Complexity of Bulk-Robust Combinatorial Optimization Problems

Dissertation
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

Der Fakultät für Mathematik der
Technischen Universität Dortmund
vorgelegt von

Felix Hommelsheim

am 27.07.2020

Dissertation

Complexity of Bulk-Robust Combinatorial Optimization Problems

Fakultät für Mathematik
Technische Universität Dortmund

Erstgutachter: Prof. Dr. Christoph Buchheim

Zweitgutachter: Prof. Dr. Sebastian Stiller

Tag der mündlichen Prüfung: 25.09.2020

Acknowledgment

There are numerous co-supervisors who guided me during my PhD and it was a pleasure to work with all of them. First, I would like to thank my co-supervisor Christoph Buchheim for his advice and support. I am also grateful to David Adjiashvili, Dennis Michaels, Moritz Mühlenthaler, and Oliver Schaudt who co-supervised me during my PhD and were always approachable for my problems and questions. Also most of the results of this thesis were established by the joint work with David Adjiashvili, Moritz Mühlenthaler, and Oliver Schaudt. The fruitful discussions with my second examiner Sebastian Stiller are also very appreciated. The time at the TU Dortmund was wonderful and I would like to thank all colleagues for creating a great working-atmosphere.

The support of my work by the German Research Foundation (DFG) within the Research Training Group ”Discrete Optimization of Technical Systems under Uncertainty” is gratefully acknowledged. Parts of this thesis were carried out during my research stay at the Institute for Operations Research (IFOR) at the ETH Zurich. It was a fantastic time in Zurich and the hospitality at the IFOR was great.

Furthermore, I am grateful to all my friends for supporting me during my PhD and proof-reading parts of my thesis. Finally, I would like to thank my family and Svenja Griesbach for their absolute support.

Abstract

This thesis studies three robust combinatorial optimization problems on graphs. Most robust combinatorial optimization problems assume that the cost of the resources, e.g. edges in a graph, are uncertain. In this thesis, however, we study the so called *bulk-robust* approach which models the uncertainty in the underlying combinatorial structure. To be more precise, in the bulk-robust model we are given an explicit list of scenarios comprising a set of resources each, which fail if the corresponding scenario materializes. Additionally, we are given a property that we have to obtain such as 'containing a perfect matching', 's-t-connectivity', or 'being connected', which may arise from a fundamental combinatorial optimization problem. The goal of the bulk-robust optimization problem is to find a minimum weight set of resources such that the desired property is satisfied no matter which scenario materializes, i.e. no matter which set of resources from the list is removed.

We study the bulk-robust bipartite matching problem, the bulk-robust k-edge disjoint s-t-paths problem, and the bulk-robust minimum spanning tree problem. We investigate the complexity of the three problems and show that most of them are hard to approximate even if the list of scenarios consists of singletons only. We complement these inapproximability results with polynomial-time approximation algorithms that essentially match the hardness results. Furthermore, we present FPT and XP algorithms and consider special graph classes that allow for a polynomial-time exact algorithm.

Zusammenfassung

In dieser Arbeit werden drei robuste kombinatorische Optimierungsprobleme auf Graphen behandelt. Die meisten robusten kombinatorischen Optimierungsprobleme nehmen an, dass die Kosten der Ressourcen, z.B. Kanten in einem Graph, unsicher sind. Diese Arbeit jedoch behandelt den Ansatz der sogenannten bulk-Robustheit, die Unsicherheiten in der kombinatorischen Struktur widerspiegelt. Genauer gesagt ist im bulk-robusten Modell eine explizite Liste von Szenarien gegegeben und jedes dieser Szenarien enthält eine Menge von Ressourcen, die ausfällt, wenn das entsprechende Szenario eintritt. Zusätzlich ist noch eine Eigenschaft gegeben, die wir erzeugen müssen. Diese Eigenschaft könnte beispielsweise sein, dass der resultierende Graph ein perfektes Matching enthält, zwei bestimmte Knoten miteinander verbindet oder zusammenhängend ist. Das Ziel des bulk-robusten Optimierungsproblems ist es, eine kosten-minimale Menge von Ressourcen zu finden, sodass die gewünschte Eigenschaft erfüllt ist, egal welches Szenario eintrifft, d.h. unabhängig davon, welche Menge von Resourcen der Liste von der Lösung entfernt wird.

Diese Arbeit behandelt das bulk-robuste bipartite Matchingproblem, das bulk-robuste k-Kanten disjunkte s-t-Wege Problem und das bulk-robuste minimale Spannbaum Problem. Wir beschäftigen uns mit der Komplexität der drei Probleme und zeigen, dass die meisten sogar dann schwer zu approximieren sind, wenn die Liste der Szenarien nur aus einelementigen Mengen besteht. Wir komplementieren diese nicht-Approximierbarkeitsresultate mit effizienten Approximationsalgorithmen, deren Güte im Wesentlichen den hardness-Resultaten entspricht. Darüber hinaus präsentieren wir FPT und XP Algorithmen und entwickeln effiziente exakte Algorithmen für spezielle Graphklassen.

Contents

Chapter 1

Introduction

Many real-life decision making problems that we encounter in our daily life are *combinatorial optimization problems*. For example, on our way to work we would like to take the fastest route from our home to our working place. This problem can be modeled as a combinatorial optimization problem by a *shortest path problem* in a graph. Arrived at our working place, we might need to schedule the work plan of the employees in our company, i.e. we need to assign tasks to employees. Such a scheduling problem can be modeled as a *matching problem* in a so called bipartite graph – another combinatorial optimization problem. Many classic combinatorial optimization problems such as the shortest path problem and the matching problem have been investigated exhaustively and there are numerous algorithms that solve these problems efficiently.

However, all those combinatorial optimization problems do not consider that the input data of real-world applications may be *uncertain*. In fact, almost all data that we encounter in our daily life is subject to uncertainty: a traffic jam on the highway on our way to work, an employee gets sick and cannot perform the scheduled job or an adversary who attacks our computer infrastructure are just a few examples where we encounter uncertainty.

There is a vast amount of literature addressing optimization under uncertainty. Most of the different models can be divided either into *stochastic optimization* or *robust optimization*. Stochastic models incorporate probabilities to represent uncertainty underlying the input data. The decision maker then usually needs to find a solution that optimizes a certain probabilistic function *in expectation.* Clearly, such models need to be fed with probabilities for all possible scenarios and these probabilities themselves are again uncertain in real-life problems. However, an optimal solution to some stochastic optimization problem is good in the average case, since it optimizes some kind of expected value, even though there might be some realizations in which the solution performs very poorly.

Exactly this very poor performance for some (possibly very unlikely) scenarios is addressed in robust optimization. In robust optimization, the decision maker needs to optimize the worst possible outcome, that is, he needs to optimize the worst-case scenario over all possible scenarios. This phenomenon can be interpreted by a so called *adversary.* After the decision maker selected the solution, an adversary picks the worst-case scenario over all possible realizations of the data set, resulting in the worst possible outcome for the decision maker. Clearly, such models are very conservative and, in the average case, might perform worse compared to a solution that optimizes some expected value. However, using a robust model allows the decision maker to give a certain guarantee on the value of the computed solution, which is not possible when optimizing the expected value.

In this thesis we study robust combinatorial optimization problems on graphs. The field of robust combinatorial optimization itself can be roughly divided into two fields: Uncertainty in the *cost* of the underlying resources and uncertainty in the *structure* of the problem, affecting the set of feasible solutions. Furthermore, there are many different ways in modeling the uncertainty of a robust optimization problem, especially in the case when the cost are uncertain. For example, the possible realizations of the cost vector could be restricted to some kind of polyhedral, ellipsoidal or discrete set. In this thesis, however, we only focus on discrete uncertainty, that is, we are given a finite set of possible realizations of the input data. We demonstrate this with an example.

Consider the problem of finding a minimum-cost subgraph of some graph $G = (V, E)$ that is connected and spanning (i.e. finding a minimum spanning tree in G) for some cost function $c : E \to \mathbb{Q}_{\geq 0}$. In the *cost-robust counterpart* the cost of the resources are uncertain. That is, we are given $k \in \mathbb{N}$ scenarios and each scenario describes a different realization of the cost function c. To be more precise, we are given a set of k cost functions $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$ with $c : E \to \mathbb{Q}_{\geq 0}$ for each $c \in \mathcal{C}$. Let $\mathcal{Y}$ contain all subsets X of E such that (V, X) is a spanning tree of G. The task is to solve the following optimization problem:

$$\min_{X \in \mathcal{Y}} \max_{c \in \mathcal{C}} c(X)$$

Thus, the task is to find a connected spanning subgraph which minimizes the worst-case cost over all possible realizations of the cost functions.

To the contrary, uncertainty in the structure of the problem affects the set of feasible solutions. Again, consider the above example of finding a minimum-cost subgraph of some graph $G = (V, E)$ that is connected and spanning for some cost function $c : E \to \mathbb{Q}$. In the *structural-robust counterpart* the scenario set $\mathcal{U}$ describes sets of edges that can fail after the solution has been computed. To be more precise, $\mathcal{U}$ is a family of subsets of edges, i.e. for each $F \in \mathcal{U}$ we have $F \subseteq E$. Then the task is to solve the following optimization problem:

$$\min_{X \subseteq E} \{c(X) \mid (V, X \setminus F) \text{ contains a spanning tree of } G \text{ for each } F \in \mathcal{U}\}$$

Thus, the task is to find a connected spanning subgraph that *contains* a spanning tree, no matter which set of edges F of $\mathcal{U}$ is removed from the solution. The concept of structural robustness presented here is only one of many possible variants to address structural uncertainty. It is called *bulk-robustness* and was introduced by Adjiashvili, Stiller and Zenklusen [ASZ15].

In this thesis we investigate bulk-robust counterparts for classic combinatorial optimization problems on graphs. Before bulk-robustness was introduced, most structural-robust counterparts of combinatorial optimization problems assumed a uniform failure model. Concretely, in such a model any subset of resources A of size at most k is subject to failure. A typical example is the k-edge-connected spanning subgraph problem, in which the goal is to obtain a minimum-cost connected spanning subgraph, no matter which $k - 1$ edges fail. In contrast, the bulk-robust model is on the far opposite site of the uniformity level. In the bulk-robust model we are given a precise list $\mathcal{U} \subseteq 2^A$ of scenarios comprising a set of resources each. Additionally, we are given a property $\mathcal{X}$ that we wish to obtain such as 'containing a perfect matching', 's-t-connectivity' or 'being connected', which may arise from a fundamental combinatorial optimization problem. The goal of the bulk-robust optimization problem is to find a minimum cost set X of resources A such that the desired property is satisfied no matter which scenario materializes, i.e. no matter which set of resources from the list is removed. Put differently, for a given cost function $c \in \mathbb{Z}^A_{\geq 0}$ we are given the following problem:

$$\min_{X \subseteq A} \{c(X) \mid X \setminus F \text{ has property } \mathcal{X} \text{ for every } F \in \mathcal{U}\}$$

Adjiashvili, Stiller and Zenklusen considered the properties 's-t-connectivity in a graph' and 'containing a basis of a matroid'. Among other things they showed that the problem is as hard to approximate as Set Cover and thus does not allow for a sublogarithmic-factor approximation unless $\mathsf{P} = \mathsf{NP}$. This difficulty does not originate from the combinatorial structure of the underlying property $\mathcal{X}$ but only from the combinatorial structure of the scenario list $\mathcal{U}$.

To overcome this issue, the main focus of this thesis is on bulk-robust problems in which the scenario list $\mathcal{U}$ consists of singletons only such that the difficulty of the problems does not originate only from the difficult combinatorial structure of $\mathcal{U}$. In particular, the focus of this thesis is on bulk-robust problems with properties on graphs, namely 'containing a perfect matching', 'k disjoint paths between two given vertices s and t' or 'containing a connected spanning subgraph', in which each set in $\mathcal{U}$ has size at most 1.

Our main focus in this thesis is the design of approximation algorithms, FPT algorithms, XP algorithms and polynomial-time exact algorithms for (variants of) all three mentioned problems. We complement these algorithmic results with NP-hardness and inapproximability results.

We will briefly discuss all three problems that are addressed in this thesis.

Robust Matchings. The first problem we consider is a variant of the bulk-robust version of the perfect matching problem and is formally defined as follows.

Problem 1.1 (Robust Matching Augmentation)**.** Given an undirected bipartite graph $G = (U + W, E)$ that admits a perfect matching, the task is to find a set $L \subseteq \overline{E}$ of minimum cardinality, such that the graph $G + L - e$ contains a perfect matching for each $e \in E$.

Here, by $\overline{E}$ we refer to the edges of the bipartite complement of E. Let us rephrase the problem. Assume we are given a perfect matching and some additional edges comprising some bipartite graph G. Then the task is to add a minimum-cardinality subset of edges L such that $G + L$ contains a perfect matching, even if an adversary is allowed to remove any single edge (only one at a time) of the perfect matching.

We consider both the unweighted (as in the above definition) and weighted version of the problem (we then associate certain cost with each edge in $\overline{E}$). In our main hardness results we show that both versions are notoriously hard to approximate. In the unweighted case the problem is as hard to approximate as Set Cover (implying that there is no polynomial-time sublogarithmic-factor approximation unless $\mathsf{P} = \mathsf{NP}$), while the weighted case is as hard to approximate as Directed Steiner Forest (implying that there is no polynomial-time $\log^{2-\varepsilon}(n)$-approximation for any $\varepsilon > 0$ unless $\mathsf{NP} \subseteq \mathsf{ZTIME}(n^{\mathsf{polylog}(n)})$).

For the unweighted case, we first complement this inapproximability result by a $\log_2(n)$-factor approximation algorithm, thus essentially matching the bound from above. Furthermore, we show that the problem is FPT when parameterized by treewidth and show that it is polynomial-time solvable in chordal bipartite graphs. In the weighted case, we establish a close connection to the problem Directed Steiner Forest to provide a polynomial-time $(1 + n^{\frac{1}{2}n+\varepsilon})$-approximation algorithm for every $\varepsilon > 0$. Finally, we prove a complexity dichotomy based on graph minors and show that the weighted problem, restricted to a class $\mathcal{T}$ of connected graphs closed under connected minors, is NP-complete if and only if $\mathcal{T}$ contains at least one of two simple graph classes (we define these classes in Chapter 3).

