume that the budget available for upgrading the graph satisfies $B > 0$. We can make this assumption without loss of generality, since in the case $B = 0$ the problem falls back to the well studied MINIMUM COST FLOW problem.

Similarly, we consider the case $C = 0$ separately. In this case, no budget is available for driving the flow. Obviously a valid solution can only use arcs with zero flow cost. Thus for each arc we have only two cases: Either the arc has initially positive flow cost and is not part of the solution, or the arc is upgraded exactly until it has zero flow cost and is part of the solution. When considering an optimal solution, in the first case the arc is not at all upgraded since an optimal solution does not waste budget. Hence the problem is equivalent to MAXFL-ARCWT in this case. So we assume from now on that $C > 0$ is true.

Pseudopolynomial Algorithm

By exploiting the fact that series-parallel graphs can be easily decomposed one can construct an exact algorithm for the problem. This algorithm has pseudopolynomial running time.

The main idea is similar to the approach given before. If H is an arbitrary series-parallel graph with two terminals, we define

$$F_H(\beta, \gamma)$$

to be the maximal flow value in H between the terminals which can be achieved within two restrictions: The total flow cost must not exceed γ, while the budget for upgrading the graph is bounded by β. Obviously, value $F_G(B, C)$ is the overall solution on instance I.

A solution is denoted by

$$\sigma = (x, y)\,,$$

where x is the flow on the arc and y is the upgrade strategy. Further we write

$$\mathrm{OPT} := F(\sigma^*)$$

where $\sigma^* = (x^*, y^*)$ denotes an optimal solution.

In the following we describe the dynamic programming scheme employed to solve the problem. Consider a single arc r and arbitrary bounds β, γ with $0 \leq \beta \leq B$ and $0 \leq \gamma \leq C$. The maximal improvement $y(r)$ is given by

$$y(r) = \min\left\{ \left\lfloor \frac{\beta}{p(r)} \right\rfloor, \left\lfloor \frac{c(r) - c_{\min}(r)}{d(r)} \right\rfloor \right\}. \tag{6.3}$$

The maximal flow $x(r)$ which equals $F_r(\beta, \gamma)$ is restricted by both the available flow cost budget and the capacity of the arc. Hence

$$F_r(\beta, \gamma) = x(r) = \min\left\{ \left\lfloor \frac{\gamma}{c(r) - d(r)y(r)} \right\rfloor, u(r) \right\}. \tag{6.4}$$

Notice that the above expressions are well-defined and yield finite numbers even if one of the denominators equals zero.

To compute numbers F_G for composite graphs we can apply the same arguments as in (6.1) and (6.2) (see page 159). If H is the series composition of two subgraphs H_1 and H_2, then

$$F_H(\beta, \gamma) = \max_{\substack{0 \leq i \leq \beta \\ 0 \leq j \leq \gamma}} \min\{F_{H_1}(i, j), F_{G_2}(\beta - i, \gamma - j)\}. \tag{6.5}$$

If H is the parallel composition of H_1 and H_2, then

$$F_H(\beta,\gamma) = \max_{\substack{0\le i\le \beta \\ 0\le j\le \gamma}} F_{H_1}(i,j) + F_{H_2}(\beta - i, \gamma - j)\,. \tag{6.6}$$

The running time of the algorithm is estimated as follows: If the table is already completed with numbers for all subgraphs, then the computation of each value $F_H(\beta,\gamma)$ needs at most $O(\beta,\gamma) \subseteq O(BC)$ time. Since m is the size of the decomposition tree, the overall running time of the algorithm can be bounded by $O(mB^2C^2)$. Notice that this is a pseudopolynomial running time.

Scaling

At this point we perform a scaling. Starting with instance I, we construct a scaled instance I' by dividing both the improvement budget and the flow cost bound such that the resulting bounds admit a polynomial running time. The corresponding costs and prices on the arcs must be scaled down by the same factor, such that the ratio between all parameters remains essentially within the same order of magnitude. Then the algorithm is applied to I'. The solution found on the scaled instance I' is used as an approximate solution in the original instance I.

Let $\delta, \varepsilon > 0$ be arbitrary constants. These will be used as approximation ratio of our approach.

In order to keep the rounding errors small we start with a preprocessing on the instance. The background is the following: Assume that we would directly continue with a scaling procedure. Then we would construct scaled prices p' and discounts d' by dividing the initial values by appropriate factors and rounding the quotient to an integer. This means that for each arc we have a worst case error of (almost) one *per unit of upgrade*. If $\tilde{y}$ is the maximal upgrade permitted by the instance, then the error sums up to $\tilde{y}$ on the single arc. It has turned out that this result is too weak and can not be used for analysis. So we decided to multiply prices and discounts for each arc separately by $\tilde{y}$ before the scaling. By this we can keep a maximal rounding error of one *per whole arc*. Of course the new parameters imply slight changes on the algorithm.

We now describe the preprocessing step in detail. With respect to the problem discussed in Section 6.8, we use a more general notation which can also be applied to the case where capacities can be upgraded. To this end, we set

$$\tilde{x}(r) := u(r) \qquad \text{and} \qquad \tilde{y}(r) := \left\lfloor \frac{c(r) - c_{\min}(r)}{d(r)} \right\rfloor \tag{6.7}$$

for each arc r. Value $\tilde{x}(r)$ is the maximal flow on arc r. Value $\tilde{y}(r)$ denotes the maximal improvement on arc r, which is only restricted by the boundary conditions on the arc but not regarding any budget constraints. We continue with defining

$$\begin{aligned} \tilde{c}(r) &:= c(r) \cdot \tilde{x}(r) \cdot \tilde{y}(r)\,, \\ \tilde{d}(r) &:= d(r) \cdot \tilde{x}(r) \cdot \tilde{y}(r)\,, \\ \tilde{p}(r) &:= p(r) \cdot \tilde{y}(r)\,. \end{aligned} \tag{6.8}$$

The algorithm must be trimmed only where values F_H are computed for H being a single arc: To compute $F_r(\beta,\gamma)$ on arc r, we replace (6.3) by the computation

$$y(r) = \min\left\{ \left\lfloor \frac{\beta \cdot \tilde{y}(r)}{\tilde{p}(r)} \right\rfloor, \tilde{y}(r) \right\} \tag{6.9}$$

and replace (6.4) by

$$F_r(\beta,\gamma) = x(r) = \min\left\{ \left\lfloor \frac{\gamma \cdot \tilde{x}(r) \cdot \tilde{y}(r)}{\tilde{c}(r) - \tilde{d}(r) \cdot y(r)} \right\rfloor, \tilde{x}(r) \right\}. \tag{6.10}$$

It is immediate from (6.8) that these modifications are valid.

At this point we can construct the scaled instance I'. For the sake of easier to read formulae we omit the argument r of all functions in the sequel, i. e., we write c instead of $c(r)$ for example.

Parameter δ adjusts the quality of the algorithm with respect to the bound on the upgrade budget. We scale down both the budget and the related arc properties, namely the price, by essentially a factor of $B\delta/3m$:

$$\begin{aligned} B' &:= \lfloor (1 + 1/\delta) \cdot 3m \rfloor \\ p' &:= \left\lceil \frac{3m\tilde{p}}{B\delta} \right\rceil \end{aligned}$$

Observe that $p' = 0$ if and only if $\tilde{p} = 0$.

Similarly, parameter ε adjusts the excess in the total flow cost constraint. The arc properties related to the flow cost constraint are the unit flow cost c and the discount d. All those figures are scaled down by essentially a factor of $C\varepsilon/3m$. For technical reasons only, we work with $\varepsilon' := 3/4 \cdot \varepsilon$ in the sequel.

$$\begin{aligned} C' &:= \lfloor (1 + 1/\varepsilon') \cdot 3m \rfloor \\ c' &:= \left\lceil \frac{3m\tilde{c}}{C\varepsilon'} \right\rceil \\ d' &:= \left\lceil \frac{3m\tilde{d}}{C\varepsilon'} \right\rceil \end{aligned}$$

Notice that the boundary conditions $c' > 0$ and $d' > 0$ (see page 148) still hold. Thus it is admissible to apply the (modified) algorithm to the scaled instance. At this point the construction of the scaled instance I' is complete.

We now run the modified algorithm on the scaled instance I'. Due to the new budget constraints the running time of the algorithm is in $O(m^5(1+\frac{1}{\delta})^2(1+\frac{4}{3\varepsilon})^2)$ and in particular polynomial.

Approximation Algorithm

Let $\sigma = (x, y)$ be a valid solution found on the scaled instance I'. We simply plug σ into the initial instance I and take it as an approximation of the problem. Lemma 6.18 and Lemma 6.19 show that σ is an almost feasible solution.

Lemma 6.18 (Bound on Upgrade Budget)
Solution σ satisfies $B(\sigma) \leq (1+\delta)B$.