Robust Disjoint s-t-Paths. The second problem we consider is a bulk-robust version of the k-disjoint path problem and generalizes the bulk-robust shortest path problem. It is formally defined as follows.

Problem 1.2 (Bulk-robust k-Disjoint Paths)**.** We are given an (undirected or directed) graph $G = (V, E)$, two vertices $s, t \in V$, costs $c \in \mathbb{Z}^E_{\geq 0}$, an integer $k \in \mathbb{N}$ and a collection of interdiction sets $\mathcal{F} = \{F_1, F_2, \dots, F_p\}$ with $F_i \subseteq E$ for all $i \in \{1, 2, \dots, p\}$. The task is to find a set $X \subseteq E$ of minimum cost, such that the graph $G[X - F_i]$ contains k edge-disjoint s-t paths for every $i \in \{1, 2, \dots, p\}$.

The width ℓ of the set $\mathcal{F}$ is the size of a largest interdiction set in $\mathcal{F}$, i.e.

$\ell = \max_{F\in\mathcal{F}} |F|$. Note that for $k = 1$ we simply obtain the bulk-robust shortest path problem.

In this thesis, we present various results for different values of the parameters ℓ and k and distinguish between directed and undirected graphs. Here, we outline only a few of these results.

As mentioned before, the above problem generalizes the bulk-robust shortest path problem considered in [ASZ15] and therefore all inapproximability results also apply here, including SET COVER hardness for 'large' ℓ. Furthermore, for $\ell = 1$, it is not hard to see that BULK-ROBUST k-DISJOINT PATHS generalizes the weighted version of ROBUST MATCHING AUGMENTATION (for 'large' k). Hence, the hardness results from above also apply here.

We complement the hardness result for 'large' ℓ by an efficient $O(\log n)$-factor approximation algorithm for constant k and constant ℓ, which is based on the approximation algorithm presented in [ASZ15]. Furthermore, when $\ell = 1$ and k is constant, we develop a polynomial-time 2-approximation algorithm that is based on a polynomial-time exact algorithm for the augmentation problem. In a nutshell, in the augmentation problem we are given k-disjoint s-t paths for free and the task is to augment this edge set to a feasible solution (note that in this case only single edges can be removed). In fact, the polynomial-time exact algorithm is an XP algorithm when parameterized by k.

Robust Spanning Trees. The third problem we consider is a bulk-robust version of the minimum spanning tree problem and is formally defined as follows.

Problem 1.3 (FLEXIBLE GRAPH CONNECTIVITY)**.** We are given an undirected connected graph $G = (V, E)$, non-negative edge weights $w \in \mathbb{Q}^E_{\geq 0}$, and a set of edges $F \subseteq E$ called *unsafe* edges. Let $\overline{F} := E \setminus F$ be the *safe* edges.

The task is to compute a minimum-weight edge set $S \subseteq E$ with the property that $(V, S - f)$ is a connected spanning graph for every $f \in F$.

FLEXIBLE GRAPH CONNECTIVITY encapsulates several fundamental combinatorial optimization problems, such as the minimum-cost spanning tree problem, the minimum-cost 2-edge connected spanning subgraph problem and the weighted tree augmentation problem. As a direct consequence we obtain that FLEXIBLE GRAPH CONNECTIVITY is APX-hard.

First, we complement this result by a polynomial-time 2.523-approximation algorithm for general instances and a polynomial-time 2.404-approximation algorithm for bounded weight instances. A bounded weight instance is an instance in which all weights are in the range of $[1, M]$ for some constant M. Note that achieving the better approximation guarantee for bounded weight instances comes with an increased running time, which is exponential in M.

Furthermore, we study the unweighted case of Flexible Graph Connectivity in which we want to minimize the cardinality of S and provide a 1.5-approximation algorithm (in fact, we provide a more general result and refer to Chapter 5 for more details).

Outline. The next chapter contains basic concepts and notation of graphs and complexity theory that we use throughout the thesis. Furthermore, we present some results on important combinatorial optimization problems on graphs and give a brief overview of the robust combinatorial optimization literature under discrete uncertainty. Chapters 3–5 contain the main contributions of the thesis and comprise the results on all three problems discussed above (in the same order). Finally, Chapter 6 concludes the thesis.

The Chapters 3, 4 and 5 are more or less self-contained and do not require knowledge about a previous chapter (unless probably Chapter 2, where we introduce some notation used throughout the thesis).

Publications. An extended abstract of the contents of Chapter 3 has been published in [HMS19] and the full version is submitted to SIAM Journal on Discrete Mathematics. This was a joint work with Moritz Mühlenthaler and Oliver Schaudt. An extended abstract of the contents of Chapter 5 has been published in [AHM20] and a more detailed version is submitted to the journal Mathematical Programming, Series B. This was a joint work with David Adjiashvili and Moritz Mühlenthaler.

Chapter 2

Preliminaries and Related Work

This section introduces most of the terminology used throughout the thesis. However, some definitions of special terms will be given in the particular chapter. We mostly stick to the notation introduced by Diestel [Die12] and Schrijver [Sch03].

Most of the algorithmic results presented in this thesis are based on graphs. We will introduce basic graph theoretic terminology in the next section. Additionally, most of our results also require some basic knowledge of complexity theory, so we will give a brief introduction in Section 2.2. Section 2.3 contains definitions of fundamental combinatorial optimization problems on graphs as well as some results used throughout this work. Finally, in Section 2.4 we present some related work on robust combinatorial optimization.

2.1 Graph Theory

In this section we present the basic graph theoretic terminology. We mostly stick to the book of Diestel [Die12]. By $\mathbb{N}$ we denote the set of natural numbers, including zero. For a set $\{1, 2, \ldots, k\}$ of integers we sometimes simply write $[k]$ for short. A set $\mathcal{A} = \{A_1, \ldots A_n\}$ of disjoint subsets of a set A is a *partition* of A if the union $\bigcup \mathcal{A}$ of all the sets $A_i \in \mathcal{A}$ is A and $A_i \neq \emptyset$ for each $i \in [n]$. For disjoint sets U, W, we denote by $U + W$ their disjoint union.

Graphs. An *undirected graph* $G = (V, E)$ consists of a finite set of *vertices* V (sometimes also called *nodes*) and a finite set of *edges* E (sometimes also called *links*). An edge $e = \{u, v\}$ is a set of two vertices $u, v \in V$. In a *directed* graph, each edge has a direction, i.e. it has a *head* u and a *tail* v. In this case we write $e = (u, v)$ as a 2-tuple to clarify that the ordering of the vertices has to be considered. If it is clear from the context whether e is directed or undirected, we may simply write uv. In a directed graph, the edges are also sometimes called *arcs*. By $[V]^2$ we refer to the set of all pairs of vertices. Thus, in undirected graphs we can also write $E \subseteq [V]^2$.

A graph with vertex set V is said to be a graph *on* V. The vertex set of a graph G is referred to as $V(G)$ and its edge set as $E(G)$, independent of the actual names of these sets. We sometimes do not strictly distinguish between a graph and its vertex or edge set and write $v \in G$ instead of $v \in V(G)$. We usually set $n := |V(G)|$.

The *endpoints* (or sometimes *ends* for short) of an edge $e = \{u, v\}$ are the vertices u and v. If v is an endpoint of an edge e, we say that v is *incident* to e. Two vertices $u, v \in V$ are *adjacent* or *neighbors* if there is an edge $e \in E$ containing both u and v. Two edges $e \neq e'$ are adjacent if they have an endpoint in common. An edge e is called a *loop* if both of its endpoints are the same vertex. Two edges e, e' are *parallel* if both edges have the same endpoints. Non-adjacent vertices are *independent*. A graph without loops or parallel edges is called *simple*. Unless stated otherwise, graphs are loopless but may have parallel edges.

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. We call G and G' *isomorphic*, and write $G \simeq G'$ if there exists a bijection $\varphi : V \to V'$ with $uv \in E$ if and only if $\varphi(u)\varphi(v) \in E'$. Such a map φ is called an *isomorphism*. We do not normally distinguish between isomorphic graphs. Thus, we usually write $G = G'$ rather than $G \simeq G'$. A class of graphs that is closed under isomorphism is called a *graph property*.

We set $G \cup G' := (V \cup V', E \cup E')$ and $G \cap G' := (V \cap V', E \cap E')$. If $G \cap G' = \emptyset$, then G and G' are *disjoint*. If $V' \subseteq V$ and $E' \subseteq E$, then G' is a *subgraph* of G, written as $G' \subseteq G$. We may also sometimes say G *contains* G'. If $G' \subseteq G$ and G' contains all the edges $uv \in E$ with $u, v \in V'$, then G' is an *induced subgraph* of G, we then say that V' *induces* G' in G and write $G' =: G[V']$. Thus, if $U \subseteq V$ is any subset of vertices, then $G[U]$ denotes the graph on U whose edges are precisely the edges of G with both ends in U.

Finally, $G' \subseteq G$ is a *spanning subgraph* of G if V' spans all of G, i.e. $V' = V$.

For any set of vertices $U \subseteq V$ of G, we write $G - U$ for $G[V \setminus U]$. In other words, $G - U$ is obtained from G by *deleting* all vertices in $U \cap V$ and their incident edges. If $U = \{v\}$ is a singleton, we write $G - v$ rather than $G - \{v\}$. Similarly, for any subset $F \subseteq [V]^2$ we write $G - F := (V, E \setminus F)$ and $G + F := (V, E \cup F)$. As above, we write $G - e$ and $G + e$ for singletons. We call a graph *maximal* (or *minimal*) with respect to some graph property if G itself has the property but $G + uv$ (or $G - uv$) does not have this property for any two non-adjacent vertices $u, v \in V(G)$.

The *complement* $\overline{G}$ of a simple graph G is the graph on V with edge set $\overline{E} := [V]^2 \setminus E$. Similarly, for a directed graph $D = (V, A)$ we refer by $\overline{A}$ to the arcs not present in D. That is, we let $\overline{A} \subseteq (V \times V) \setminus A$.

The degree of a vertex. The set of neighbors of a vertex $v \in V(G)$ is denoted by $N_G(v)$, or $N(v)$ for short if G is clear from the context. More generally, for $U \subseteq V$, the neighbors in $V \setminus U$ of vertices in U are called *neighbors of* U, denoted by $N(U)$. The *degree* $d_G(v)$ (or $d(v)$ if G is clear from the context) of a vertex is the number of incident edges to v, i.e. $d(v) = |N(v)|$. A vertex of degree zero is called *isolated*. The number $\delta(G) := \min\{d(v) \mid v \in V\}$ is the *minimum degree* of G. Similarly, $\Delta(G) := \max\{d(v) \mid v \in V\}$ is the *maximum degree* of G.

Analogously to undirected graphs we define the notion of neighborhood in directed graphs. For a directed graph $D = (V, A)$ we define the set of outgoing neighbors of U by $N^+(U) := \{v \in V \setminus U \mid (u, v) \in A \text{ for some } u \in U\}$ and the set of incoming neighbors of U by $N^-(U) := \{v \in V \setminus U \mid (v, u) \in A \text{ for some } u \in U\}$.

Paths and Cycles. A *path* is a non-empty graph $P = (V, E)$ of the form $V = \{v_1, v_2, \dots, v_k\}$ and $E = \{v_1v_2, v_2v_3, \dots, v_{k-1}v_k\}$, where all vertices are distinct, i.e. $v_i \neq v_j$ for $i \neq j$. The vertices v_1 and v_k are called the *end vertices* of P. If P is a *directed path* we call v_0 the *start vertex* of P. The *length* of a path is defined by its number of edges. Note that a path might have length zero. In simple graphs we often simply refer to a path by its sequence of vertices, writing $P = v_1v_2 \dots v_k$. We then say that P is a path *from* v_1 *to* v_k.

For the subpath $v_i, v_{i+1}, \dots, v_{j-1}, v_j$ of P we sometimes write v_iPv_j for short.

A *walk* (of length k) in a graph G is a non-empty alternating sequence $v_1e_1v_2e_2 \dots e_{k-1}v_k$ of vertices and edges in G such that $e_i = \{v_i, v_{i+1}\}$ for all $i = 1, \dots, k-1$. If $v_1 = v_k$, the walk is *closed*, otherwise *open*.

Given two sets of vertices $A, B \subseteq V$ we call $P = v_1, v_2, \dots, v_k$ an *A-B path* if $V(P) \cap A = \{v_1\}$ and $V(P) \cap B = \{v_k\}$. As before, for singletons we simply write a-B path instead of $\{a\}$-B path. In particular, for two vertices $s, t \in V$ an *s-t path* is a graph with endpoints s and t. If the graph is directed, the edges of a path P with endpoints s and t need to be directed in the direction of P, i.e. from s to t. Then s is the start vertex and t is the end vertex of P.

If $P = v_1v_2 \dots v_k$ is a path and $k \geq 2$, then the graph $C := P + v_kv_0$ is called a *cycle*. Similar to paths we simply write $C = v_1v_2 \dots v_kv_1$ if G is simple. The length of a cycle is its number of edges. An edge that is incident to two vertices of a cycle but is not itself an edge of the cycle is called a *chord* of that cycle. A cycle in G forming an induced subgraph is called an *induced cycle* in G. This cycle has no chords. A cycle that contains all vertices is called a *Hamilton Cycle* and a graph that contains such a cycle is called *Hamiltonian.*

The distance $d_G(s, t)$ of two vertices $s, t \in G$ is the length of a shortest s-t path in G. If no such path exists, we set $d_G(s, t) := \infty$. As always, if G is clear from the context we simply write $d(s, t)$.

We often associate a certain cost or weight with each edge, defined by a cost function $c : E \to \mathbb{Q}$. Then the cost of an edge set $E' \subseteq E$ is the sum of the cost of each edge in E', i.e. $c(E') = \sum_{e \in E'} c(e)$.

Connectivity. A non-empty graph G is called *connected* if any two of its vertices are linked by a path, i.e. $d(s, t) \neq \infty$ for each pair of vertices $s, t \in V$. If $U \subseteq V(G)$ and $G[U]$ is connected, we also call U itself connected (in G). If two vertices are not connected, we say that they are *disconnected.* A maximal connected subgraph of G is called a *connected component* of G.