Proof. The proof is by a simple analysis. Consider a single arc. By definition of p' we have

$$p' = \left\lceil \frac{3m \cdot \tilde{p}}{B\delta} \right\rceil \geq \frac{3m}{B\delta} \cdot \tilde{p}\,. \tag{6.11}$$

Hence

$$B(\sigma) = \sum_{r \in R} y_r p_r \leq \sum_{r \in R} y_r \tilde{p}_r \overset{(6.11)}{\leq} \frac{B\delta}{3m} \sum_{r \in R} y_r p'_r \tag{6.12}$$

Since the algorithm computes a solution which is feasible within the scaled boundaries, we have $\sum_{r \in R} y_r p'_r \leq B'$. Therefore we continue estimation (6.12) and conclude

$$B(\sigma) \leq \frac{B\delta}{3m} \cdot B' \leq \frac{B\delta}{3m} \cdot \left(1 + \frac{1}{\delta}\right) 3m = (1+\delta)B\,.$$

This shows the claim. □

Lemma 6.19 (Bound of Flow Cost)
Solution σ satisfies $C(\sigma) \leq (1+\varepsilon)C$.

Proof. From the definition we observe for a single arc

$$c'_a = \left\lceil \frac{3m \cdot \tilde{c}_a}{C\varepsilon'} \right\rceil \geq \frac{3m}{C\varepsilon'} \cdot \tilde{c}_a \qquad \text{and} \qquad d'_a = \left\lceil \frac{3m \cdot \tilde{d}_a}{C\varepsilon'} \right\rceil \geq \frac{3m}{C\varepsilon'} \cdot \tilde{d}_a\,.$$

Further notice that $\sum_{r\in R} x(c' - d'y) \leq C'$ since the algorithm is correct. Hence

$$\begin{aligned}
C(\sigma) &= \sum_{r\in R} x \cdot (c - dy) = \sum_{r\in R} x\frac{\tilde{c} - \tilde{d}y}{\tilde{x}\tilde{y}} \\
&\leq \frac{C\varepsilon'}{3m} \cdot \sum_{r\in R} x \cdot \frac{c' - (d'-1)y}{\tilde{x}\tilde{y}} \\
&= \frac{C\varepsilon'}{3m} \sum_{r\in R} \left(x \cdot \frac{c' - d'y}{\tilde{x}\tilde{y}} + \frac{xy}{\tilde{x}\tilde{y}} \right) \\
&\leq \frac{C\varepsilon'}{3m} (C' + m) \\
&\leq \frac{C\varepsilon'}{3m} \left(\left(1 + \frac{1}{\varepsilon'}\right) 3m + m \right) = \left(1 + \frac{4}{3}\varepsilon'\right) C = (1+\varepsilon)C\,.
\end{aligned}$$

We remark that at this point we profit from the preprocessing step, since the rounding error per arc, namely $xy/\tilde{x}\tilde{y} \leq 1$, is not greater than one per arc. □

It remains to show that the overall flow value is at least the flow value of an optimal solution.

Lemma 6.20 (Optimality of Flow Value)
For solution σ, we have $F(\sigma) \geq F_G(B, C)$.

Proof. Let $\sigma^* = (x^*, y^*)$ be an optimal solution of the original (unscaled) instance. This solution exists since MAXFL-COI-FC always has a feasible solution, e. g. the trivial solution $(x \equiv 0, y \equiv 0)$.

We first show that there is a feasible solution $\sigma_0 = (x_0, y_0)$ on the scaled instance which satisfies $x_0 \geq x^*$. This is done by specifying an appropriate distribution (β_0, γ_0) of budget and flow cost.

To this end, let

$$\beta_0 := \left\lceil \frac{p'}{\tilde{y}} y^* \right\rceil \qquad \text{and} \qquad \gamma_0 := \left\lceil \frac{x^*(c' - d'y^*)}{\tilde{x}\tilde{y}} \right\rceil \tag{6.13}$$

for each arc. Clearly, $\tilde{y} \geq y^*$. If $p' > 0$, we have

$$y_0 \overset{(6.9)}{\geq} \left\lfloor \frac{\beta_0 \tilde{y}}{p'} \right\rfloor = \left\lfloor \frac{\left\lceil \frac{p'}{\tilde{y}} y^* \right\rceil}{p'} \tilde{y} \right\rfloor \geq \left\lfloor \frac{\frac{p'}{\tilde{y}} y^*}{p'} \tilde{y} \right\rfloor = y^*\,.$$

Otherwise, if $p' = 0$, then $\tilde{p} = 0$ as observed above. In this case, $y_0 \geq y^*$ follows directly from (6.9). Thus we have that $y_0 \geq y^*$ in all cases.

On the other hand, from definition we have $\tilde{x} \geq x^*$. Moreover,

$$x_0 \overset{(6.10)}{\geq} \left\lfloor \frac{\gamma \cdot \tilde{x} \cdot \tilde{y}}{c' - d' y_0} \right\rfloor = \left\lfloor \frac{\left\lceil \frac{x^*(c'-d'y^*)}{\tilde{x}\tilde{y}} \right\rceil \tilde{x}\tilde{y}}{c' - d' y_0} \right\rfloor \overset{y_0 \geq y^*}{\geq} \left\lfloor \frac{\frac{x^*(c'-d'y^*)}{\tilde{x}\tilde{y}} \cdot \tilde{x}\tilde{y}}{c' - d' y^*} \right\rfloor = x^* .$$

Since the source s has no incoming arcs, this implies that

$$F(\sigma_0) \geq F(\sigma^*) = \mathrm{OPT} .$$

It remains to show that σ_0 is a feasible solution on the scaled instance.

$$\begin{aligned}
B(\sigma_0) &\overset{(6.13)}{=} \sum_{\substack{r \in R \\ \beta \neq 0}} \left\lceil \frac{p'}{\tilde{y}} y^* \right\rceil \\
&\leq \sum_{\substack{r \in R \\ \beta \neq 0}} \left(\frac{\frac{3m \cdot \tilde{p}}{B\delta} + 1}{\tilde{y}} y^* + 1 \right) = \sum_{\substack{r \in R \\ \beta \neq 0}} \left(\frac{3m \cdot \tilde{p} y^*}{B\delta \cdot \tilde{y}} + \frac{y^*}{\tilde{y}} + 1 \right) \\
&\leq \frac{3m}{\delta} \left(\sum_{\substack{r \in R \\ \beta \neq 0}} \frac{\tilde{p} y^*}{B\tilde{y}} \right) + 2m \\
&\leq \frac{3m}{\delta} + 2m \leq \left\lfloor \left(1 + \frac{1}{\delta}\right) 3m \right\rfloor = B' .
\end{aligned}$$

On the other hand,

$$\begin{aligned}
C(\sigma_0) &\overset{(6.13)}{=} \sum_{r \in R} \left\lceil \frac{x^* c' - d' x^* y^*}{\tilde{x}\tilde{y}} \right\rceil \\
&\leq \sum_{r \in R} \left(\frac{\left(\frac{3m\tilde{c}}{C\varepsilon'} + 1\right) x^*}{\tilde{x}\tilde{y}} - \frac{\frac{3m\tilde{c}}{C\varepsilon'} x^* y^*}{\tilde{x}\tilde{y}} + 1 \right) \\
&= \sum_{r \in R} \left(\frac{3m\tilde{c}x^*}{C\varepsilon'\tilde{x}\tilde{y}} + \frac{x^*}{\tilde{x}\tilde{y}} - \frac{3m\tilde{d}x^* y^*}{C\varepsilon'\tilde{x}\tilde{y}} + 1 \right) \\
&\leq \frac{3m}{\varepsilon'} \left(\sum_{r \in R} \frac{\tilde{c}x^* - \tilde{d}x^* y^*}{C\tilde{x}\tilde{y}} \right) + 2m \\
&= \frac{3m}{\varepsilon'} + 2m \leq \left\lfloor \left(1 + \frac{1}{\varepsilon'}\right) 3m \right\rfloor = C' .
\end{aligned}$$

Hence, solution σ_0 is also feasible for the scaled instance. Since the algorithm tries all possible distributions of budget and costs this is true in particular for (β_0, γ_0). Finally we can conclude

$$F(\sigma) \geq F(\sigma_0)$$

since the algorithm is optimal. This shows the claim. □

We summarize the results obtained in this section:

Theorem 6.21 (Approximability of MAXFL-COI-FC)
For any $\delta, \varepsilon > 0$, there is an approximation algorithm for MAXFL-COI-FC *on series-parallel graphs with performance $(1, 1+\delta, 1+\varepsilon)$. Its running time is in $O(m^5(1+\frac{1}{\delta})^2(1+\frac{4}{3\varepsilon})^2)$.* □

6.6.2 Towards an Approximation on General Graphs

In this section we switch over to the related problem (MIN: FLOW COST, FLOW VALUE, COST IMPROVEMENT) and present an approach to an approximation on general graphs. The problem is denoted by MINFC-FL-COI for short. This problem consists in minimizing the flow cost with respect to constraints on flow value and upgrade investment. As noted in Theorem 1.3 on page 14, the approximation results carry over to the initial problem, MAXFL-COI-FC, by a general binary search technique.

The main idea of the approach consists in setting up a function of *compound costs* which combines unit flow costs and improvement costs at the same time. (We refer to [MR+95, KN+98, MR+98] for further applications of this approach.)

The compound cost function is made up of a linear combination of the two cost values which is weighted by a test parameter K. The shape of the compound cost function h_K is shown in Figure 6.5. It is a nondecreasing (piecewise) linear function with minimum c_{min} and maximum c. The slope of the function depends on the price and discount value on the arc.

Assume that the algorithm computes a minimal cost flow with respect to the compound costs. One may observe the following: The smaller the test parameter is, the more the algorithm tends to overlook upgrade costs and hence it uses more upgraded arcs. On the other hand, if the test parameter gets larger and larger, the upgrade effort is weighted more and more and the algorithm prefers to use arcs with low initial weight.

In this sense the test parameter describes a compromise between upgrading arcs and leaving them in the initial state. Moreover, a minimum cost flow with respect to the compound cost function can be used in combination with the test parameter to generate improvement and flow strategies on the original graph.

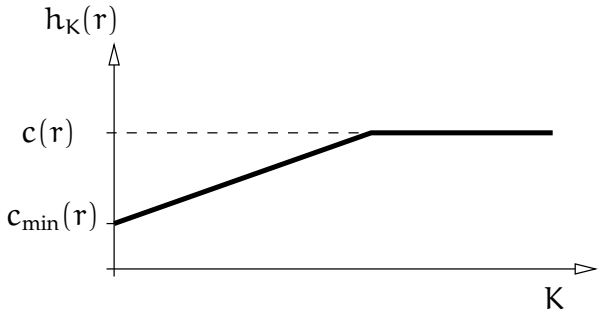

FIGURE 6.5: Shape of combined cost function h_K on arc r.

The quality of the solution determined by this approach depends on how good the test parameter was chosen. At this point, a binary search is applied in order to find a suitable test parameter.

Input: A graph $G = (V, R)$ with terminals $s \in V$ and $t \in V$,
arc capacities u
arc costs c and minimal arc costs c_{min}
upgrade prices p and discount d
flow constraint F, improvement budget B
accuracy parameters $\varepsilon_1, \varepsilon_2 > 0$

1 Let $C_{max} := \frac{1}{\varepsilon_2} \sum_{r \in R} cu$.
2 Perform a search on the set

$$M := \{ (1 + \varepsilon_1)^i \mid i = 0, 1, \ldots, \lceil \log_{1+\varepsilon_1} C_{max} \rceil \}$$

to find the smallest test parameter $K' \in M$ such that Algorithm 6.7 returns a valid solution σ and does not return COF.

Output: Approximate solution σ

ALGORITHM 6.6: Approximation algorithm for MINFC-FL-COI.

The main procedure is depicted in Algorithm 6.6 which uses Algorithm 6.7 as a subroutine. Notice that we omit the arguments r to all functions in order to improve the readability of the formulae. The algorithm is given two accuracy parameters $\varepsilon_1, \varepsilon_2 > 0$ which adjust the performance of the produced solution.

Reasonable values for the test parameter lie between 1 and roughly the maximal conceivable total flow cost, denoted by $C_{max} := \frac{1}{\varepsilon_2} \sum_{r \in R} cu$. This interval is furnished with a grid of exponentially growing steps $\{ (1 + \varepsilon_1)^i \mid i \in \mathbb{N}_0 \}$. This keeps the number of test values polynomial.

Input: Same input as for Algorithm 6.6, additionally a test parameter K

for all $r \in R$ **do**
  if $\frac{K}{B} < d \cdot \frac{u}{p}$ **then**
    Let $y := \frac{c - c_{min}}{d}$
    Let $h_K(r) := c_{min} + \frac{K}{B} \cdot \frac{c - c_{min}}{d} \cdot \frac{p}{u}$
  else
    Let $y := 0$
    Let $h_K(r) := c$
  end if
end for
Let x be a minimum cost flow of value F with respect to cost function h_K
if $\sum_{r \in R} x \cdot h_K(r) \leq (1 + \varepsilon_2) \cdot K$ **then**
  return $\sigma = (x, y)$
else
  return COF { *certificate of failure* }
end if

Output: Solution σ or COF

ALGORITHM 6.7: Test procedure used in Algorithm 6.6.

In the following we establish the result that there is a test parameter where the test in Line 11 of Algorithm 6.7 succeeds and the algorithm returns a solution.

We denote by $\sigma^* = (x^*, y^*)$ an optimal solution with flow x^* and improvement strategy y^*. The resulting flow cost is denoted by

$$\mathrm{OPT} := \sum_{r \in R} x^* \cdot (c - d y^*) .$$

Since the solution is feasible, we have $\sum_{r \in R} p y^* \leq B$.

Lemma 6.22 (Test Success)
] Let $\tilde{K} := \mathrm{OPT}/\varepsilon_2$. Algorithm 6.7 does not return a "certificate of failure" (COF) when called with a parameter $K \geq \tilde{K}$.

Proof. By construction of h_K (confer Step 4 and Step 7 of Algorithm 6.7), we have for each arc $r \in R$

$$h_K(r) = \min_{y \text{ is feasible}} \left(c - d \cdot y + \frac{K}{B} \cdot \frac{p}{u} \cdot y \right) .$$

Consider the optimal flow x^*. Then the flow cost with respect to cost function h_K are given by

$$\begin{aligned} \sum_{r in R} x^* \cdot h_K(r) &\leq \sum_{r \in R} x^* \cdot (c - d \cdot y^*) + \frac{K}{B} \cdot \sum_{r \in R} \frac{x^*}{u} \cdot p y^* \\ &\leq \mathrm{OPT} + \frac{K}{B} \cdot B = \mathrm{OPT} + K \\ &\leq \varepsilon_2 K + K = (1 + \varepsilon_2) \cdot K . \end{aligned}$$

Thus x^* is a witness for a successful test with cost function h_K. Since the algorithm computes a minimum cost flow under that cost function, its total cost would not exceed the cost of x^*. Hence the test is successful. □

It is not necessary that value $\tilde{K}$ actually appears in the set M of test parameters. Nevertheless, since M is furnished with a grid where the ratio of each two subsequent points equals $1 + \varepsilon_1$, we can conclude that the parameter K' found by the algorithm satisfies

$$K' \leq (1 + \varepsilon_1)\tilde{K} . \tag{6.14}$$

At this point we estimate the performance of the algorithm with respect to the resulting total flow cost. To this end, we denote by

$$\mathrm{MCF}(\eta)$$

the cost of a flow of value F with minimum cost with respect to an arbitrary cost function η.

Let K' be the smallest successful test parameter found by Algorithm 6.7. let $\sigma' = (x', y')$ be the corresponding solution with flow x' and upgrade strategy y'. Notice that the cost of x' equals $\mathrm{MCF}(c - dy')$.

Lemma 6.23 (Bound on Flow Costs)
The total costs of flow x' are bounded as follows:

$$\mathit{MCF}(c - dy') \leq \left(1 + \frac{1}{\varepsilon_2}\right)(1 + \varepsilon_1) \cdot \mathrm{OPT}\,.$$

Proof. At first we have

$$\begin{aligned}\sum_{r \in R} x' \cdot h_{K'}(r) &= \sum_{r \in R} x' \cdot \left(c - dy' + \frac{K'}{B} \cdot \frac{p}{u} \cdot y'\right) \\ &\geq \sum_{r \in R} x' \cdot (c - dy') \\ &\geq \mathrm{MCF}(c - dy')\,.\end{aligned}$$

Now notice that both x' and x^* are flows with flow value F. Since x' is a minimum cost flow with respect to cost function $h_{K'}$, we can conclude that

$$\sum_{r \in R} x' \cdot h_{K'}(r) \leq \sum_{r \in R} x^* \cdot h_{K'}(r)\,.$$

Following the proof of Lemma 6.22, this yields

$$\sum_{r \in R} x^* \cdot h_{K'}(r) \leq \mathrm{OPT} + K'\,.$$

Moreover, using (6.14) we proceed

$$\begin{aligned}\mathrm{OPT} + K' &\leq \mathrm{OPT} + \tilde{K}(1 + \varepsilon_1) \\ &\leq \left(\mathrm{OPT} + \tilde{K}\right)(1 + \varepsilon_1) \\ &= \mathrm{OPT}\left(1 + \frac{1}{\varepsilon_2}\right)(1 + \varepsilon_1)\,.\end{aligned}$$

This shows the claim. □

It remains to show the performance of the resulting upgrade strategy with respect to the given budget. Let $u_{\max} := \max_{r \in R} u(r)$ be the maximal capacity specified by the instance. Notice that we can assume that $u_{\max} \leq F$.