For two vertex sets $A, B \subseteq V$ and $X \subseteq V \cup E$ such that every A-B path in G contains a vertex or an edge from X, we say that X *separates* A and B in G. Note that this implies $A \cap B \subseteq X$. More generally, we say that X *separates* G if $G - X$ is disconnected, i.e. X separates some pair of vertices in G that are not in X. A separating set of vertices is called *separator.* A vertex which separates two other vertices of the same component is called a *cutvertex*, an edge separating its two endpoints is called a *cutedge* or *bridge*. If $\{V_1, V_2\}$ is a partition of V, the set $E(V_1, V_2)$ of all edges *crossing* this partition is called a *cut*. In other words, each edge of a cut has precisely one endpoint in both V_1 and V_2. We also denote the cut by $\delta(V_1) := E(V_1, V_2)$. In a directed graph $D = (V, A)$ a cut is defined by the outgoing edges of V_1, i.e., for a partition of V into V_1 and V_2 we define the cut by $\delta(V_1) := \{(u, v) \in A \mid u \in V_1 \text{ and } v \in V_2\}$.

A graph G is called *k-connected* if $|G| > k$ and $G - X$ is connected for every set of vertices $X \subseteq V$ with $|X| < k$. Similarly, a graph is called *k-edge-connected* if $|G| > 1$ and $G - F$ is connected for every set of edges $F \subseteq E$ satisfying $|F| < k$.

Trees. A graph G that does not contain any cycles is *acyclic* and is called a *forest*. A connected forest is called a *tree*. Thus, a forest is a graph whose connected components are (possibly trivial) trees. The vertices of degree 1 in a tree are its *leaves*.

Bipartite Graphs. A graph $G = (V, E)$ is called *bipartite* if V admits a partition into two disjoint sets U and W such that every edge of E has one endpoint in U and the other in W. For a bipartite graph we often write $G = (U + W, E)$ to indicate that U and W are the partition of V. A cycle of odd length is called an *odd cycle*. Clearly, a bipartite graph can not have odd cycles. In fact, they are characterized by this property.

Proposition 2.1. *A graph is bipartite if and only if it contains no odd cycle.*

A bipartite graph is *balanced* if every induced cycle has length $0 \mod 4$. For a bipartite graph $G = (U + W, E)$ with bipartition (U, W), we denote by $\overline{E}$ the edge-set of its bipartite complement, i.e.

$$\overline{E} := \{uw \mid u \in U, w \in W \text{ and } uw \notin E\}.$$

Contraction and Minors. Let $e = uw$ be an edge of some graph $G = (V, E)$. By G/e we denote the graph obtained from G by *contracting* the edge e into a new vertex v_e, which becomes adjacent to all the former neighbors of u and w. Formally, G/e is a graph $G' = (V', E')$ with vertex set

$$V' := (V \setminus \{u, w\}) \cup \{v_e\}$$

(where v_e is the 'new' vertex, i.e. $v_e \notin V \cup E$) and edge set

$$E' := \big\{xy \in E \mid \{x,y\} \cap \{u,w\} = \emptyset\big\} \cup \big\{v_e y \mid uy \in E \setminus \{e\} \text{ or } wy \in E \setminus \{e\}\big\}.$$

A graph H is called a *minor* of some other graph G if one can obtain H from G by a sequence of edge contractions, edge deletions and vertex deletions. Similarly, we say that H is an *induced minor* of G if one can obtain H from G by a sequence of edge contraction and vertex deletion. We obtain a *subdivision* of some edge if we replace the edge with a path between its endpoints.

Miscellaneous. A *Euler tour* of some graph is a closed walk that contains every edge precisely once. A graph is *Eulerian* if it admits a Euler tour.

Theorem 2.2. *A connected graph is Eulerian if and only if every vertex has even degree.*

A *cactus graph* is a 2-edge-connected spanning graph in which two distinct cycles share at most one vertex. The *complete graph* on n vertices K_n is the graph in which every pair of vertices is adjacent. Similarly, a *clique* in a graph is a set of vertices $V' \subseteq V$ such that every pair of vertices is adjacent and a *stable set* (or *independent set*) is a set of vertices $V' \subseteq V$ such that every pair of vertices is non-adjacent. Note that a clique in G corresponds to a stable set in $\overline{G}$. The *complete bipartite graph* on $2n$ vertices $K_{n,n}$ is a bipartite graph where each partition U and W has precisely n vertices and any two vertices $u \in U$ and $w \in W$ are adjacent.

A directed graph $D = (V, A)$ is an *orientation* of an (undirected) graph G if $V(D) = V(G)$ and $E(D) = E(G)$, and for every undirected edge in G there is a directed edge between its endpoints. For a directed graph D, by $\mathcal{U}(D)$ we refer to the *underlying undirected graph* of D. That is, $\mathcal{U}(D)$ has the same vertex set as D and for each directed edge in D there is an undirected edge in $\mathcal{U}(D)$ with the same endpoints. A directed graph D is *weakly connected* if its underlying undirected graph $\mathcal{U}(D)$ is connected. A directed graph $D = (V, A)$ is *strongly connected* if for each pair of vertices $u, v \in V$ there is a directed path from u to v and a directed path from v to u.

Let $D = (V, A)$ be a directed graph. Two directed edges $(u, v), (w, x) \in A$ are called *anti-parallel* if $u = x$ and $v = w$. That is, they are parallel edges in the underlying graph but have the reversed direction in A.

2.2 Complexity Theory

In this section we give a brief introduction to complexity theory, in particular to polynomial-time solvability, NP-hardness, approximability and fixed parameter tractability. We give a largely informal outline as presented by Schrijver in [Sch03] and also stick to their notation. For a more formal and thorough overview (especially for parameterized complexity and hardness of approximation) we refer the reader to the books [GJ79, CFK$^+$15, WS11].

An *algorithm* can be seen as a finite set of instructions that perform operations on certain data. The *input* of an algorithm will give the initial data. When the algorithm stops, the *output* will be found in prescribed locations of the data set. The instructions need not to be performed in a linear order. For example, an instruction can determine which of the instructions should be performed next or it can prescribe to stop the algorithm.

While the set of instructions constituting the algorithm is finite and fixed, the size of the data set can vary and depends on the input. Usually, the data are stored in finite sequences called arrays. The length of the arrays depends on the input. The number of arrays, however, is fixed and depends only on the algorithm.

The data may consist of numbers, letters and other symbols. In a computer model they are usually stored as finite strings of 0's and 1's (*bits*). The *size* of the data is the total length of these strings. Note that natural numbers are also stored in the binary encoding and thus the size of a natural number n is $\lceil \log n \rceil$.

The Random Access Machine. We use the algorithmic model of the *random access machine*, abbreviated with RAM. This model operates on entries that are $0, 1$ strings that can represent abstract objects like vertices or edges of a graph or rational numbers. An instruction can read several but a fixed number of entries at the same time, perform arithmetic operations on them (like addition or subtraction) and store the answers in array positions prescribed by the instruction. The array positions that should be read and written are given in the location determined by the instruction.

Polynomial-time Solvability. A *polynomial-time algorithm* is an algorithm that terminates after a number of steps bounded by a polynomial in the input size. A *step* of the algorithm consists of performing one instruction. Such algorithms are also called *efficient algorithms*.

In this definition, the *input size* is the size of the input, that is, the number of bits that describe the input. We say that a problem is *polynomial-time solvable*, or *solvable in polynomial-time*, if there exists a polynomial-time algorithm that solves the problem. We also sometimes say that the problem is *tractable*. This definition may depend on the chosen algorithmic model, but it has turned out that for most models the set of problems solvable in polynomial-time is the same. However, we here fix the algorithmic model from above, i.e. the random access model.

The Class P. The classes P and NP are collections of *decision problems*. Decision problems are problems that can be answered by 'yes' or 'no'. A typical example is whether a given graph is connected or not.

A decision problem is completely described by the inputs for which the answer is 'yes'. To formalize this, fix some finite set Σ, called the *alphabet*, of size at least 2. This could be for example $\{0, 1\}$. Let Σ^* denote the set of all finite strings, also called *words*, of letters from Σ. The *size* of a word is the number of letters (counting multiplicities) in the word. We denote the size of a word w by $\text{size}(w)$.

As an example, consider the graph $G = (V, E)$ with four vertices $V = \{u, v, w, x\}$ and five edges $E = \{\{u, v\}, \{v, w\}, \{u, w\}, \{u, x\}, \{w, x\}\}$. It can be represented by the word

$$(\{u, v, w, x\}, \{\{u, v\}, \{v, w\}, \{u, w\}, \{u, x\}, \{w, x\}\})$$

where the alphabet Σ contains all these symbols. Its size is 43.

A *problem* is any subset Π of Σ^*. The corresponding 'informal' problem is:

$$\text{given a word } x \in \Sigma^*, \text{ does } x \text{ belong to } \Pi?$$

As an example, the problem if a given graph is connected is formalized by the collection of all strings representing a connected graph.

The string x is called the *input* of the problem. One speaks of an *instance* of a problem Π if one asks for one concrete input x whether x belongs to Π.

A problem Π is called *polynomial-time solvable* if there exists a polynomial-time algorithm that decides if a given word $x \in \Sigma^*$ belongs to Π or not. The collection of all polynomial-time solvable problems $\Pi \subseteq \Sigma^*$ is denoted by P.

The Class NP. Informally speaking, the class NP is defined as the collection of all decision problems for which each input with positive answer has a polynomial-time checkable 'certificate' of correctness of the answer. As an example, consider the following question:

is a given graph connected?

A positive answer can be 'certified' by giving a spanning tree in the graph and the correctness can be verified in polynomial time. In this case, there is also a certificate for the opposite question:

is a given graph not connected?

Here, a certificate could be a cut $\delta(V')$ of size 0 for some $\emptyset \neq V' \subset V$. However, there is not always a certificate for the opposite question.

As an example, consider the following question:

is a given graph Hamiltonian?

Again, a positive answer can be 'certified' by giving a Hamiltonian cycle in the graph and the correctness can be verified in polynomial-time. However, no such certificate is known for the opposite question:

is a given graph Non-Hamiltonian?

Checking the certificate in polynomial-time means checking it in time bounded by a polynomial in the original input size. In particular, this implies that the certificate itself has size bounded by a polynomial in the input size.

We can formalize this as follows. The class NP is the collection of problems $\Pi \subseteq \Sigma^*$ for which there is a problem $\Pi' \in \mathsf{P}$ and a polynomial p such that for each $w \in \Sigma^*$ one has

$$w \in \Sigma \Leftrightarrow \text{ there exists a word } x \text{ of size at most } p(\text{size}(w)) \text{ with } wx \in \Pi'.$$

The word x is called a *certificate* for w. NP stands for *nondeterministically polynomial-time*, since the string x could be chosen by the algorithm by guessing. Therefore, guessing well would lead to a polynomial-time algorithm for some problem in NP.

Trivially, we have $\mathsf{P} \subseteq \mathsf{NP}$, since if $\Pi \in \mathsf{P}$, we can take $\Pi' = \Pi$ and $p \equiv 0$ in the above equation.

Most of the problems that ask for the existence of some structure belong to NP, since the certificate can be chosen as the structure itself. One of the biggest unsolved mathematical questions is whether $\mathsf{P} = \mathsf{NP}$. However, most mathematicians believe that $\mathsf{P} \neq \mathsf{NP}$.

Optimization Problems. In this thesis we usually do not consider decision problems but *optimization problems.* As an example consider a graph $G = (V, E)$ and some cost function $c : E \to \mathbb{Q}_{\geq 0}$. Then an optimization problem could be to minimize the cost of a Hamiltonian cycle.

Optimization problems can be transformed into decision problems as follows. Consider a *minimization* problem: minimize $f(X)$ over $x \in X$, where X is a collection of elements derived from the input of the problem and where f is a rational-valued function on X (like the example above). This can be transformed to the following decision problem:

$$\text{given a rational numer } r, \text{ is there an } x \in X \text{ with } f(x) \leq r?$$

If we have an upper bound β on the size of the minimum value, then we can find the optimum value by binary search by solving the above decision problem for $O(\beta)$ choices of r. This leads to a polynomial-time algorithm for the minimization problem from a polynomial-time algorithm for the decision problem (analogous for maximization problems). Note that almost all combinatorial optimization problems belong to NP when transformed to a decision problem.

NP-complete Problems. The NP-complete problems are the problems that are the hardest in NP: every problem in NP can be reduced to them. To be more precise, a problem $\Pi \subseteq \Sigma^*$ is said to be *reducible* to problem $\Lambda \subseteq \Sigma^*$ if there exists a polynomial-time algorithm that returns, for any input $w \in \Sigma^*$, an output $x \in \Sigma^*$ with the property:

$$w \in \Pi \Leftrightarrow x \in \Lambda.$$

In particular, this implies that if Π is reducible to Λ and Λ belongs to P, then also Π belong to P. The same is true for NP.

A problem Π is said to be NP-*complete* if it is in NP and each problem in NP is reducible to Π. Hence, if some NP-complete problem belongs to P, then $\mathsf{P} = \mathsf{NP}$. Cook [Coo71] proved that SATISFIABILITY is NP-complete. Afterwards, many problems have shown to be NP-complete by reducing SATISFIABILITY to them. Prominent examples are the problems HAMILTON CYCLE, MAXIMUM CLIQUE or MAXIMUM CUT. By NP-hard we refer to all problems Π such that any other problem in NP can be reduced to Π in polynomial-time, but Π itself does not necessarily have to be in NP.

The concept of polynomial reductions easily extends to optimization problems: a decision problem polynomially reduces to an optimization problem if it has a polynomial-time *oracle algorithm* using the optimization problem as an oracle. In an oracle algorithm we may ask the oracle at any time and we get a correct answer in one step. Then, an optimization problem Π is called NP-hard if all problems in NP polynomially reduce to Π.

Approximation Algorithms. In this thesis we consider various combinatorial optimization problems on graphs. We will show that most of them are NP-hard and therefore it is very unlikely that we find a polynomial-time algorithm solving the problem (finding one would then prove $\mathsf{P} = \mathsf{NP}$).

We now have two options: either we develop an algorithm that computes an optimal solution for the problem (often referred to as *exact algorithm*) which has a running-time that can not be bounded by some polynomial, or we develop an algorithm that computes an *approximate* solution to the problem, but is a polynomial-time algorithm. By approximate we mean that we can not guarantee that the algorithm computes an optimal solution.