Lemma 6.24 (Bound on Upgrade Costs)
The upgrade costs of solution σ' satisfy

$$\sum_{r \in R} y'p \leq (1+\varepsilon_2) u_{\max} B \,.$$

Proof. The costs of flow x' satisfy

$$\begin{aligned} \mathrm{MCF}(h_{K'}) &= \sum_{r \in R} x' \cdot \left(c - dy' + \frac{K'}{B} \cdot \frac{p}{u} \cdot y \right) \\ &\geq \frac{K'}{B} \cdot \sum_{r \in R} \frac{x'}{u} \cdot py \,. \end{aligned}$$

Since $x'/u \geq 1/u_{\max}$, the right hand side can further be bounded by

$$\frac{K'}{B} \cdot \sum_{r \in R} \frac{x'}{u} \cdot py \geq \frac{K'}{B} \cdot \sum_{r \in R} py \,.$$

On the other hand, since the test was successful, we have

$$\mathrm{MCF}(h_{K'}) \leq (1+\varepsilon_2) K' \,.$$

Multiplying the chain of inequalities by B/K' yields the claim. □

The following theorem summarizes the results of the current section. Notice that the result is not quite satisfying unless for those instances where $u_{\max}$ is small.

Theorem 6.25 (Approximation of MINFC-FL-COI on general graphs)
For any constant $\varepsilon_1, \varepsilon_2 > 0$, problem MINFC-FL-COI *can be approximated with performance ratio*

$$\left((1+1/\varepsilon_2)(1+\varepsilon_1), 1, (1+\varepsilon_2) u_{\max} \right) ,$$

where $u_{\max} = \max_{r \in R} u(r)$ is the maximal capacity. □

At the end we estimate the running time of the algorithm. Algorithm 6.6 performs at most $\log_{1+\varepsilon_1} C_{\max}$ iterations where $C_{\max} = \frac{1}{\varepsilon_2} \sum_{r \in R} cu$. The running time of the test algorithm (Algorithm 6.7) is dominated by the time T_{MCF} needed for a minimum cost flow computation. Thus we have

$$O\left(\log_{1+\varepsilon_1} \left(\frac{1}{\varepsilon_2} n c_{\max} u_{\max} \right) \cdot T_{\mathrm{MCF}} \right)$$

as a bound on the overall running time.

6.7 Hardness of Unit Flow Cost Improvement Problems

When attempting to apply a programming approach to the family of problems one gets the easy insight that MINFC-FL-COI can be written as a bilinear program with disjoint constraints. This means, it can be written as a program

$$\begin{aligned} &\text{minimize } xMy \\ \text{subject to} \\ &Ax \leq b \\ &Cy \leq d \end{aligned}$$

Since it is known that programs of this type are NP-hard to solve [FV94], this can be considered as a hint to the fact that even MINFC-FL-COI and the related problems are intractable. Instead of tracking this approach we directly provide non-approximability results by using other reductions in the sequel.

Theorem 6.26 (Non-approximability of MAXFL-COI-FC)
For any (polynomial time computable) function $f(n)$, the existence of an approximation algorithm for MAXFL-COI-FC *with performance $(1, 1, f(n))$ on series-parallel graphs with n nodes implies $P = NP$.*

Proof. We show the hardness by a reduction from KNAPSACK [GJ79, Problem MP9]. An instance of KNAPSACK is given by a finite set $A = \{a_1, \ldots, a_k\}$ of items, each of weight $w(a_i) \geq 0$ and value $v(a_i) \geq 0$, and two numbers $W, V \in \mathbb{N}$. It is NP-complete to decide whether there is a subset $A' \subseteq A$ such that $w(A') \leq W$ and $v(A') \geq V$.

Assume there is a $(1, 1, f)$-approximation algorithm for MAXFL-COI-FC. Given an instance of KNAPSACK, we construct a graph with vertex set $\{s, t\}$ joined by k parallel arcs. For item a_i, arc r_i has capacity $u(r_i) := v(a_i)$, and price $p(r_i) := w(a_i)$. Further, for all arcs set the initial flow cost $c(r) := f \cdot V + 1$, the minimal flow cost $c_{\min}(r) := 1$, and the discount $d(r) := f \cdot V$. Finally, set the flow cost constraint $C := V$, and the budget constraint $B := W$.

We claim that there is a solution of the MAXFL-COI-FC instance with flow value at least V if and only if there is a solution to the KNAPSACK instance.

Assume that there is a solution of MAXFL-COI-FC with flow value $F \geq V$ obeying both constraints. Let R' be the set of upgraded arcs, choose A' to be the corresponding set of items. Due to the flow cost constraint it is easy to see that the solution of MAXFL-COI-FC uses only upgraded arcs. Then $\sum_{a' \in A'} v(a') = \sum_{r' \in R'} u(r') \geq F \geq V$ and $\sum_{a' \in A'} w(a') = \sum_{r' \in R'} p(r') \leq B = W$, therefore A'

is a solution for KNAPSACK. Conversely, constructing a solution for MAXFL-COI-FC out of a solution of KNAPSACK is easily achieved by upgrading the arcs corresponding to the items of the solution.

Since the flow cost of an arc r which is not upgraded exceeds $f \cdot V$, any $(1, 1, f)$-approximation algorithm for MAXFL-COI-FC must in fact solve the underlying KNAPSACK problem exactly. □

We briefly compare this result to Theorem 6.21 on page 171. The approximation violates both constraints (albeit by small factors only). The above result shows, that at least the violation of the improvement budget is necessary in order to get a good approximation. We supplement a non-approximability result for the class of bipartite graphs.

Theorem 6.27 (Non-approximability of MAXFL-COI-FC)
For any $\varepsilon > 0$ and any (polynomial time computable) function $f(n)$, the existence of a $(1, (1-\varepsilon)\ln n, f(n))$-approximation algorithm for MAXFL-COI-FC *on bipartite graphs with n nodes implies NP $\subseteq$ DTIME$(N^{O(\log\log N)})$.*

Proof. The proof uses a reduction from MINIMUM DOMINATING SET [GJ79, Problem GT2]. An instance of MINIMUM DOMINATING SET is given by a graph $G = (V, E)$ and a number $K \in \mathbb{N}$. A subset $V' \subseteq V$ is called *dominating*, if each node in V is either contained in V' or incident to a node from V'. It is NP-complete to decide whether G admits a dominating set of size at most K.

Given an instance $G' = (V', E')$ of MINIMUM DOMINATING SET with $n' := |V'|$ nodes, construct a graph G with node set $V_1 \cup V_2 \cup \{s, t\}$, where each V_i is a copy of V'. Insert an arc between $v \in V_1$ and $w \in V_2$ if and only if w is dominated by v in the original graph. For each $v \in V_1$ insert an arc (s, v) of cost $c := f + 1$, discount $d := f$, minimum cost $c_{min} := 1$ and price $p := 1$. For each $w \in V_2$ add an arc (w, t) of capacity $u := 1$. If not specified yet, set the capacity on the remaining arcs to $u := n'$, and the costs and prices to zero. Notice that the resulting graph is bipartite, and only the arcs incident to s cause flow costs.

Observe that due to the capacity constraint, a flow with flow value n' induces a flow of value 1 through each of the nodes in layer V_2. Therefore the set of vertices in layer V_1 with nonzero flow form a dominating set in the original graph. The cost of the flow is equal to n' if it does not use an arc incident to s which is not upgraded; otherwise the costs are strictly larger than f.

Let OPT be the size of a minimum dominating set in the original graph. Perform for each $i \in \{1, \ldots, n\}$ a test T_i as follows: Run the approximation algorithm for MAXFL-COI-FC on graph G with bound i on the upgrade costs and bound n on the flow costs, and check whether the resulting flow value is not less than n'.

Assume there is a $(1, (1-\varepsilon)\ln n, f)$-approximation algorithm for MAXFL-COI-FC. Consider instance T_{OPT}. By our observation, there is a feasible solution for T_{OPT} with flow value n'. Since the approximation ratio with respect to the flow cost is f, the algorithm must in fact produce a solution which does not use an arc incident to s which is not upgraded. By the performance bound on the improvement cost, the algorithm must find a dominating set of size at most $(1-\varepsilon)\ln n \cdot OPT$. Finally, the resulting flow value is n'.

After performing all tests T_i for $i = 1, \ldots, n'$ and taking the upgrade budget minimal solution out of those solutions which achieve a flow value of n', the final solution induces a dominating set of size at most $(1-\varepsilon)\ln n \cdot OPT$. Observe that the number n of nodes in the constructed graph satisfies $n = 2n' + 2$. If n' is large enough (this depends solely on ε), there is an $\varepsilon' > 0$ such that $(1-\varepsilon)\ln n \leq (1-\varepsilon')\ln n'$. Hence the algorithm induces an approximation of MINIMUM DOMINATING SET with performance $(1-\varepsilon')\ln n'$ which implies $NP \subset DTIME(N^{O(\log\log N)})$ in accordance with the result of Theorem 1.8 on page 17. □

6.8 Combined Improvement

In this section we turn over to problem MAXFL-COCPI-FC (see Definition 6.7 on page 152). This problem is formulated under the combined upgrade model which admits both improvement of unit flow costs and improvement of capacities at the same time.