In this thesis we often develop the latter kind of algorithms, called *approximation algorithms* and usually wish to give some kind of guarantee on the value of the solution that the algorithm computes. More formally, an approximation algorithm is defined as follows. We stick to the notation from [WS11].

Definition 2.3. An α-*approximation algorithm* for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of α of the value of an optimal solution.

Let us clarify the definition. Let Π be some minimization problem and let $\mathcal{A}$ be a a polynomial-time α-approximation algorithm for Π. Then the value of some solution $\mathcal{A}(\mathrm{I})$ of Algorithm $\mathcal{A}$ satisfies

$$\frac{\text{value}(\mathcal{A}(\mathrm{I}))}{\mathrm{OPT}(\mathrm{I})} \leq \alpha,$$

for *any* instance I of Π, where $\mathrm{OPT}(\mathrm{I})$ denotes the value of an optimal solution for I (throughout the thesis we will denote by $\mathrm{OPT}(\mathrm{I})$ the value of an optimal solution to an instance I).

There are many different kinds of approximation algorithms, depending on the factor α. For example, α could be any *constant* number, resulting in a *constant-factor approximation algorithm.* Even 'better' than a constant-factor approximation algorithm is a *polynomial-time approximation scheme* (PTAS).

Definition 2.4. A *polynomial-time approximation scheme* (PTAS) is a family of algorithms $\{A_\varepsilon\}$, where there is an algorithm for each $\varepsilon > 0$, such that A_ε is a $(1+\varepsilon)$-approximation algorithm (for minimization problems).

The definition for maximization problems is analogue. Note that for fixed ε, the running time of a PTAS is always polynomial but could increase dramatically as ε goes to 0, for example if the running time is $O(n^{\frac{1}{\varepsilon}})$.

However, not all problems allow for a PTAS and there is a class for such problems, called APX. Informally, the class APX contains all problems that allow for a polynomial-time constant-factor approximation algorithm. To this point, we will not formally define APX but note that many interesting problems have shown to be contained in it, such as MAXIMUM SATISFYABILITY or MAXIMUM CUT. Similarly to NP-hardness, one can introduce APX-hardness. Again, we will not formally define this class here but note that the following has been shown.

Theorem 2.5. *For any* APX*-hard problem, there does not exist a polynomial-time approximation scheme unless* $\mathsf{P} = \mathsf{NP}$.

Thus, for any APX-hard problem Π, there is some constant $c(\Pi) > 1$ that depends on Π, such that Π does not admit a $c(\Pi)$-approximation algorithm unless $\mathsf{P} = \mathsf{NP}$. Therefore, the best we can hope for is to find a constant-factor approximation for these problems.

However, for some problems even a constant-factor approximation is impossible and the guarantee of the approximation algorithm has to somehow depend on the size of the input. As we will see in Section 2.3, there are several of them. As an example, for the problem Set Cover there is no $(1 - \varepsilon) \ln n$-factor approximation for any $\varepsilon > 0$ unless $\mathsf{P} = \mathsf{NP}$ [DS14] (for a precise definition of Set Cover we refer to Section 2.3).

Such *inapproximability results* as Theorem 2.5 or the statement above for Set Cover are usually proved by more careful reduction than presented in the previous paragraphs. We do not formalize this, but note that in the reduction, in particular we have to care about the values of feasible solutions and the size of the instance. All reductions we present here in the thesis are somewhat self-explained, that is, if the result implies an inapproximability result, then it is easy to see why.

We sometimes hide some constant factors in the approximation guarantee and write, for example, that an algorithm is an $O(\log n)$-approximation algorithm. We write it like this to encode that the logarithmic factor is the most important one and that the constant factors can be neglected for large n.

For a more detailed introduction to approximation algorithms we refer to the book of Williamson and Shmoys [WS11].

Fixed Parameter Tractability. We start similar to the last paragraph: For an NP-hard optimization problem it is very unlikely that there is a polynomial-time algorithm that computes an optimal solution. A typical question then might be: what exactly does make the problem hard to solve?

This is one of the questions asked in *parameterized complexity.* What kind of parameter (e.g. solution size or maximum degree of the graph) does make the problem hard? Put differently, if we fix some of these parameters, is the resulting problem tractable?

Let us be a bit more formal. Let Π be a an optimization problem and let $\mathcal{I}$ be the set of instances of Π . We call a function $k : \mathcal{I} \to \mathbb{Q}$ a parameter. As mentioned before, such a parameter could be for example the size of an optimal solution.

For a parameter k of some problem Π we call the tuple (Π, k) a parameterized problem. We usually say that Π *is parameterized by k*. We are now ready to define fixed parameter tractable algorithms.

Definition 2.6. A *fixed-parameter tractable* (FPT) algorithm for a parameterized problem (Π, k) is an exact algorithm that has a running time of $f(k)\cdot|\mathrm{I}|^{O(1)}$, where f is a computable function solely depending on k and $|\mathrm{I}|$ denotes the input size of the instance I.

We denote the class of problems that can be solved by an FPT algorithm by FPT. Clearly, all problems in P are also in FPT. However, there are problems for which we do not know if there exists an FPT algorithm or not and it is very unlikely that such an algorithm exists. For example all problems in the class $\mathsf{W}[1]$ do not allow for an FPT algorithm unless $\mathsf{P} = \mathsf{NP}$. We do not want to formally define the classes $\mathsf{W}[i]$, $i \in \mathbb{N}_{\geq 1}$, but only mention here that all problems in these classes do not allow for an FPT algorithm unless $\mathsf{P} = \mathsf{NP}$. Such 'non-FPT' results are usually proved by very careful reduction in which we have to ensure that the respective parameter remains independent of all other inputs (for example, increasing a parameter k to k^2 is fine, but increasing it to $O(|\mathrm{I}|)\cdot k$ is not allowed). Again, all reductions presented in this thesis are somewhat self-explained, so we do not need the precise definitions.

FPT algorithms can be put in contrast with less efficient XP algorithms (XP stands for slice-wise polynomial):

Definition 2.7. An XP algorithm for a parameterized problem (Π, k) is an exact algorithm that has a running time of $f(k)\cdot|\mathrm{I}|^{g(k)}$, where f and g are computable function solely depending on k and $|\mathrm{I}|$ denotes the input size of the instance I.

For a more detailed introduction to parameterized complexity and FPT algorithms we refer to the book of Cygan et al. [CFK+15].

Further Complexity Classes. Most of our hardness results hold unless $\mathsf{P} = \mathsf{NP}$. However, we sometimes provide stronger inapproximability results if we make stronger assumptions than simply $\mathsf{P} \neq \mathsf{NP}$. This motivates to study further complexity classes.

A *randomized algorithm* (sometimes also called *probabilistic algorithm*) is an algorithm that has a certain degree of randomness as part of its instruction. Such an algorithm typically uses uniformly random bits as an auxiliary input to guide its behavior, hoping to achieve good performance in the 'average case' over all possible choices of random bits. A randomized algorithm does not necessarily always compute a feasible solution and sometimes has a certain *error probability*, i.e. the probability that the algorithm computes an infeasible solution. However, some randomized algorithms can be *derandomized* to obtain a deterministic algorithm.

We denote by $\mathsf{DTIME}(t)$ the class of decision problems that have a deterministic algorithm that runs in time t. Furthermore, we denote by $\mathsf{ZTIME}(t)$ the class of decision problems that have a probabilistic algorithm that runs in expected time t (with zero error probability). Here, *expected time* means that the time t is not an upper bound for the running time, but the time the algorithm needs *in expectation*. Therefore, the randomness of the probabilistic algorithms in the class $\mathsf{ZTIME}(t)$ is expressed in the unknown running time of the algorithm and not in the correctness of its solution.

As an example, we will later see the class $\mathsf{ZTIME}(n^{\mathsf{polylog}(n)})$, i.e. the class of decision problems that have a probabilistic algorithm that runs in expected time $n^{\mathsf{polylog}(n)}$ (with zero error probability). For some results, we then do not have the typical condition 'unless $\mathsf{P} = \mathsf{NP}$' but the condition 'unless $\mathsf{NP} \subseteq \mathsf{ZTIME}(n^{\mathsf{polylog}(n)})$'. The function $\mathsf{polylog}(n)$ is a polynomial in $\log n$ (instead n), i.e. it can be written as $\sum_{k=1}^{\ell} a_k(\log n)^k$.

Again, the terminology of $\mathsf{DTIME}(t)$ and $\mathsf{ZTIME}(t)$ can be easily extended to optimization problems.

2.3 Combinatorial Optimization on Graphs

In this section we define basic combinatorial optimization problems on graphs and present results that are used throughout the thesis. Most of the problems we work with are graph problems and can be divided into either connectivity or matching problems. Therefore, in Section 2.3.1, we start by introducing various kinds of connectivity problems on graphs. In Section 2.3.2 we investigate matchings and covering problems. In most of the considered problems we associate certain costs with the resources we are able to buy. When we speak about the *unweighted case* of some problem we refer to the same problem but we want to minimize the cardinality instead of the cost.

2.3.1 Connectivity Problems

The most fundamental problem on graphs is to compute a minimum-cost spanning subgraph that is connected. Then the task is to compute a minimum spanning tree (MST), i.e. a spanning tree in G of minimum cost. This problem is formally defined as follows.

Problem 2.8 (Minimum Spanning Tree)**.** Given an undirected graph $G = (V, E)$ and costs $c \in \mathbb{Q}^E_{\geq 0}$, the task is to find a spanning tree T in G of minimum cost.

It is well-known that Minimum Spanning Tree admits a polynomial-time algorithm. In fact, Minimum Spanning Tree is a special case of the problem of finding a minimum-cost basis of a matroid.

Problem 2.9 (Minimum Matroid Basis)**.** Given a matroid $M = (E, \mathcal{I})$ and a costs $c \in \mathbb{Q}^E_{\geq 0}$, the task is to find a basis $X \subseteq E$ of minimum cost.

The classical greedy-algorithm computes an optimal solution to Minimum Matroid Basis. For example the algorithms developed by Kruskal [Kru56] or Prim [Pri57] compute an optimal solution to the problem Minimum Spanning Tree and also work in a greedy fashion.

Instead of searching for a subgraph that is only 1-edge-connected, one can also search for a graph that is k-edge-connected for some $k = 1, 2, \ldots, |V| - 1$. That is, we would like to compute a minimum cost spanning subgraph that is k-edge-connected. The problem is formally defined as follows.

Problem 2.10 (k-Edge-Connected Spanning Subgraph)**.** Given an undirected graph $G = (V, E)$ and costs $c \in \mathbb{Q}^E_{\geq 0}$, the task is to find a minimum-cost edge-set $X \subseteq E$ such that (V, X) is a k-edge-connected spanning subgraph.

The problem is known to be APX-hard even in the unweighted case and for $k = 2$. However, there is a polynomial-time 2-approximation algorithm for all $k \geq 2$ based on an iterative rounding approach by Jain [Jai01]. In the unweighted case, there are various different best approximation guarantees depending on k. The best known guarantees are $4/3$ for $k = 2$ due to a result of Sebő and Vygen [SV14], $1 + 2/(k + 1)$ for $3 \leq k \leq 6$ due to Cheriyan and Thurimella [CT00], and $1 + 1/2k + O(1/k^2)$ for $k \geq 7$ due to Gabow and Gallagher [GG12].

Similarly, in the problem k-Edge-Connectivity Augmentation we are given a $(k - 1)$-edge-connected spanning subgraph H and would like like to augment it to a k-edge-connected spanning subgraph. More formally, the problem is defined as follows.

Problem 2.11 (k-Edge-Connectivity Augmentation)**.** We are given an undirected graph $G = (V, E)$, a set of edges Y such that (V, Y) is a $(k - 1)$-edge-connected spanning subgraph with $Y \cap E = \emptyset$, and costs $c \in \mathbb{Q}^E_{\geq 0}$. The task is to find a minimum-cost set of edges $X \subseteq E$ such that $(V, Y + X)$ is a k-edge-connected spanning subgraph.

It is shown in [DKL76] that k-Edge-Connectivity Augmentation (k-ECA) can be reduced to Weighted Tree Augmentation (WTAP) for odd k and to Weighted Cactus Augmentation (WCA) for even k. The latter two problems are precisely the special cases of k-ECA for $k = 1$ (WTAP) and $k = 2$ (WCA). Thus, the recent result for Cactus Augmentation [BGA20] implies a 1.91-approximation for Unweighted k-ECA (together with the 1.46-approximation for Unweighted Tree Augmentation [GKZ18]).

In Chapter 5 we will use approximation algorithms for 2-Edge Connected Spanning Subgraph (2-ECSS) and WTAP. Therefore, we also formally define these two problems here and discuss results for these problems in more detail.

Problem 2.12 (2-Edge Connected Spanning Subgraph)**.** Given an undirected graph $G = (V, E)$ and costs $c \in \mathbb{Q}^E_{\geq 0}$, the task is to find a minimum-cost edge-set $X \subseteq E$ such that (V, X) is a 2-edge-connected spanning subgraph.

Problem 2.13 (Weighted Tree Augmentation)**.** We are given an undirected graph $G = (V, E)$, a set of edges Y such that (V, Y) is a tree with $Y \cap E = \emptyset$, and costs $c \in \mathbb{Q}^E_{\geq 0}$. The task is to find a minimum-cost set of edges $X \subseteq E$ such that $(V, Y + X)$ is a 2-edge-connected spanning subgraph.

In some other definitions of WTAP the edges in E are also called links. Note that even though WTAP is a special case of 2-ECSS, the best known approximation guarantee for both problems is 2.

In contrast, for the unweighted versions of both problems (and in the case of WTAP more generally for bounded weights), a long line of results has generated numerous improvements beyond ratio two, leading to the currently best known bounds of 4/3 for unweighted 2-ECSS [SV14], 1.46 for unweighted tree augmenation [GKZ18] and 1.5 for WTAP with bounded weights [FGKS18, GKZ18]. A *bounded weight instance* is an instance in which all weights are in the range of $[1, M]$ for some constant M. Note that achieving the better approximation guarantee for bounded weight instances comes with an increased running time, which is exponential in M.

The case of unit (or bounded) weights is where techniques for approximating 2-ECSS and WTAP start to differ significantly. In both cases, the best known bounds are achieved by combining LP-based techniques with clever combinatorial tools. Nevertheless, there seems to be very little intersection in both the nature of used LPs and the overall approaches, as techniques suitable for one problem do not seem to provide competitive ratios for the other. This is further discussed in Chapter 5.

A similar problem to Weighted Tree Augmentation is the problem Matching Augmentation in which we want to obtain a 2-edge-connected spanning subgraph, but the graph available at zero cost is a matching instead of a tree. The problem is formally defined as follows.

Problem 2.14 (Matching Augmentation)**.** We are given an undirected graph $G = (V, E)$, a set of edges Y such that (V, Y) is a matching with $Y \cap E = \emptyset$ and costs $c \in \mathbb{Q}^E_{\geq 0}$. The task is to find a minimum-cost set of edges $X \subseteq E$ such that $(V, Y + X)$ is a 2-edge-connected spanning subgraph.

Clearly, this problem is also APX-hard. Cheriyan et al. [CDG+19] considered the unweighted version of the problem and gave a $\frac{7}{4}$-approximation algorithm.

Steiner Problems. All the aforementioned problems establish some kind of connectivity between *all pairs* of vertices. A more general approach than connecting all pairs of vertices is simply connecting only a *subset* of the vertices.

This is the concept of *Steiner problems*. The most famous problem is Steiner Tree.

Problem 2.15 (Steiner Tree)**.** We are given an undirected graph $G = (V, E)$, a subset of vertices $T \subseteq V$ called terminals together with a root s, and costs $c \in \mathbb{Q}^E_{\geq 0}$. The task is to find a minimum-cost set of edges $F \subseteq E$ in G such that s is connected to all terminals $t \in T$ in (V, F).

The directed version is called Directed Steiner Tree and there we require directed paths from s to all terminals $t \in T$ in (V, F). Note that in both variants an inclusion-wise minimal solution does not contain cycles.

Both problems are known to be APX-hard, but the directed version is much more difficult when it comes to approximation. The best known approximation algorithm for the undirected version is an $\ln(4)+\varepsilon < 1.39$-approximation algorithm by Byrka, Grandoni, Rothvoß and Sanità [BGRS13]. In comparison, the best known approximation algorithm for the directed version is an $O(k^{\frac{1}{\varepsilon}})$ approximation for any $\varepsilon > 0$, where k is the number of terminal pairs [CCC+99]. Furthermore by a result of Halperin and Krauthgamer [HK03], Directed Steiner Tree does not admit a $(\log^{2-\varepsilon}(n))$-approximation algorithm unless $\mathsf{NP} \subseteq \mathrm{ZTIME}(n^{\mathsf{polylog}(n)})$ for every $\varepsilon > 0$. On the positive side, the problem is FPT in the number of terminal vertices for both the directed and undirected case [DW71].

A generalization of Steiner Tree is the problem Steiner Forest, formally defined as follows.

Problem 2.16 (Steiner Forest)**.** We are given an undirected graph $G = (V, E)$, k terminal pairs $(s_i, t_i)_{1 \leq i \leq k}$, and costs $c \in \mathbb{Q}^E_{\geq 0}$. The task is to find a minimum-cost set of edges $F \subseteq E$ in G such that for each $1 \leq i \leq k$, the vertex s_i is connected to t_i in (V, F).

Again, in the problem Directed Steiner Forest we ask for a forest such that there is a directed path between the terminal pairs $(s_i, t_i)_{1 \leq i \leq k}$. Clearly, Steiner Forest generalizes Steiner Tree since the terminal pairs are not required to be distinct. Therefore, we obtain the same hardness results for Steiner Forest as for Steiner Tree, in both the undirected and directed version. For the undirected case the best approximation algorithm is the 2-approximation algorithm by Jain [Jai01]. The directed case is much more difficult in terms of approximability and there are two incomparable results. There is a $|V|^{2/3+\varepsilon}$-approximation algorithm due to a result of Berman et al. [BBM$^+$13] and a $k^{1/2+\varepsilon}$-approximation algorithm for every $\varepsilon > 0$ due to the results in [CEGS08, FKN09], where k is the number of terminal pairs.

From the FPT algorithm for Steiner Tree it is not hard to derive an FPT algorithm for Steiner Forest. However, it is known that Directed Steiner Forest does not admit an FPT-algorithm when parameterized by the number of terminal pairs [GNS11]. Nevertheless, there is a slicewise polynomial algorithm by Feldman and Ruhl [FR06] that computes an optimal solution in time $O(|E||V|^{4k-2} + |V|^{4k-1} \log |V|)$.

***s*-*t* Connectivity.** On the far opposite site from connecting all vertices is the task of simply connecting two given vertices. This problem is called Shortest Path and is formally defined as follows.

Problem 2.17 (Shortest Path)**.** Given an undirected graph $G = (V, E)$, two vertices $s, t \in V$, and costs $c \in \mathbb{Q}^E_{\geq 0}$ the task is to find a minimum-cost path from s to t.

It is folklore that the problem can be solved in polynomial-time by Dijkstras algorithm [D$^+$59] and can even be solved efficiently if G does not have cycles of negative cost (in case c can also be negative) [FJ56, Bel58]. The problem is NP-hard if there are cycles of negative cost in G.

A generalization of Shortest Path is the task of finding k disjoint s-t-paths in a graph, called k-Disjoint Paths and is defined as follows.

Problem 2.18 (k-Disjoint Paths)**.** Given an undirected graph $G = (V, E)$, an integer k, two vertices $s, t \in V$, and costs $c \in \mathbb{Q}^E_{\geq 0}$, the task is to find a minimum-cost set of edges X such that (V, X) contains k disjoint s-t-paths.

It is well-known that k-Disjoint Paths can be solved by an algorithm for the problem Min Cost s-t-Flow in polynomial-time. In Chapter 4 we will make use of a minimum-cost flow algorithm and some results associated with maximum flows. Therefore, we will introduce this terminology here.

Definition 2.19. We are given a directed graph $D = (V, A)$, two vertices $s, t \in V$, and capacities $b : A \to \mathbb{Q}_{\geq 0}$, denoted by $b(u, v)$ for $uv \in A$. A *flow* is a function $f : A \to \mathbb{Q}_{\geq 0}$, denoted by $f(u, v)$ for each $uv \in A$, such that

1. $f(u,v) \leq b(u,v)$ for each $uv \in A$

2. $\sum_{u:uv \in A} f(u,v) - \sum_{u:vu \in A} f(v,u) = 0$ for all $v \in V \setminus \{s,t\}$.

The *value* of a flow is $|f| = \sum_{su \in A} f(s,u)$.

Such a flow is also called an *s-t-flow*. In the problem MAX s-t-FLOW we wish to find a maximum s-t flow in D, i.e. we want to maximize $|f|$. There are numerous polynomial-time algorithms for finding maximum flows in a graph, for example in [FF09, GT88, Orl13, KRT94]. Note that algorithms for solving MAX s-t-FLOW work on both directed and undirected graphs since for each undirected edge we can add two directed arcs in each direction (observe that an inclusion-wise minimal optimal solution to MAX s-t-FLOW does not contain anti-parallel edges).

The dual problem to MAX s-t-FLOW is the problem MIN s-t-CUT. In the problem MIN s-t-CUT the task is to find a cut between two given vertices of minimum capacity.

Problem 2.20 (MIN s-t-CUT)**.** Given a directed graph $D = (V, A)$, two vertices $s, t \in V$, and capacities $b \in \mathbb{Q}^A_{\geq 0}$ the task is to find a cut $\delta(S)$, $S \subseteq V$, of minimum capacity $b(\delta(S))$ such that $s \in S$ and $t \notin S$.

It is well-known that the optimal value to MAX s-t-FLOW is equal to the optimal value to MIN s-t-CUT.

Theorem 2.21 (Max flow min cut theorem)**.** *We are given a directed graph $D = (V, A)$, two vertices $s, t \in V$, and capacities $b \in \mathbb{Q}^A_{\geq 0}$, for $uv \in A$. Then the optimal value to* MAX s-t-FLOW *is equal to the optimal value to* MIN s-t-CUT.

Note that from a solution to MAX s-t-FLOW we can compute in polynomial time a solution to MIN s-t-CUT of the same value. Hence, MIN s-t-CUT can be solved in polynomial-time.

Finally, we are ready to formalize the problem MIN COST s-t-FLOW.

Problem 2.22 (MIN COST s-t-FLOW)**.** We are given a directed graph $D = (V, A)$, two vertices $s, t \in V$, an integer k, capacities $b \in \mathbb{Q}^A_{\geq 0}$, and costs $c \in \mathbb{Q}^A_{\geq 0}$. The task is to find an s-t flow of value precisely k at minimum cost.

It is well-known that the problem Min Cost s-t-Flow can be solved in polynomial-time [OPT93]. Furthermore, if the capacity function b is integral, then there is also an optimal integral solution and this integral solution can be found in polynomial-time.

2.3.2 Matching and Covering Problems

In this section we define matching and covering problems that play an important role throughout the thesis, especially in Section 3.

Let $G = (V, E)$ be a graph. A subset of the edges $M \subseteq E$ is called a *matching* if the edges in M are pairwise non-adjacent. In other words, any two edges in M do not have a vertex in common. The size of a maximum matching in G is denoted by $\nu(G)$. Note that a natural upper bound on $\nu(G)$ is $\frac{1}{2}|V|$. Therefore, a matching M is called *perfect* if $|M| = \frac{1}{2}|V|$. Computing a maximum size matching is a fundamental problem in combinatorial optimization and there are numerous algorithms that solve this problem [Edm65, MV80]. However, in this thesis we are more interested in computing a *minimum-cost perfect matching.*

Problem 2.23 (Min Cost Perfect Matching)**.** Given an undirected graph $G = (V, E)$ and costs $c \in \mathbb{Q}^E_{\geq 0}$, the task is to find a perfect matching M minimizing $c(M)$ or to decide that G does not admit a perfect matching.

Using an algorithm for finding a maximum matching as a subroutine, one can show that Min Cost Perfect Matching can be solved in polynomial-time [Kol09]. The dual problem of finding a maximum matching is the task of finding a minimum vertex cover. A subset of vertices $V' \subseteq V$ is called a *vertex cover* if $G - V'$ has no edges. In other words, for every edge $e \in E$ at least one of its endpoint is in V'.

Problem 2.24 (Vertex Cover)**.** Given an undirected graph $G = (V, E)$ and costs $c \in \mathbb{Q}^E_{\geq 0}$, the task is to find a vertex cover $V' \subseteq V$ minimizing $c(V')$.

Let us consider the unweighted case of finding a maximum matching and a minimum vertex cover in a graph. The minimum size of a vertex cover is denoted by $\tau(G)$. It is not hard to see that $\tau(G) \geq \nu(G)$ for every graph G since every edge of some matching in G needs to be covered. However, in bipartite graphs we have $\tau(G) = \nu(G)$. This result is called Kőnig's Theorem, named after the Hungarian mathematician Dénes Kőnig. Using an algorithm for maximum matchings, one can compute a minimum vertex cover in bipartite graphs in polynomial time. We also mention here that one can use an algorithm for MAX s-t-FLOW to compute a maximum matching in bipartite graphs (and therefore also a minimum vertex cover).

In general graphs, VERTEX COVER is APX-hard even in the unweighted case. Moreover, if the unique games conjecture is true then VERTEX COVER cannot be approximated within any factor below 2 [KR08]. However, there is a very simple algorithm that achieves a 2-approximation that was, according to [GJ79], discovered independently by Gavril and Yannakakis.

A generalization of VERTEX COVER is a problem called SET COVER, formally defined as follows.

Problem 2.25 (SET COVER)**.** Given a universe $U = \{1, 2, \ldots, n\}$, a family of subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ whose union is U, and costs $c \in \mathbb{Q}^{\mathcal{S}}_{\geq 0}$, the task is to find a minimum-cost subset $\mathcal{S}'$ of $\mathcal{S}$ whose union equals U.

On the positive side, an approximation guarantee of $(\ln(n) + 1)$ can be achieved by a simple greedy algorithm. On the negative side, Dinur and Steurer proved that there is no $(1 - \varepsilon)\ln(n)$-factor approximation for SET COVER for any $\varepsilon > 0$ unless $\mathsf{P} = \mathsf{NP}$ [DS14]. Both results hold for the weighted and unweighted case. Thus, there is no difference in terms of approximability for the unweighted and weighted case of SET COVER.

2.4 Robust Combinatorial Optimization

In this section we briefly discuss three different concepts of robust combinatorial optimization on graphs. As mentioned in the introduction, we focus on *discrete uncertainty*, that is, we are given a certain finite set of realizations of the input data. We first consider the bulk-robust model in which the set of feasible solutions in uncertain, followed by the cost-robust model in which the cost of the resources is subject to uncertainty. Finally, we consider the so called *demand-robustness*, a model in which the demand is uncertain, that is, it is not clear which of the constraints we have to satisfy.

For a more detailed overview of robust combinatorial optimization we refer to the PhD thesis of David Adjiashvili [Adj12], which also deals with other robust concepts such as *recoverable robustness.*

2.4.1 Structural Robustness

The main part of this thesis deals with structural robustness. Structural robustness deals with uncertainty in the set of feasible solutions. We begin with an example.

When building up infrastructure from scratch we would like to ensure that the infrastructure can withstand some kind of failure and still serves its purpose. Consider for example the task of building a connected spanning network, which is exactly the problem Minimum Spanning Tree. If an adversary is able to remove one edge of the computed solution to Minimum Spanning Tree, this solution is not feasible anymore. Such a problem can be addressed if we compute a k-edge-connected spanning subgraph. Then, the solution is *robust* against the failure of up to $k-1$ edges and still satisfies its purpose.

However, in this case we assume that *any* subset of $k-1$ edges is subject to failure. This homogeneity of the resources usually does not reflect the situation in real-life problems. In the bulk-robust model we allow to precisely specify which subsets of the resources can fail simultaneously.