6.8.1 Approximating Combined Improvement Problems

We now argue that the methods from Section 6.6.1 for approximating MAXFL-COI-FC on series-parallel graphs can be successfully applied also to MAXFL-COCPI-FC with only small changes. This enables us to derive similar approximation results.

Pseudopolynomial Algorithm

Recall from Section 6.6.1 that for arbitrary series-parallel graph H,

$$F_H(\beta, \gamma)$$

denotes the maximal flow between the terminals of H where the improvement budget is restricted to β and the total flow cost are bounded by γ.

For a single arc r, value $F_r(\beta, \gamma)$ is determined as follows: assume for the moment that we know how the budget is distributed between cost improvement and capacity improvement. Let $\beta = \beta_a + \beta_d$ be the partition of the budget. Clearly, the optimal upgrade strategy is then defined by

$$y_a(r) = \min\left\{\left\lfloor\frac{\beta_a}{p_a(r)}\right\rfloor, \left\lfloor\frac{u_{max}(r) - u(r)}{a(r)}\right\rfloor\right\} \qquad \text{and}$$
$$y_d(r) = \min\left\{\left\lfloor\frac{\beta_d}{p_d(r)}\right\rfloor, \left\lfloor\frac{c(r) - c_{min}(r)}{d(r)}\right\rfloor\right\}.$$

The optimal flow $F_r(\beta, \gamma)$ is computed by trying all possible distributions of the budget into the two types of improvement. Thus,

$$F_r(\beta, \gamma) = x(r) = \max_{\beta_a + \beta_d = \beta} \min\left\{\left\lfloor\frac{\gamma}{c(r) - d(r)y_d(r)}\right\rfloor, u(r) + a(r)y_a(r)\right\}.$$

The computations for interior nodes of the decomposition tree are exactly as rules (6.5) and (6.6) described on page 165 for the variant under the simpler cost improvement model. Thus, the desired value $F_G(B, C)$ can be computed in pseudopolynomial running time $O(mB^2C^2)$ as before.

Scaling

The idea behind the scaling are directly derived from Section 6.6.1. At first, we introduce upper bound functions $\tilde{x}$, $\tilde{y}_a$, and $\tilde{y}_d$ similar to (6.7). In the current context, this reads

$$\tilde{y}_a(r) := \left\lfloor\frac{u_{max}(r) - u(r)}{a(r)}\right\rfloor$$
$$\tilde{y}_d(r) := \left\lfloor\frac{c(r) - c_{min}(r)}{d(r)}\right\rfloor$$
$$\tilde{x}(r) := \tilde{y}_a(r)a(r) + u(r)$$

Analogously to (6.8), we set

$$\tilde{c}(r) := c(r) \cdot \tilde{x}(r) \cdot \tilde{y}_d(r)$$
$$\tilde{d}(r) := d(r) \cdot \tilde{x}(r) \cdot \tilde{y}_d(r)$$
$$\tilde{p}_a(r) := p_a(r) \cdot \tilde{y}_a(r)$$
$$\tilde{p}_d(r) := p_d(r) \cdot \tilde{y}_d(r).$$

After that, the calculations (6.9) and (6.10) for $F_r(\beta, \gamma)$ on a leaf of the decomposition tree are changed to

$$y_a := \min\left\{\left\lfloor \frac{\beta_a \tilde{y}_a}{\tilde{p}_a} \right\rfloor, \tilde{y}_a\right\}$$
$$y_d := \min\left\{\left\lfloor \frac{\beta_d \tilde{y}_d}{\tilde{p}_d} \right\rfloor, \tilde{y}_d\right\}$$
$$F_r(\beta, \gamma) = x(r) := \max_{\beta_a + \beta_d = \beta} \min\left\{\left\lfloor \frac{\gamma \cdot \tilde{x} \cdot \tilde{y}_d}{\tilde{c} - \tilde{d} \cdot y_d} \right\rfloor, u + a \cdot y_a\right\}$$

At this point, the scaled instance can be constructed. Given accuracy parameters $\delta, \varepsilon > 0$, the budget related values are scaled as

$$B' := \left\lfloor \left(1 + \frac{1}{\delta}\right) \cdot 5m \right\rfloor$$
$$p_a := \left\lceil \frac{5m \cdot \tilde{p}_a}{B\delta} \right\rceil$$
$$p_d := \left\lceil \frac{5m \cdot \tilde{p}_d}{B\delta} \right\rceil .$$

The cost related values are scaled like (using $\varepsilon' := 5/6 \cdot \varepsilon$)

$$C' := \left\lfloor \left(1 + \frac{1'}{\varepsilon}\right) \cdot 5m \right\rfloor$$
$$c' := \left\lceil \frac{5m \cdot \tilde{c}}{C\varepsilon'} \right\rceil$$
$$d' := \left\lceil \frac{5m \cdot \tilde{d}}{C\varepsilon'} \right\rceil .$$

Approximation Algorithm

As before, the algorithm is run on the scaled instance and the solution is used as an approximation to the initial problem on the unscaled instance. The calculations from Lemma 6.18, Lemma 6.19, and Lemma 6.20 can immediately be carried out in the current situation without major changes. Thus we can conclude the following result:

Theorem 6.28 (Approximability of MAXFL-COCPI-FC)
For any $\delta, \varepsilon > 0$, there is an approximation algorithm for MAXFL-COCPI-FC *on series-parallel graphs with performance $(1, 1 + \delta, 1 + \varepsilon)$. Its running time is in $O(m^5(1 + \frac{1}{\delta})^2(1 + \frac{6}{5\varepsilon})^2)$.*

6.8.2 Hardness of Combined Improvement Problems

It is easy to see that MAXFL-COCPI-FC contains MAXFL-COI-FC as a special case: A reduction can be achieved by choosing the price $p_a(r)$ for upgrading the capacity of arc r to a constant exceeding the available budget. Therefore all hardness and non-approximability results derived in Section 6.7 immediately carry over to MAXFL-COCPI-FC.

6.9 Concluding Remarks

A summary of the explicit results obtained in this chapter is displayed in Table 6.8 on the next page. The results carry over to the family of related problems according to Theorem 1.3 on page 14.

All negative results, i. e., results on hardness and lower bounds on the approximability, rely upon the widely believed assumptions $P \neq NP$ and $NP \not\subseteq DTIME(N^{O(\log\log N)})$. For some problems there has been provided a (multi-criteria) fully polynomial approximation scheme which means that the running time of the approximation algorithm is both polynomial in the size of the graph and in $1/\delta$ and $1/\varepsilon$ (see page 15 for definition).

Problem	Positive Results	Negative Results
MAXFL-CPI		
CONTINUOUS	polynomial *(Theorem 6.8)*	
INTEGER	polynomial *(Theorem 6.9)*	
0/1	dual problem 0/1-MINCPI-FL is equivalent to MINARCWT-FL *(Theorem 6.5)*	
MINARCWT-FL	FPAS $(1+\varepsilon, 1)$-approximable on series-parallel graphs *(Theorem 6.13)*	NP-hard on series-parallel graphs *(Theorem 6.16)*
	$(F, 1)$-approximable on general graphs *(Theorem 6.15)*	lower bound $(\ln F, 1)$ on bipartite graphs *(Theorem 6.17)*
MAXFL-COI-FC	mFPAS $(1, 1+\delta, 1+\varepsilon)$-approximable on series-parallel graphs *(Theorem 6.21)*	lower bound $(1, 1, \alpha)$ on series-parallel graphs *(Theorem 6.26)*
	related problem MINFC-FL-COI $((1+1/\varepsilon)(1+\delta), 1, (1+\varepsilon)u_{max})$-approximable on general graphs *(Theorem 6.25)*	lower bound $(1, \ln n, \alpha)$ on bipartite graphs *(Theorem 6.27)*
MAXFL-COCPI-FC	mFPAS $(1, 1+\delta, 1+\varepsilon)$-approximable on series-parallel graphs *(Theorem 6.28)*	as least as hard as MAXFL-COI-FC

TABLE 6.8: Summary of explicit results presented in this chapter.

Synopsis

This chapter is intended as a brief reference for the results obtained in this thesis. It consists mainly of an uncommented compilation of facts.