The concept of bulk-robustness was introduced by Adjiashvili, Stiller and Zenklusen [ASZ15]. In the nominal combinatorial optimization problem P we are given a graph $G = (V, E)$ and some kind of property $\mathcal{X}$ that we would like to obtain, such as 'containing a perfect matching', 's-t-connectivity'. Furthermore, we are given a precise list $\Omega \subseteq 2^E$ of scenarios comprising a set of edges each. The goal of the bulk-robust counterpart of P is to find a minimum cost set of edges X such that the desired property is satisfied no matter which scenario materializes, i.e. no matter which set of edges from the list is deleted. Put differently, for a given cost function $c \in \mathbb{Q}^E_{\geq 0}$ we are given the following problem:

$$\min_{X \subseteq E} \{c(X) \mid X \setminus F \text{ has property } \mathcal{X} \text{ for every } F \in \Omega\}$$

The width of an instance, denoted by ℓ, is the largest size of some set $F \in \Omega$, i.e. $\ell = \max_{F \in \Omega} |F|$.

Adjiashvili, Stiller and Zenklusen considered the properties 's-t-connectivity in a graph' and 'containing a spanning tree' (in fact, they considered the property 'basis of a matroid', but for simplicity we stick to the special case of a graphic matroid, i.e. a spanning tree).

First, they showed that the problem is as hard to approximate as Set Cover even for uniform matroids of rank 1. Therefore, almost any bulk-robust counterpart of some combinatorial optimization problem is as hard to approximate as Set Cover and thus does not allow for a sublogarithmic-factor approximation unless $\mathsf{P} = \mathsf{NP}$. Furthermore, they showed even stronger inapproximability results for Bulk-robust Shortest Path, the bulk-robust counterpart of the problem Shortest Path.

They showed that there is no $2^{\frac{1}{4}\ell^{1-\varepsilon}}$-approximation algorithm for Directed Bulk-robust Shortest Path for any $\varepsilon > 0$, where ℓ is the width of the instance, unless $\mathsf{P} = \mathsf{DTIME}(n^{\log\log n})$. Here, directed means that the underlying graph is directed. Additionally, they showed that the problem is APX-hard even for $\ell = 2$.

They complemented these inapproximability results by an efficient $O(\log n)$-factor approximation algorithm for Bulk-robust Spanning Tree and an efficient $O(\log n)$-factor approximation algorithm for Bulk-robust Shortest Path with constant width ℓ. Furthermore, they gave a polynomial-time 13-approximation for Bulk-robust Shortest Path for $\ell = 2$.

Not many other bulk-robust counterparts of combinatorial optimization problems have been considered so far. To the best of our knowledge, only two variants of Bulk-robust Perfect Matching have been considered by Adjiashvili, Bindewald and Michaels [ABM16, ABM17] (and by Bindewald in his PhD thesis [Bin18]).

In [ABM16] they considered the problem Bulk-robust Perfect Matching in bipartite graphs in which the edges are subject to failure. Among other things they showed that the problem is as hard to approximate as Set Cover even if in each scenario at most one edge can be removed from the solution. Additionally, they proposed a polynomial-time $O(\log n)$-factor approximation algorithm based on a randomized rounding algorithm. However, in their work there is a subtle but crucial error, such that the bound on the guarantee is not correct. Furthermore, as we will see in Chapter 3, an $O(\log n)$-factor approximation algorithm for this problem is very unlikely as it would imply $\mathsf{NP} \subseteq \mathsf{ZTIME}(n^{\mathsf{polylog}(n)})$ (Corollary 3.25).

In [ABM17] Adjiashvili, Bindewald and Michaels considered the problem Bulk-robust Perfect Matching in which the vertices are the resources and subject to failure. That is, for a given bipartite graph $G = (U + W, E)$ and a cost function $c : W \to \mathbb{Q}$, the task is to find a minimum weight set of vertices $W' \subseteq W$ such that $G[U + W' - w]$ contains a U-perfect matching for each $w \in W$. Again, they showed that the problem is as hard to approximate as Set Cover and complement this inapproximability result by a polynomial-time $(\log(n) + 2)$-approximation algorithm.

2.4.2 Cost Robustness

In this section we consider a general combinatorial optimization problems of the form

$$\min_{x \in X} c(x),$$

where $X \subseteq \{0,1\}^n$ describes the certain set of feasible solutions and where only the cost vector $c \in \mathbb{Q}^n$ is subject to uncertainty. Furthermore, we are given an uncertainty set $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$ of possible realizations of c. The task then is to solve the following problem:

$$\min_{x \in X} \max_{c \in \mathcal{C}} c(x)$$

We give an overview of results for discrete uncertainty sets. For a more thorough overview of cost-robust combinatorial optimization and other uncertainty sets (like polyhedral or ellipsoidal uncertainty sets) we refer to the survey of Buchheim and Kurtz [BK18].

Unfortunately, even for $X = \{0,1\}^n$ and two scenarios the problem is already (weakly) NP-hard by a reduction to the problem Subset Sum. However, the cost-robust counterparts of several problems, e.g., Shortest Path, Minimum Spanning Tree, the Knapsack Problem and the Unconstrained Binary Problem admit pseudo polynomial algorithms if the number m of scenarios is fixed [KY13, ABV05c, BBI14]. A pseudo polynomial algorithm is an algorithm that is polynomial in the numeric value of the input (the largest integer present in the input) but not necessarily in the size of the input. Furthermore, the cost-robust counterparts of Shortest Path, Minimum Spanning Tree and the Knapsack Problem all admit an FPTAS [ABV05a] for a fixed number of scenarios. Aissi, Bazgan and Vanderpooten [ABV05c] prove that there is always a pseudo polynomial algorithm for the cost-robust counterpart with a fixed number of scenarios if the underlying search problem can be solved in polynomial time. However, Buchheim [Buc20] showed that an algorithm which can access the underlying certain problem only by an optimization oracle needs exponentially many oracle calls in the worst case even without assuming $\mathsf{P} \neq \mathsf{NP}$. Moreover, he showed that oracle-based pseudopolynomial or approximation algorithms cannot exist either.

Interesting to note is that the cost-robust counterpart of Minimum Cut is polynomial-time solvable, while the cost-robust counterpart of Min s-t-Cut is strongly NP-hard even if the number of scenarios is fixed [AZ06, ABV08]. According to [BK18], it is still an open question whether the cost-robust counterpart of the assignment problem is weakly or strongly NP-hard for a fixed number of scenarios.

Finally, if the number of scenarios is unbounded, all of the above problems are strongly NP-hard [KY13, ABV05b, ABV08].

2.4.3 Demand Robustness

Demand robustness was introduced by Dhamdere, Goyal, Ravi and Singh in [DGRS05] and falls into the category of *two-stage optimization problems*. In the first stage, the decision maker has to construct a solution that is only based on the possible realizations of the constraints. In the second stage the constraints materialize and the decision maker can augment his first stage solution to satisfy the final constraints. It is called demand robustness since the set of constraints, or the demand, is unknown in the first stage.

As an example, we consider the demand-robust version of Shortest Path. We are given a graph $G = (V, E)$, a root r, possible target vertices $t_1, t_2, \dots, t_k$ and two cost functions $c, d \in \mathbb{Q}^E_{\geq 0}$. The task is to connect the root r to one of the target vertices $t_1, t_2, \dots, t_k$. In the first stage, the decision maker is able to construct a solution X_1 with cost function c without knowing which target vertex he has to connect to the root. In the second stage the final target t_i is revealed and the decision maker can augment X_1 with a second stage solution X_2 with usually higher costs according to d such that $(V, X_1 \cup X_2)$ contains an r-t_i path. In the demand robust model, the decision maker has to minimize the worst-case cost over all possible scenarios, i.e. he has to compute a robust solution. There are also stochastic versions of the problem in which the decision maker has to minimize the expected value of a solution. However, in this thesis we stick to the robust model.

In [DGRS05] the authors considered demand robust versions of the problems Shortest Path, Vertex Cover, Facility Location, Min s-t-Cut and Min Multi-Cut and provided approximation algorithms with guarantees 30, 4, 5, $O(\log m)$ and $O(\log rm \log\log rm)$, respectively, where m denotes the number of scenarios and r denotes the number of pairs per scenario. In [GGP+15] the approximation guarantees were improved significantly for the demand robust versions of Shortest Path and Min s-t-Cut to 3.39 and 2, respectively.

Feige, Jain, Mahdian and Mirrokni [FJMM07] later extended the model to allow exponential sets of scenarios which were defined implicitly. They considered demand robust versions of Set Cover and Vertex Cover and provided polynomial-time approximation algorithms with guarantees $O(\log m \log n)$ and $2(\frac{2e}{e-1} + 1)$, respectively. For example, the implicit list of demands for Set Cover is the list of all subsets of size k of the elements.

Chapter 3

Robust Matchings

3.1 Introduction

In this chapter we present our results for robust matchings. We say that a bipartite graph is *robust* if it admits a perfect matching after the removal of any single edge. Our goal is to make a bipartite graph robust at minimal cost by adding edges from its bipartite complement and we study the complexity of the corresponding optimization problem.

More formally, in this chapter we study the following problem.

Problem 3.1 (Robust Matching Augmentation)**.** Given an undirected bipartite graph $G = (U + W, E)$ that admits a perfect matching, the task is to find a set $L \subseteq \overline{E}$ of minimum cardinality, such that the graph $G + L$ is robust.

Recap that for a bipartite graph (V, E), we denote by $\overline{E}$ the edge-set of its bipartite complement. Based on a characterization of robustness in terms of strong connectivity, we provide a deterministic $\log_2 n$-factor approximation for Robust Matching Augmentation, as well as a fixed parameter tractable (FPT) algorithm for the same problem parameterized by the treewidth of the input graph. We also give a polynomial-time algorithm for instances on chordal-bipartite graphs, which are bipartite graphs without induced cycles of length at least six. Furthermore, we show that Robust Matching Augmentation admits no polynomial-time sublogarithmic-factor approximation algorithm unless $\mathsf{P} = \mathsf{NP}$, so our approximation guarantee is essentially tight. We also consider the following more general setting. Let us call a bipartite graph k-robust, if it admits a matching of cardinality k after the removal of any single edge. By a simple reduction we show that our algorithmic results carry over to the task of making a bipartite graph k-robust.

We refer by Weighted Robust Matching Augmentation to the generalization of Robust Matching Augmentation, where each edge $e \in \overline{E}$ has a non-negative cost $c(e)$. The task is to find a minimum-cost edge set $L \subseteq \overline{E}$ such that the graph $G + L$ is robust. First, we show that the approximability of Weighted Robust Matching Augmentation is closely linked to that of Directed Steiner Forest. In particular, we show that an $f(n)$-factor approximation algorithm for Weighted Robust Matching Augmentation implies an $f(n + k)$-factor approximation algorithm for Directed Steiner Forest, where k is the number of terminal pairs. By a result of Halperin and Krauthgamer [HK03] it follows that there is no $\log^{2-\varepsilon}(n)$-factor approximation for Weighted Robust Matching Augmentation unless $\mathsf{NP} \subseteq \mathsf{ZTIME}(n^{\mathsf{polylog}(n)})$. On the positive side, we show that an $f(k)$-factor approximation for the Directed Steiner Forest problem yields an $(f(k) + 1)$-factor approximation for Weighted Robust Matching Augmentation. Hence, the algorithms from [CEGS08, FKN09] give an approximation guarantee of $1 + n^{\frac{1}{2}+\varepsilon}$ for Weighted Robust Matching Augmentation, for every $\varepsilon > 0$.

Second, we prove a complexity dichotomy based on graph minors. Let $\mathcal{T}$ be a class of connected graphs closed under connected minors. We show that Weighted Robust Matching Augmentation restricted to input graphs from $\mathcal{T}$ is NP-complete if and only if $\mathcal{T}$ contains at least one of two simple graph classes, which will be defined in Section 3.4, and admits a polynomial-time algorithm otherwise. The polynomial-time algorithm for the remaining instance classes uses a reduction to the Directed Steiner Forest problem with a constant number of terminal pairs, which in turn admits a (slice-wise) polynomial-time algorithm due to a result by Feldman and Ruhl [FR06]. The terminal pairs of the instance are computed by the Eswaran-Tarjan algorithm.

An overview of inapproximability results and approximation factors of our approximation algorithms can be found in Table 3.1.

	unweighted	weighted
Hardness of approximation	$\log n$	$\log^{2-\varepsilon}(n)$
Approximation factor	$\log n$	$1 + n^{\frac{1}{2}+\varepsilon}$

Table 3.1: Hardness of approximation results and approximation factors of our algorithms for unweighted and weighted Robust Matching Augmentation.

Summary of Algorithmic Techniques. By close inspection, it turns out that in order to make some bipartite graph G robust at minimum cost, we may restrict our attention to failures of single edges from any fixed perfect matching M of G. We then show that the resulting problem is equivalent to augmenting a minimum-cost set A of arcs to a given digraph D, such that in the graph $D + A$, each vertex is contained in a strongly connected component and each strongly connected component contains at least two vertices. In order to satisfy these connectivity requirements, we select certain sources and sinks of the condensation of the digraph and add a minimum-cardinality set of arcs, such that the selected sources and sinks are contained in a single strongly connected component. For this purpose, we use the classical Eswaran-Tarjan algorithm. From the arcs we added we obtain an optimal set L of edges such that $G + L$ is robust, provided that the selection of sources and sinks was optimal.

We model the task of selecting sources and sinks as a variant of the Set Cover problem with some additional structure. Given an acyclic digraph, the task is to select a minimum-cardinality subset of the sources, such that each sink is reachable from at least one of the selected sources. We refer to this problem as Source Cover and remark that its complexity may be of independent interest, since it generalizes Set Cover but is a special case of Directed Steiner Tree. We give an FPT algorithm for the Source Cover problem parameterized by the treewidth of the input graph and a polynomial-time algorithm for chordal-bipartite graphs (ignoring orientations). The FPT algorithm is single exponential in the treewidth. Our reductions from Robust Matching Augmentation to Source Cover preserve chordal-bipartiteness and bounded treewidth, so efficient algorithms for Source Cover on these graph classes imply efficient algorithms for Robust Matching Augmentation on the same classes.

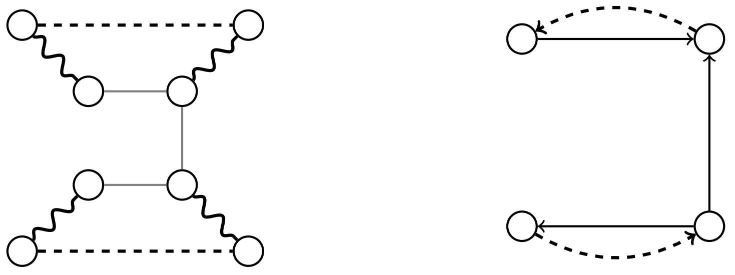
As a by-product of our analysis of the Source Cover problem, we obtain FPT algorithms for the node-weighted and arc-weighted versions of the Directed Steiner Tree problem on acyclic digraphs, which are exponential in the treewidth and linear in the number of nodes of the input graph.