Minimum Label Spanning Tree

Approach	Performance	Reference
Algorithm 2.3 running time $O(m + \alpha(m,n) \cdot l \cdot \min\{\ln, m\})$	$2 \ln n + 1$	Theorem 2.8
polynomial time algorithm	$(1+\varepsilon) \ln n$ for any $\varepsilon > 0$	Theorem 2.9
lower bound	$6/31 \cdot \ln n$	Theorem 2.11

Minimum Reload Cost Subgraphs

Problem	Degree bound	Lower bound on approximability	Reference
M$^{\Delta}$RPATH	none	solvable in $O(\|V\|^3 + \|E\|)$	Corollary 3.11
MRRADT	5	any $f(\|V\|)$	Corollary 3.31
M$^{\Delta}$RRADT	5	2	Corollary 3.32
	3	solvable in $O(\|V\|)$	Theorem 3.18
MRDIAT	5	any $f(\|V\|)$	Corollary 3.30
M$^{\Delta}$RDIAT	none	$1/4 \cdot \ln \|V\|$	Theorem 3.35
	5	3	Corollary 3.34
	3	solvable in $O(\|V\|^2 \log \|V\|)$	Theorem 3.28

Dial a Ride

Graph class	DARP	SOURCE-DARP	PENALTY-SOURCE-DARP
paths	polynomial time solvable *[AK88]*	polynomial time solvable *(Theorem 4.21)*	NP-hard *(Theorem 4.40)*
			approximable within 5/3 *(Corollary 4.35)*
trees	NP-hard, even on caterpillars *(Theorem 4.36)*	NP-hard, even on caterpillars *(Theorem 4.36)*	NP-hard
	approximable within 5/4 *[FG93]*	approximable within 3/2 *(Theorem 4.32)*	approximable within 5/3 *(Corollary 4.35)*
general graphs	NP-hard *[FHK78]*	NP-hard	NP-hard
	approximable within 9/5 *[FHK78]*	approximable within 9/4 *(Theorem 4.29)*	approximable within 9/4 *(Corollary 4.35)*

Node Upgrade Constrained Forest

lower bound	$(\ln \lvert K\rvert, 1)$	*(Theorem 5.17)*
	$(\ln \lvert K\rvert, f(\lvert V\rvert))$	*(Corollary 5.18)*
upper bound	$(2\ln(\sqrt{e}/2 \cdot \lvert K\rvert), 1)$	*(Theorem 5.16)*

Problem	Positive Results	Negative Results
MAXFL-CPI		
CONTINUOUS	polynomial *(Theorem 6.8)*	
INTEGER	polynomial *(Theorem 6.9)*	
0/1	dual problem 0/1-MINCPI-FL is equivalent to MINARCWT-FL *(Theorem 6.5)*	
MINARCWT-FL	FPAS $(1+\varepsilon, 1)$-approximable on series-parallel graphs *(Theorem 6.13)*	NP-hard on series-parallel graphs *(Theorem 6.16)*
	$(F, 1)$-approximable on general graphs *(Theorem 6.15)*	lower bound $(\ln F, 1)$ on bipartite graphs *(Theorem 6.17)*
MAXFL-COI-FC	mFPAS $(1, 1+\delta, 1+\varepsilon)$-approximable on series-parallel graphs *(Theorem 6.21)*	lower bound $(1, 1, \alpha)$ on series-parallel graphs *(Theorem 6.26)*
	related problem MINFC-FL-COI $((1+1/\varepsilon)(1+\delta), 1, (1+\varepsilon)u_{max})$-approximable on general graphs *(Theorem 6.25)*	lower bound $(1, \ln n, \alpha)$ on bipartite graphs *(Theorem 6.27)*
MAXFL-COCPI-FC	mFPAS $(1, 1+\delta, 1+\varepsilon)$-approximable on series-parallel graphs *(Theorem 6.28)*	as least as hard as MAXFL-COI-FC

Bibliography

[AA+95] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala · *Improved approximation guarantees for minimum-weight* k*-trees and prize-collecting salesmen* · Proceedings of the 27th Annual ACM Symposium on the Theory of Computing (STOC'95), 1995, pp. 277–283.

[AC+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi · *Complexity and approximation* · Springer, Berlin, Heidelberg, New York, 1999.

[AF+95] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo · *Competitive algorithms for the traveling salesman* · Proceedings of the 4th Workshop on Algorithms and Data Structures (WADS'95), Lecture Notes on Computer Science, vol. 955, August 1995, pp. 206–217.

[AG+98] N. Ascheuer, M. Grötschel, S. O. Krumke, and J. Rambau · *Combinatorial online optimization* · Proceedings of the International Conference of Operations Research (OR'98), Springer, 1998, pp. 21–37.

[AK88] M. J. Atallah and S. R. Kosaraju · *Efficient solutions to some transportation problems with applications to minimizing robot arm travel* · SIAM Journal on Computing **17** (1988), no. 5, 849–869.

[AKR98] N. Ascheuer, S. O. Krumke, and J. Rambau · *Competitive scheduling of elevators* · Preprint SC 98-34, Konrad-Zuse-Zentrum für Informationstechnik Berlin, November 1998, An improved version appears as [AKR00].

[AKR00] N. Ascheuer, S. O. Krumke, and J. Rambau · *Online dial-a-ride problems: Minimizing the completion time* · Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science (STACS'00), Lecture Notes on Computer Science, vol. 1770, Springer, 2000, pp. 639–650.

[AL97] S. Arora and C. Lund · *Hardness of approximations* · in [Hoc97], 1997.

[AMN98] E. M. Arkin, J. S. B. Mitchell, and G. Narasimhan · *Resource-constrained geometric network optimization* · Proceedings of the ACM Symposium on Computational Geometry, June 1998, pp. 307–316.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin · *Network flows* · Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[Arc00] A. Archer · *Two* $O(\log^* k)$*-approximation algorithms for the asymmetric* k*-center problem* · talk given at 17th International Symposium on Mathematical Programming (ISMP 2000), Atlanta GA, August 2000.

[ARR98] S. Arora, P. Raghavan, and S. Rao · *Approximation schemes for Eucledian* k*-medians and related problems* · Proceedings of the 30th Annual ACM Symposium on the Theory of Computing (STOC'98), 1998, pp. 106–113.

[BCV95] A. Blum, P. Chalasani, and S. Vempala · *A constant-factor approximation for the* k*-MST problem in the plane* · Proceedings of the 27th Annual ACM Symposium on the Theory of Computing (STOC'95), 1995, pp. 294–302.

[Ber92] O. Berman · *Improving the location of minisum facilities through network modification* · Annals of Operations Research **40** (1992), 1–16.

[BJG00] J. Bang-Jensen and G. Gutin · *Digraphs: Theory, algorithms and applications* · Springer, 2000.

[BKZ00] R. E. Burkard, B. Klinz, and J. Zhang · *Bottleneck capacity expansion problems with general budget constraints* · Tech. Report SFB-Report 189, Institute of Mathematics, TU Graz, March 2000.

[BLW87] M. W. Bern, E. L. Lawler, and A. L. Wong · *Linear-time computation of optimal subgraphs of decomposable graphs* · Journal of Algorithms **8** (1987), 216–235.

[BRV96] A. Blum, R. Ravi, and S. Vempala · *A constant factor approximation for the* k*-MST problem* · Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC'96), 1996, pp. 442–448.

[BRV99] A. Blum, R. Ravi, and S. Vempala · *A constant-factor approximation algorithm for the* k*-MST problem* · Journal of Computer and System Sciences **58** (1999), 101–108.

[CD+00] R. D. Carr, S. Doddi, G. Konjevod, and M. Marathe · *On the red-blue set cover problem* · Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000), 2000, pp. 345–353.

[CG99a] M. Charikar and S. Guha · *Improved combinatorial algorithms for facility location and* k*-median problems* · 40th Annual Symposium on Foundations of Computer Science (FOCS'99), 1999, pp. 378–388.

[CG+99b] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys · *A constant factor approximation algorithm for the* k*-median problem* · Proceedings of the 31st Annual ACM Symposium on the Theory of Computing (STOC'99), 1999.

[CGR98] S. Chaudhuri, N. Garg, and R. Ravi · *The* p*-neighbor* k*-center problem* · Information Processing Letters **65** (1998), no. 3, 131–134.

[CGS98] S. Chopra, I. Gilboa, and S. T. Sastry · *Source sink flows with capacity installation in batches* · Discrete Applied Mathematics **85** (1998), 165–192.

[Cha00] B. Chazelle · *A minimum spanning tree algorithm with inverse-ackermann type complexity* · Journal of the ACM (2000), no. 47, 1028–1047.

[Chr76] N. Christofides · *Worst-case analysis of a new heuristic for the traveling salesman problem* · Tech. report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.

[CK] P. Crescenzi and V. Kann · *A compendium of NP optimization problems* · http://www.nada.kth.se/theory/compendium/ , snapshot version appeared in [AC+99].

[CL97] R.-S. Chang and S.-J. Leu · *The minimum labeling spanning trees* · Information Processing Letters **63** (1997), 277–282.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest · *Introduction to algorithms* · MIT Press, 1990.

[CM+86] P. M. Camerini, F. Maffioli, S. Martello, and P. Toth · *Most and least uniform spanning trees* · Discrete Applied Mathematics **15** (1986), 181–187.

[CR98] M. Charikar and B. Raghavachari · *The finite capacity dial-a-ride problem* · Proceedings of the 39th Annual IEEE Symposium on the Foundations of Computer Science (FOCS'98), 1998.

[Dem00] I. Demgensky · *Netzwerkoptimierung – Flussausbauprobleme, Flusskostensenkungsprobleme (network optimization by augmenting capacities and decreasing flow costs, in German)* · Diploma Thesis, University of Würzburg, January 2000.

[DF85] M. E. Dyer and A. M. Frieze · *A simple heuristic for the* p*-center problem* · Operations Research Letters **3** (1985), no. 6, 285–288.

[Die96] R. Diestel · *Graphentheorie* · Springer, 1996.

[Dij59] E. W. Dijkstra · *A note on two problems in connexion with graphs* · Numerische Mathematik **1** (1959), 269–271.

[DK99] Y. Dodis and S. Khanna · *Designing networks with bounded pairwise distance* · Proceedings of the 31st Annual ACM Symposium on the Theory of Computing (STOC'99), 1999, pp. 750–759.

[DNW00] I. Demgensky, H. Noltemeier, and H.-C. Wirth · *Optimizing cost flows by modifying arc costs and capacities* · Proc. 26th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'00), Lecture Notes in Computer Science, vol. 1928, Springer, 2000, pp. 116–126.

[DS99] I. Dinur and S. Safra · *On the hardness of approximating label cover* · Electronic Colloquium on Computational Complexity (ECCC), TR 99–015, Tel Aviv University, April 1999.

[DST87] M. Dror, H. Stern, and P. Trudeau · *Postman tour on a graph with precedence relation on the arcs* · Networks **17** (1987), 283–294.

[Duf65] R. J. Duffin · *Topology of series-parallel networks* · Journal of Mathematical Analysis and Applications **10** (1965), 303–318.

[EG00] M. Ehrgott and X. Gandibleux · *An annotated bibliography of mulitobjective combinatorial optimization* · Report in Wirtschaftsmathematik Nr. 62/2000, Universität Kaiserslautern, April 2000.

[Ehr00] M. Ehrgott · *Multicriteria optimization* · Springer, Berlin, 2000.

[EJ73] J. Edmonds and E. L. Johnson · *Matching, Euler tours and the Chinese postman* · Mathematical Programming **5** (1973), 88–124.

[ESW98] S. Eidenbenz, C. Stamm, and P. Widmayer · *Positioning guards at fixed height above a terrain – an optimum inapproximability result* · Proceedings of the 6th Annual European Symposium on Algorithms, Lecture Notes in Computer Science, vol. 1461, 1998, pp. 187–198.

[ET76] K. P. Eswaran and R. E. Tarjan · *Augmentation problems* · SIAM Journal on Computing **10** (1976), no. 2, 270–283.

[Fei96] U. Feige · *A threshold of* $\ln n$ *for approximating set cover* · Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC'96), 1996, pp. 314–318.

[FG93] G. N. Frederickson and D. J. Guan · *Nonpreemptive ensemble motion planning on a tree* · Journal of Algorithms **15** (1993), 29–60.

[FHK78] G. N. Frederickson, M. S. Hecht, and C. E. Kim · *Approximation algorithms for some routing problems* · SIAM Journal on Computing **7** (1978), no. 2, 178–193.

[Fre79] G. N. Frederickson · *Approximation algorithms for some postman problems* · Journal of the ACM **26** (1979), no. 3, 538–554.

[FS01] E. Feuerstein and L. Stougie · *On-line single server dial-a-ride problems* · Theoretical Computer Science (2001), To appear.

[FSO96] G. N. Frederickson and R. Solis-Oba · *Increasing the weight of minimum spanning trees* · Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'96), January 1996, pp. 539–546.

[FV94] C. A. Floudas and V. Visweswaran · *Quadratic optimization* · in [HP94], 1994.

[GCF99] B. Gendron, T. G. Crainic, and A. Frangioni · *Multicommodity capacitated network design* · in [SS99], 1999.

[GG+94] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, E. Tardos, and D. P. Williamson · *Improved approximation algorithms for network design problems* · Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'94), January 1994, pp. 223–232.

[GJ79] M. R. Garey and D. S. Johnson · *Computers and intractability (a guide to the theory of NP-completeness)* · W.H. Freeman and Company, New York, 1979.

[GK99a] S. Guha and S. Khuller · *Greedy strikes back: Improved facility location algorithms* · Journal of Algorithms **31** (1999), no. 1, 228–248.

[GK99b] S. Guha and S. Khuller · *Improved methods for approximating node weighted Steiner trees and connected dominating sets* · Information and Computation **150** (1999), 57–74.

[GKR98] N. Garg, G. Konjevod, and R. Ravi · *A polylogarithmic approximation algorithm for the group Steiner tree problem* · Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'98), 1998.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver · *Geometric algorithms and combinatorial optimization* · Springer-Verlag, Berlin Heidelberg, 1988.

[GM97] M. Goldwasser and R. Motwani · *Intractability of assembly sequencing: Unit disks in the plane* · Proceedings of the Workshop on Algorithms and Data Structures (WADS'97), Lecture Notes on Computer Science, vol. 1272, 1997, pp. 307–320.

[GS88] Z. Galil and B. Schieber · *On finding most uniform spanning trees* · Discrete Applied Mathematics **20** (1988), 173–175.

[Gua98] D. J. Guan · *Routing a vehicle of capacity greater than one* · Discrete Applied Mathematics **81** (1998), no. 1, 41–57.

[GW95] M. X. Goemans and D. P. Williamson · *A general approximation technique for constrained forest problems* · SIAM Journal on Computing **24** (1995), no. 2, 296–317.

[GW97] M. X. Goemans and D. P. Williamson · *The primal-dual method for approximation algorithms and its application to network design problems* · in [Hoc97], 1997.

[Har72] F. Harary · *Graph theory* · Addison-Wesley Publishing Company, Inc., 1972.

[HK+99] D. Hauptmeier, S. O. Krumke, J. Rambau, and H.-C. Wirth · *Euler is standing in line* · Proc. 25th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'99), Lecture Notes in Computer Science, vol. 1665, Springer, 1999, pp. 42–54.

[HK+01] D. Hauptmeier, S. O. Krumke, J. Rambau, and H.-C. Wirth · *Euler is standing in line. Dial a ride problems with precedence constraints* · Discrete Applied Mathematics (2001), to appear.

[HL+91] J. M. Ho, D. T. Lee, C. H. Chang, and C. K. Wong · *Minimum diameter spanning trees and related problems* · SIAM Journal on Computing **20** (1991), 987–997.

[Hoc97] D. S. Hochbaum (ed.) · *Approximation algorithms for NP-hard problems* · PWS Publishing Company, Boston, 1997.

[HP94] R. Horst and P. M. Pardalos (eds.) · *Handbook of global optimization* · Kluwer Academic Publishers, 1994.

[HS85] D. S. Hochbaum and D. B. Shmoys · *A best possible heuristic for the* k*-center problem* · Mathematics of Operations Research **10** (1985), no. 2, 180–184.

[HT84] D. Harel and R. E. Tarjan · *Fast algorithms for finding nearest common ancestors* · SIAM Journal on Computing **13** (1984), no. 2, 338–355.

[HT95] R. Hassin and A. Tamir · *On the minimum diameter spanning tree problem* · Information processing letters **53** (1995), 109–111.

[HT97] S. E. Hambrush and H.-Y. Tu · *Edge weight reduction problems in directed acyclic graphs* · Journal of Algorithms **24** (1997), 66–93.

[JK+99] R. Jacob, G. Konjevod, S. O. Krumke, M. Marathe, R. Ravi, and H.-C. Wirth · *The minimum label path problem* · Unpublished manuscript, Los Alamos National Laboratory, 1999.

[JV99] K. Jain and V. Vazirani · *Primal-dual approximation algorithms for metric facility location and* k*-median problems* · Manuscript, March 1999. Available at `http://www.cc.gatech.edu/fac/Vijay.Vazirani/k-median.ps` , 1999.

[JV00] K. Jain and V. V. Vazirani · *An approximation algorithm for the fault tolerant metric facility location problem* · Third International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2000), Lecture Notes in Computer Science, vol. 1913, Springer, 2000, pp. 177–183.

[KH79a] O. Kariv and S. L. Hakimi · *An algorithmic approach to network location problems, part I: The* p*-center* · SIAM Journal of Applied Mathematics **37** (1979), 513–537.

[KH79b] O. Kariv and S. L. Hakimi · *An algorithmic approach to network location problems, part II:* p*-medians* · SIAM Journal of Applied Mathematics **37** (1979), 539–560.

[KM+98a] S. O. Krumke, M. V. Marathe, H. Noltemeier, R. Ravi, and S. S. Ravi · *Approximation algorithms for certain network improvement problems* · Journal of Combinatorial Optimization **2** (1998), no. 3, 257–288.

[KM+98b] S. O. Krumke, M. V. Marathe, H. Noltemeier, R. Ravi, and S. S. Ravi · *Network improvement problems* · in [PD98], 1998, pp. 247–268.

[KM+99a] S. O. Krumke, M. V. Marathe, H. Noltemeier, R. Ravi, S. S. Ravi, R. Sundaram, and H. C. Wirth · *Improving minimum cost spanning trees by upgrading nodes* · Journal of Algorithms **33** (1999), no. 1, 92–111.