Organization of the Chapter. The remainder of this chapter is organized as follows. We illustrate the relation between robust matching augmentation and strong connectivity augmentation in Section 3.2. In Section 3.3 we show an even closer relation of Robust Matching Augmentation to the Source Cover problem. Algorithms for the Source Cover problem are given in Section 3.3.3 as well as our results on Robust Matching Augmentation with unit costs. In Section 3.4 we give the complexity classification for the weighted version of the problem.

3.2 Robust Matchings and Strong Connectivity Augmentation

In this section we give some preliminary observations on the close relationship between Robust Matching Augmentation with unit costs and strong connectivity augmentation. For this purpose, we fix an arbitrary perfect matching and construct an auxiliary digraph that is somewhat similar to the *alternating tree* used in Edmond's blossom algorithm. We show that the original graph is robust if the auxiliary graph is strongly connected (but not vice versa). Furthermore, we show that there is an optimal edge-set making the given graph robust, that corresponds to a set of arcs connecting sources and sinks in the auxiliary digraph. Finally, if no source or sink of the auxiliary digraph corresponds to a non-trivial robust part of the original graph, then we may use the algorithm for strong connectivity augmentation by Eswaran and Tarjan [ET76] to make the original graph robust. As a consequence, we have that Robust Matching Augmentation on trees can be solved efficiently by using the Eswaran-Tarjan algorithm. In Section 3.3, we will generalize this result.

Let $G = (U + W, E)$ be a bipartite graph that admits a perfect matching and let M be an arbitrary but fixed perfect matching M of G. We call an



(a) Graph G and matching M (wiggly edges).

(b) Digraph $D(G, M)$.

Figure 3.1: Illustration of the correspondence between the dotted edges of G and dotted arcs of $D(G, M)$.

edge $e \in M$ *critical* if $G - e$ admits no perfect matching. Observe that an edge $e \in M$ is critical if and only if it is not contained in an M-alternating cycle. Furthermore, no edge in $E \setminus M$ is critical. Since M is perfect, each edge $e \in M$ is incident to a unique vertex u_e of U. We consider the following auxiliary digraph $D(G, M) = (U, A)$, whose arc-set A is given by

$$A := \{uu' \mid u, u' \in U \mid \exists\, w \in W \text{ such that } uw \in M \text{ and } wu' \in E \setminus M\}.$$

We first note that the choice of the bipartition of G is irrelevant.

Fact 3.2. *Let $G' = (U' + W', E)$, where (U', W') is a bipartition of G. Then $D(G, M)$ is isomorphic to $D(G', M)$.*

Note that we may perform the reverse construction as well. That is, from any digraph D' we may obtain a corresponding undirected graph G and a perfect matching M of G such that $D(G, M) = D'$. In fact, augmenting edges to G is equivalent to augmenting arcs to $D(G, M)$.

Fact 3.3. *Let $\overline{A}$ be the set of arcs that are not present in $D(G, M)$. Then there is a one-to-one correspondence between $\overline{E}$ and $\overline{A}$.*

An example of the correspondence mentioned in Fact 3.3 is shown in Figure 3.1. In order to keep our notation tidy, we will make implicit use of Fact 3.3 and refer to $\overline{A}$ and $\overline{E}$ interchangeably. Observe that for edges $e, f \in M$ there is an M-alternating path containing e and f in G if and only if u_e is connected to u_f in $D(G, M)$. This implies the following characterization of robustness.

Fact 3.4. *G is robust if and only if each strongly connected component of $D(G,M)$ is non-trivial, that is, it contains at least two vertices.*

Let D' be a digraph. A vertex of D' is called a *source* (*sink*) if it has no incoming (outgoing) arc. We refer to the set of sources (sinks) of D' by $V^+(D')$ $(V^-(D'))$. Furthermore, we denote by $C(D')$ the *condensation* of D', that is, the directed acyclic graph of strongly connected components of D'. We call a source or sink of $C(D')$ *strong* if the corresponding strongly connected component of D' is non-trivial. From Fact 3.4 it follows that a subgraph of G that corresponds to a strong source or a strong sink is robust against the failure of a single edge. Furthermore, observe that the choice of the perfect matching M of G is irrelevant in the following sense.

Fact 3.5. *Let M and M' be perfect matchings of G. Then $C(D(G,M))$ is isomorphic to $C(D(G,M'))$.*

Proof. Let M and M' be two distinct perfect matchings of G. Then their symmetric difference $M \,\Delta\, M'$ is a sum of (M,M')-alternating cycles. But each cycle is in some strong component of $D(G,M)$ and $D(G,M')$, so both condensations must be isomorphic. ■

Fact 3.5 is of key importance for our algorithmic results, for which we generally assume that some fixed perfect matching is given. Next, we observe that for unit costs we may restrict our attention to connecting sources and sinks of $C(D)$ in order to make G robust. It is easy to check that this does not hold for general non-negative costs.

Fact 3.6. *Let $L \subseteq \overline{E}$ such that $G+L$ is robust. Then there is some $L' \subseteq \overline{E}$ of cardinality at most $|L|$, such that $G+L'$ is robust and L' connects only sinks to sources of $C(D(G,M))$.*

Proof. Let vw be an arc in L. Let L' be a copy of L, where the arc vw is replaced by an arc $v'w'$ from a sink v' of $C(D(G,M))$ reachable from v to a source w' of $C(D(G,M))$ reachable from w. We show that $G+L'$ is robust. Suppose for a contradiction that this is not the case. Then there is some edge $xy \in M$, such that $x \in U$, $y \in W$, and xy is not on an M-alternating cycle in $G+L'$. Equivalently, x is not contained in a directed cycle of $D+L'$.

However, since $G + L$ is robust, we have that x and the arc vw are contained in some directed cycle $C = \{v_1, v_2, \ldots, v_k = v_1\}$ of $D + L$. That is, there are $1 \leq i, j < k$, such that $x = v_i$, $v = v_j$, and $w = v_{j+1}$. Let P (Q) be a path connecting v and v' (w' and w). Then $C' := v_1, v_2, \ldots, v_{j-1}, P, Q, v_{j+2}, \ldots, v_k$ is a closed walk that contains a simple directed cycle visiting x. This contradicts our assumption that x is not on a directed cycle in $G + L'$. By iterating this argument we obtain an arc-set L' such that $|L'| \leq |L|$ and $G + L'$ is robust. By construction, L' contains only arcs that connect sources and sinks of $C(D(G, M))$. ∎

We remark that the construction of L' given in the proof of Fact 3.6 can be performed in polynomial time.

We denote by $\gamma(D')$ the minimal number of arcs to be added to a digraph D' in order to make it strongly connected. Eswaran an Tarjan have proved the following min-max relation [ET76].

Fact 3.7. *Let D' be a digraph. Then $\gamma(D') = \max\{|V^+(D')|, |V^-(D')|\}$.*

From the proof of Fact 3.7 it is easy to obtain a polynomial-time algorithm that, given a digraph D', computes an arc-set L of cardinality $\gamma(D')$ such that $D' + L$ is strongly connected [FJ15]. We will refer to this algorithm by Eswaran-Tarjan. The following proposition illustrates the usefulness of the algorithm Eswaran-Tarjan for Robust Matching Augmentation, and at the same time its limitations.

Fact 3.8. *Suppose that $C(D(G, M))$ contains no strong sources or sinks. Then Eswaran-Tarjan computes a set $L \subseteq \overline{E}$ of minimum cardinality such that $G + L$ is robust.*

Proof. By assumption, we have that $C(D(G, M))$ contains no strong sources or sinks. Therefore, each source and each sink of $C(D(G, M))$ corresponds to a critical edge of the matching M. Let $L' \subseteq \overline{E}$ of minimum cardinality, such that $G + L'$ is robust. By Fact 3.6, we may assume that L' connects only sinks to sources of $C(D(G, M))$.

If $|L'| < \gamma(D(G, M)) = \max\{|V^+(C(D(G, M)))|, |V^-(C(D(G, M)))|\}$, then at least one sink or at least one source is not incident to an arc of L'. Therefore, the graph $G + L'$ is not robust. ∎

Fact 3.8 implies that Eswaran-Tarjan solves Robust Matching Augmentation on trees. If strong sources or sinks are present in $D(G, M)$, then we may or may not need to consider them in order to make G robust. This is precisely what makes the problem Robust Matching Augmentation hard. This close connection will be presented in Section 3.3. We will formalize the task of selecting strong sources and sinks in terms of the Source Cover problem, which is discussed in Section 3.3.3.

3.3 Unweighted Robust Matching Augmentation

In this section we present our main technical tool for solving the problem Robust Matching Augmentation. By combining it with the results in Section 3.3.3 we obtain our algorithmic results. We say that a perfect matching $M \subseteq E$ of G is robust if $G - e$ contains a perfect matching for each $e \in M$. Now let us restate the problem Robust Matching Augmentation in a slightly different way.

Problem 3.9 (Alternative formulation for Robust Matching Augmentation)**.** Given a bipartite graph $G = (U + W, E)$ and a perfect matching M of G, the task is to find a minimum-cardinality set $L \subseteq \overline{E}$ such that M is robust in $G + L$.

Fixing the perfect matching M in the instance is just for notational convenience, since we can compute a perfect matching in polynomial time and our results do not depend on the exact choice of M, according to the discussion in Section 3.2.

3.3.1 Complexity

We show that the problem Robust Matching Augmentation is NP-hard, even on (bipartite) graphs of maximum degree three. Furthermore, it is NP-hard to find a $o(\log n)$-approximate solution in polynomial time. The result mainly follows from the results of [Bin18] and an additional lemma. Nevertheless, we give the full proof here.

Proposition 3.10. Robust Matching Augmentation *parameterized by the solution size is* W[2]*-hard, even on graphs of maximum degree three.*

Proof. We give a parameterized reduction from Set Cover, which is W[2]-hard. Let $(X, \mathcal{S})$ be an instance of Set Cover. We construct an instance (G, M) of Robust Matching Augmentation as follows. Let d be the maximal cardinality of the sets in $\mathcal{S}$. For each set $S \in \mathcal{S}$, we add a cycle C_S of length $2d$ on the vertices $c_S^1, c_S^2, \ldots, c_S^{2d}$ and for each item $u \in X$, we add an edge u_1u_2 to G. For each $u \in X$ and $S \in \mathcal{S}$, if $u \in S$, we join u_1 to c_S^i by an edge, such that i is odd and the vertex c_S^i has maximum degree three. This is possible since C_S has length $2d$. Finally, we add two vertices t_1 and t_2 to G, join them by an edge, and connect for each $u \in X$, u_2 to t_1. The matching M contains for each $S \in \mathcal{S}$ the edges $c_S^1c_S^2, c_S^3c_S^4, \ldots, c_S^{2d-1}c_S^{2d}$ and for each $u \in X$ the edge u_1u_2, and also t_1t_2. It is readily verified that M is a perfect matching of G. Let us choose the bipartition (U, W) of G such that $u_1 \in U$ for some $u \in X$.

Claim 1. *$C(D(G, M))$ contains a single sink t_1 and for each $S \in \mathcal{S}$ its node-set $V(C_S)$ defines a strong source.*

Proof. Clearly, the vertices of each cycle C_S are in a strong component of $D(G, M)$. Observe that by the construction of G, any maximal M-alternating path that leaves a cycle C_S terminates in t_2. It follows that t_1 is the only sink of $C(D(G, M))$. Moreover, no two distinct cycles C_S and $C_{S'}$ are in the same strong component of $C(D(G, M))$. This completes the proof of Claim 1. □

Let $L \subseteq \overline{E}$ be an optimal solution to (G, M). By Fact 3.6, we can assume that L connects sources to the unique sink of $C(D(G, M))$. We now set

$$C_L := \{S \in \mathcal{S} \mid L \text{ connects } C_S \text{ to } t_1\}.$$

Next, we prove that L is a solution of size ℓ if and only if C_L is a solution of size ℓ. For the only if part, assume this is not true and let $u \in X$ be not covered by C_L. Thus, none of the sets containing u is contained in C_L, meaning that L does not connect t_1 to a strong source that is a predecessor of u_1 in $D(G, M)$ (as L only connects t_1 to strong sources). Hence u_1u_2 is not contained in an alternating cycle, a contradiction. For the if part, let C_L be a cover of size ℓ and let L be the corresponding arcs in D. Assume u_1 is not contained in a strong component in $D(G+L, M)$. As L only connects strong sources to sinks, no predecessor of u_1 has an edge to t_1. This is a contradiction to C_L being a cover.

We now describe how to reduce the degree of the constructed graph. Note that the only vertices with degree possibly greater than 3 are t_1 and u_1, $u \in X$. All of them are in U. Consider a vertex $u \in U$ of degree at least $q > 3$ with its neighbors $w_1, \ldots, w_q$. We do not connect the vertices w_i, $1 \leq i \leq q$ directly to u. Instead we add a path $P = \{u'_1 w'_1 u'_2 w'_2 \ldots u'_q = u\}$, where for $1 \leq i < q$ the edges $u'_i w'_i$ are matching edges. Instead of $w_i u$ we add the edges $w_i u'_i$ for $1 \leq i \leq q$. Observe that we still have the same properties as before but each vertex in G has degree at most 3. This concludes the proof of Proposition 3.10. ■

Proposition 3.11. Robust Matching Augmentation *admits no polynomial time $o(\log n)$-factor approximation algorithm unless* $\mathsf{P} = \mathsf{NP}$*, where n is the number of critical edges of the input graph.*

Proof. Assume for a contradiction that there is a polynomial-time algorithm A that computes an $f(n)$-approximate solution of Robust Matching Augmentation, where $f(n) = o(\log n)$. Let $\mathrm{I}' = (X, \mathcal{S})$ be an instance of Set Cover and construct from I' in polynomial time an instance I of Robust Matching Augmentation as in the proof of Proposition 3.10. We now also have that $\mathrm{OPT}(\mathrm{I}) = \mathrm{OPT}(\mathrm{I}')$ and $n = |X|$. Applying algorithm A on I yields a solution L of cardinality at most $f(n) \cdot \mathrm{OPT}(\mathrm{I})$. Without loss of generality, we may assume that L only connects sources and sinks due to Fact 3.6. We now set

$$C_L \coloneqq \{S \in \mathcal{S} \mid L \text{ connects } C_S \text{ to } t_1\}.$$

By the same arguments as in the proof of Proposition 3.10, we observe that C_L is a feasible solution to I' of cardinality at most $f(n) \cdot \mathrm{OPT}(\mathrm{I}) = f(n') \cdot \mathrm{OPT}(\mathrm{I}')$. This contradicts an inapproximability result of Dinur and Steurer for Set Cover [DS14]. ■

3.3.2 Main Result

For the main theorem of this section we need to introduce the Source Cover problem. Given an acyclic digraph, the Source Cover problem asks for a minimum-cardinality subset of its sources, such that each sink is reachable from at least one selected source. The Source Cover problem is formally defined as follows.