[KM+99b] S. O. Krumke, M. V. Marathe, H. Noltemeier, R. Ravi, S. S. Ravi, R. Sundaram, and H. C. Wirth · *Improving spanning trees by upgrading nodes* · Theoretical Computer Science **221** (1999), no. 1–2, 139–156.

[KM+00] S. O. Krumke, M. V. Marathe, C. Phillips, and E. Sundberg · *A new algorithm for the network inhibition problem* · talk given at 17th International Symposium on Mathematical Programming (ISMP 2000), Atlanta GA, August 2000.

[KM+01] S. O. Krumke, M. V. Marathe, H. Noltemeier, S. S. Ravi, and H.-C. Wirth · *Upgrading bottleneck constrained forests* · Discrete Applied Mathematics **108** (2001), 129–142.

[KMR97] D. Karger, R. Motwani, and G. D. S. Ramkumar · *On approximating the longest path in a graph* · Algorithmica **18** (1997), 82–98.

[KN+98] S. O. Krumke, H. Noltemeier, S. S. Ravi, M. V. Marathe, and K. U. Drangmeister · *Modifying networks to obtain low cost subgraphs* · Theoretical Computer Science **203** (1998), no. 1, 91–121.

[KN+99] S. O. Krumke, H. Noltemeier, R. Ravi, S. Schwarz, and H.-C. Wirth · *Flow improvement and flows with fixed costs* · Proceedings of the International Conference of Operations Research Zürich (OR'98), Editors: H.-J. Lüthi and P. Kall, Operations Research Proceedings, Springer, 1999.

[KPR00] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman · *Analysis of a local search heuristic for facility location problems* · Journal of Algorithms **37** (2000), 146–188.

[KPS00] S. Khuller, R. Pless, and Y. Sussmann · *Fault tolerant* k*-center problems* · Theoretical Computer Science **242** (2000), no. 1–2, 237–245.

[KR95] P. Klein and R. Ravi · *A nearly best-possible approximation for node-weighted Steiner trees* · Journal of Algorithms **19** (1995), 104–115.

[Kru56] J. B. Kruskal · *On the shortest spanning subtree of a graph and the traveling salesman problem* · Proceedings of the American Mathematical Society **7** (1956), 48–50.

[Kru95] S. O. Krumke · *On a generalization of the* p*-center problem* · Information Processing Letters **56** (1995), 67–71.

[KW98] S. O. Krumke and H.-C. Wirth · *On the minimum label spanning tree problem* · Information Processing Letters **66** (1998), no. 2, 81–85.

[KZ97] Marek Karpinski and Alexander Zelikovski · *New approximation algorithms for Steiner tree problems* · Journal of Combinatorial Optimization **1** (1997), 1–19.

[Law76] E. L. Lawler · *Combinatorial optimization: Networks and matroids* · Holt, Rinehart and Winston, 1976.

[LV92] J.-H. Lin and J. S. Vitter · *Approximation algorithms for geometric median problems* · Information Processing Letters **44** (1992), 245–249.

[LY93] C. Lund and M. Yannakakis · *On the hardness of approximating minimization problems* · Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (STOC'93), May 1993, pp. 286–293.

[MF90] P. B. Mirchandani and R. L. Francis · *Discrete location theory* · Wiley-Interscience Series in Discrete Mathematics and Optimization, 1990.

[Mit96] J. S. B. Mitchell · *Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric* k*-MST problem* · Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'96), 1996, pp. 402–408.

[Mit98] J. S. B. Mitchell · *Guillotine subdivisions approximate polygonal subdivisions: Part II – a simple polynomial-time approximation scheme for geometric TSP,* k*-MST, and related problems* · SIAM Journal on Computing (1998).

[Mit00] J. S. B. Mitchell · *Geometric shortest paths and network optimization* · 2000, in: [SU00].

[MM93] T. L. Magnanti and P. Mirchandani · *Shortest paths, single origin-destination network design and associated polyhedra* · Networks **23** (1993), no. 2, 103–121.

[MN99] K. Mehlhorn and S. Näher · *The LEDA platform of combinatorial and geometric computing* · Cambridge University Press, 1999.

[MR+95] M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III · *Bicriteria network design problems* · Proceedings of the 22nd International Colloquium on Automata, Languages and Programming (ICALP'95), Lecture Notes in Computer Science, vol. 944, 1995, pp. 487–498.

[MR+98] M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III · *Bicriteria network design problems* · Journal of Algorithms **28** (1998), no. 1, 142–171.

[NMK99] H. Noltemeier, C. Mauckner, and A. Kaußner · *Polynomial time approximation schemes for the median tour problem and the newspaper delivery problem in the eucledian plane* · Tech. report, University of Würzburg, Germany, 1999, To appear.

[Nol76] H. Noltemeier · *Graphentheorie: mit Algorithmen und Anwendungen* · de Gruyter Lehrbuch, 1976.

[Pap94] C. M. Papadimitriou · *Computational complexity* · Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1994.

[PD98] P. M. Pardalos and D. Du (eds.) · *Network design: Connectivity and facilities location* · AMS-DIMACS Volume Series on Discrete Mathematics and Computer Science, vol. 40, American Mathematical Society, 1998.

[Phi93] C. Phillips · *The network inhibition problem* · Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (STOC'93), May 1993, pp. 776–785.

[Ple81] J. Plesník · *The complexity of designing a network with minimum diameter* · Networks **11** (1981), 77–85.

[PS82] C. H. Papadimitriou and K. Steiglitz · *Combinatorial optimization* · Prentice-Hall, Inc., 1982.

[PS95] D. Paik and S. Sahni · *Network upgrading problems* · Networks **26** (1995), 45–58.

[PY91] C. H. Papadimitriou and M. Yannakakis · *Optimization, approximation, and complexity classes* · Journal of computer and system sciences **43** (1991), 425–440.

[SC+98] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian · *Buy-at-bulk network design: Approximating the single-sink edge installation problem* · Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'98), 1998, pp. 619–628.

[Sch95] B. Schoenmakers · *A new algorithm for the recognition of series parallel graphs* · Tech. Report CS-R9504, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, 1995.

[Seg87] A. Segev · *The node-weighted Steiner tree problem* · Networks **17** (1987), 1–18.

[Shm00] D. B. Shmoys · *Approximation algorithms for facility location problems* · Third International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2000), Lecture Notes in Computer Science, vol. 1913, Springer, 2000, pp. 27–33.

[SS99] B. Sanso and P. Soriano (eds.) · *Telecommunications network planning* · Kluwer Academic Publishers, Boston, 1999.

[STA97] D. Shmoys, E. Tardos, and K. Aardal · *Approximation algorithms for facility location problems* · Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC'97), 1997, pp. 265–274.

[Ste99] J. Steffan · *Umladeprobleme* · Diplomarbeit, Universität Würzburg, Lehrstuhl Informatik I, Januar 1999.

[SU00] J. R. Sack and J. Urrutia · *Handbook of computational geometry* · Elsevier Science, 2000.

[SV88] B. Schieber and U. Vishkin · *On finding lowest common ancestors: Simplification and parallelization* · SIAM Journal on Computing **17** (1988), no. 6, 1253–1262.

[Tar77] R. E. Tarjan · *Finding optimum branchings* · Networks **7** (1977), 25–35.

[Tho97] M. Thorup · *Undirected single source shortest paths in linear time* · Proceedings of the 38th Annual IEEE Symposium on the Foundations of Computer Science (FOCS'97), 1997, pp. 426–435.

[Vis97] S. Vishwanathan · *An* $O(\log^*)$ *approximation algorithm for the asymmetric* p*-center problem* · Indian Institute of Technology, Bombay, 1997.

[VTL82] J. Valdes, R. E. Tarjan, and E. L. Lawler · *The recognition of series-parallel digraphs* · SIAM Journal on Computing **11** (1982), no. 2, 298–313.

[WCT00] B. Y. Wu, K.-M. Chao, and C. Y. Tang · *Approximation algorithms for the shortest total path length spanning tree problem* · Discrete Applied Mathematics **105** (2000), 273–289.

[WL+98] B. Y. Wu, G. Lancia, V. Bafna, K.-M. Chao, R. Ravi, and C. Y. Tang · *A polynomial time approximation scheme for minimum routing cost spanning trees* · Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'98), 1998.

[Woe99] G. J. Woeginger · *When does a dynamic programming formulation guarantee the existence of an FPTAS?* · Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99), 1999, pp. 820–829.

[WS99] H.-C. Wirth and J. Steffan · *On minimum diameter spanning trees under reload costs* · Proc. 25th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'99), Lecture Notes in Computer Science, vol. 1665, Springer, 1999, pp. 78–88.

[WS01] H.-C. Wirth and J. Steffan · *Reload cost problems: Minimum diameter spanning tree* · Discrete Applied Mathematics (2001), to appear.

[WW85] K. Wagner and G. Wechsung · *Computational complexity* · Mathematics and its applications, D. Reidel Publishing Company, Kluwer Academic Publishers, 1985.

Index

If there is more than one reference, the underlined page number refers to the definition of the entry.

F

G

H

I

J

K

L

Q

R

V

W