Problem 3.12 (Source Cover)**.** Given an acyclic digraph $D = (V, A)$, the task is to find a minimum-cardinality subset S of the sources $V^+(D)$ of D, such that for each sink $t \in V^-(D)$ there is an S-t-path in D.

We are now ready to state our main theorem of this section.

Theorem 3.13. *There is a polynomial-time algorithm that, given an instance* $\mathrm{I} = (G, M)$ *of* Robust Matching Augmentation*, computes two instances* $\mathcal{A}_1 = (S_1)$ *and* $\mathcal{A}_2 = (S_2)$ *of* Source Cover *such that the following holds.*

1. $\mathcal{U}(S_1)$ *and* $\mathcal{U}(S_2)$ *are induced minors of* $\mathcal{U}(D(G, M))$*.*
2. $\mathrm{OPT}(\mathrm{I}) = \max\{\mathrm{OPT}(\mathcal{A}_1), \mathrm{OPT}(\mathcal{A}_2)\}$*.*
3. *From a solution* C_1 *of* $\mathcal{A}_1$ *and a solution* C_2 *of* $\mathcal{A}_2$ *we can construct in polynomial time a solution* L *of* I *of cardinality* $\max\{|C_1|, |C_2|\}$*.*

Proof. Let G be given by $G = (U + W, E)$. Our goal is to obtain from solutions of the Source Cover instances a suitable selection of sources and sinks of $C(D(G, M))$, such that we can make M robust by connecting the selected sources and sinks, using the algorithm Eswaran-Tarjan. Let us denote by u_e the vertex in U that is incident to an edge $e \in M$. Furthermore, let $D := D(G, M)$. We construct the Source Cover instance $\mathcal{A}_1$ as follows. For each unsafe edge $e \in M$, we remove from D each vertex $v \in U - u_e$, such that v is reachable from u_e in D. Furthermore, we delete all vertices that are in a strong sink of $C(D)$ and iterate this process until the resulting graph, say D', has no strong sink. Then the Source Cover instance $\mathcal{A}_1$ is given by $\mathcal{A}_1 := (C(D'))$. The construction of $\mathcal{A}_2$ is as for $\mathcal{A}_1$, but with the arcs of D reversed. This turns the sources of D into sinks and vice versa. Clearly, the input graphs of $\mathcal{A}_1$ and $\mathcal{A}_2$ are induced minors of $\mathcal{U}(D)$, since they were constructed by deleting vertices of $\mathcal{U}(D)$ and contracting strong components. By Fact 3.4, the set of unsafe edges can be obtained efficiently by Tarjan's classical algorithm for computing strongly connected components. It remains to prove statements 2 and 3.

Let C_1 (C_2) be a solution to $\mathcal{A}_1$ ($\mathcal{A}_2$). We show how to construct in polynomial-time a solution L of I of cardinality $\max\{|C_1|, |C_2|\}$. Let $X \subseteq V(D)$ be the set of vertices incident to unsafe edges. Moreover, let $\widehat{D} \subseteq C(D)$ be the graph induced by the vertices of $C(D)$ that are on C_1X-paths or on XC_2-paths in $C(D)$. Note that $\widehat{D}$ can be computed by a depth-first search applied on each source and sink. By running Eswaran-Tarjan on $\widehat{D}$ we obtain an arc-set L^* such that $\widehat{D} + L^*$ is strongly connected. Hence, each $u \in X$ is on some directed cycle in $\widehat{D} + L^*$. From L^* we can obtain in a straight-forward way an arc-set L of the same cardinality, such that each $u \in X$ is on some directed cycle of $D + L$. For each arc $ss' \in L^*$, we add an arc uu' to L, where u (u') is some vertex in the strong component s (s') of D. By the construction of L, each $u \in X$ is on some directed cycle of D. By Facts 3.3 and 3.7 we have constructed a solution L of I of cardinality $|L| = |L^*| = \max\{|C_1|, |C_2|\}$. This concludes the proof of Statement 3.

It remains to prove that $\text{OPT(I)} \geq \max\{\text{OPT}(\mathcal{A}_1), \text{OPT}(\mathcal{A}_2)\}$. Suppose for a contradiction that $\text{OPT(I)} < \max\{\text{OPT}(\mathcal{A}_1), \text{OPT}(\mathcal{A}_2)\}$. Without loss of generality, let $\text{OPT}(\mathcal{A}_1)$ attain the maximum. Due to Fact 3.6, we may assume that an optimal solution L of I connects sources and sinks of $C(D)$. Let $R \subseteq V(C(D))$ be the corresponding sources of $C(D)$. Then for each unsafe edge $e \in M$, the vertex u_e must be reachable from some source $s \in R$. But then R is a solution of $\mathcal{A}_1$ of cardinality $|R| = \text{OPT(I)} < \text{OPT}(\mathcal{A}_1)$, which is a contradiction. This concludes the proof of Theorem 3.13. ■

By Theorem 3.13, in order to solve Robust Matching Augmentation, is suffices to solve two instances of Source Cover. Due to Statement 1 of the theorem, structural features of the input graph, such as bounded treewidth and chordal-bipartiteness, are passed on to the digraphs of the source cover instances. We now consider the following more general setting, which we call Robust k-Matching Augmentation. We call a bipartite graph *k-robust* if it admits a matching of cardinality k after the removal of any single edge.

Problem 3.14 (Robust k-Matching Augmentation). Given a bipartite graph $G = (U + W, E)$ that admits a matching of size k, the task is to find a minimum-cardinality set $L \subseteq \overline{E}$ such that the graph $G + L$ is k-robust.

Note that if k is less than the size of a maximum matching, then $L = \emptyset$ is optimal due to the existence of a larger matching. We give a polynomial-time reduction from Robust k-Matching Augmentation to Robust Matching Augmentation. Let (G, M) be an instance of Robust k-Matching Augmentation, where the input graph G is given by $G = (V, E)$. Without loss of generality, we assume that M is U-perfect, so $|U| \leq |W|$. Otherwise, adding an edge joining two unmatched vertices solves the problem. We construct an instance (G', M') of Robust Matching Augmentation as follows. Let G' be a copy of G to which we add a leaf to each unmatched vertex of W. We then add a vertex z to U joined to each vertex of the other part of the bipartition. Finally, we add a vertex z' joined to z and each leaf added in the previous step. We extend the matching M of G to a perfect matching M' of G' by adding to M the edge zz' and the edges between the additional leaves and the previously unmatched vertices. Note that by construction, if e is a unsafe edge of G', then $G - e$ does not admit a matching of cardinality $|M|$.

Note that the construction increases the treewidth by at most two, but does not preserve chordal-bipartiteness of the input graph. However, the corresponding digraph contains no induced cycle of length at least six, so all our algorithmic results for Robust Matching Augmentation carry over to Robust k-Matching Augmentation.

Proposition 3.15. *There is a polynomial-time reduction f from* Robust k-Matching Augmentation *to* Robust Matching Augmentation, *such that the following holds. Let $\mathrm{I} := (G)$ be an instance of* Robust k-Matching Augmentation *and let $f(\mathrm{I}) = (G')$. Then the following statements are true.*

1. $\mathrm{OPT}(f(\mathrm{I})) = \mathrm{OPT}(\mathrm{I})$ *and from a solution L' of $f(\mathrm{I})$ we can construct in polynomial-time a solution L of I such that $|L| \leq |L'|$.*
2. $tw(G') \leq tw(G) + 2$.
3. *If G is chordal-bipartite, then $\mathcal{U}(D(G', M'))$ has no induced cycle of length at least six.*

Proof. We prove the statements one by one.

Claim 1. $\mathrm{OPT}(\mathrm{I}') = \mathrm{OPT}(\mathrm{I})$ *and from a solution L' of I' we can construct in polynomial time a solution L of I such that $|L| \leq |L'|$.*

Proof. Let (U, W) be the bipartition of G as chosen in the construction, i.e., such that $z \in U$. Note that since z is joined to each vertex $w \in W$, there is an arc from each vertex of $D(G', M')$ to z. Therefore, $C(D(G', M'))$ has a single strong sink, say $\widehat{S}$, originated from the vertex set $\widehat{Y} \subseteq V(D(G', M'))$. Observe that $z, z', u'_1, u'_2, \ldots, u'_k \in \widehat{Y}$. For a strong component s of $D(G', M')$, let Y_s be the set of vertices of $V(D(G', M'))$ in the component s.

We first show that $\mathrm{OPT}(\mathrm{I}) \leq \mathrm{OPT}(\mathrm{I}')$. Let $\tilde{L}$ be a solution of I'. According to Fact 3.6 and the algorithm contained in its proof we may construct from $\tilde{L}$ a solution L' to I' of cardinality at most $|\tilde{L}|$, such that L' connects only sources and sinks of $C(D(G', M'))$. Since there is only the sink $\widehat{S}$, we may further assume that L' connects $\widehat{S}$ to a selection $S \subseteq V^+(D(G', M'))$. Let $x \in W$ be M-exposed. We construct a solution L of I as follows. For each source $s \in S$, we pick a vertex $u \in U$ in the corresponding component in $D(G', M')$ and add the edge ux to L. We now show that $G + L$ is robust. Recap that by construction, the critical edges of (G', M') are precisely the critical edges of (G, M). Let $e \in M$ be a critical edge of (G, M). Since L' is feasible for I', any vertex $u \in U$ that is incident to a critical edge of (G', M') is reachable from some $s \in S$ by a directed path in $C(D(G', M'))$. This directed path corresponds to an M-alternating path in G starting from any vertex $u \in Y_s$ with an M-edge. Therefore, the edge e is not critical in $(G + ux, M)$ for any $u \in Y_s$. Hence, $(G + L, M)$ has no critical edges and from $|L| = |L'| \leq |\tilde{L}|$ we conclude that $\mathrm{OPT}(\mathrm{I}) \leq \mathrm{OPT}(\mathrm{I}')$. Moreover, we can construct L from L' in polynomial time.

It remains to show that $\mathrm{OPT}(\mathrm{I}') \leq \mathrm{OPT}(\mathrm{I})$. Let L be an optimal solution of I. Note that each critical edge of (G, M) is on an M-alternating cycle or a maximal even-length M-alternating path in $G + L$. We construct from L a solution L' to I'. Let $x \in W$ be M-exposed. For each $u \in U$ and $w \in W$ such that $uw \in L$, we add the edge ux to L'. We show that L' is feasible for I'. Let $uw \in L$ and let $e \in M$ be a critical edge of (G, M) on a maximal M-alternating path P of even length. By replacing uw by ux, we split P into at most two maximal M-alternating paths of even length. On the other hand, suppose that e be on some M-alternating cycle involving uw. Replacing uw by ux yields a maximal M-alternating path containing e. Therefore, each critical edge of (G, M) is on some maximal M-alternating path of even length in $G+L'$. By the construction above, each critical edge of (G', M') is hence on some maximal M'-alternating cycle of $(G'+L', M')$, so M' is robust in $G'+L'$. Since $|L| = |L'|$, we have that $\text{OPT(I)} \leq \text{OPT(I}')$. This proves Claim 1. $\square$

Claim 2. $tw(G') \leq tw(G) + 2$.

Proof. To prove Claim 2, observe that adding a single vertex to a graph increases its treewidth by at most one. Furthermore, adding a leaf vertex to a graph does not increase its treewidth. We obtain G' from G by adding leaf vertices to each exposed vertex and finally add two more vertices. Therefore, $tw(G') \leq tw(G) + 2$. This proves Claim 2. $\square$

Claim 3. *If G is chordal-bipartite, then $\mathcal{U}(D(G', M'))$ has no induced cycle of length at least six.*

Proof. Now suppose that G is chordal-bipartite. Assume for a contradiction that $H = \mathcal{U}(D(G', M'))$ has an induced cycle C' of length at least six. It is easy to see that z is not contained in C' since z is adjacent to all $v \in H$. In order to obtain a cycle C in G, for every edge e in $H[C']$, replace e by the unique corresponding path P_e in G' consisting of a matching edge and a non-matching edge. If two consecutive paths P_e and $P_{e'}$ use the same matching edge, simply delete those matching edges in C such that C is a cycle. Note that all edges in $H[C']$ incident to U' are directed from U' to U in $D(G', M')$. Hence, consecutive edges to vertices in U' use the same matching edges, which are then deleted. Therefore $V(C) \subseteq V(G)$. Now if $G[C]$ contains a chord then $H[C']$ also contains a chord due to fact 3.3. Therefore C is an induced cycle in G (since $z \notin C'$) and $|C| \geq |C'| \geq 6$, a contradiction. This concludes the proof of Claim 3. $\square$

This proves Proposition 3.15. ■

3.3.3 The Source Cover Problem

In Section 3.3.2 we made precise the close relation between Robust Matching Augmentation and the Source Cover problem. In this section we present our algorithmic results for the Source Cover problem as well as their consequences for Robust k-Matching Augmentation. Recall that the Source Cover problem asks for a minimum-cardinality subset of the sources of a given digraph, such that each sink is reachable from at least one selected source. It is easy to see that Source Cover is a special case of the Directed Steiner Tree problem and that it generalizes Set Cover. We give a simple polynomial-time algorithm for Source Cover if the input graph is chordal-bipartite (ignoring orientations). Furthermore, we show that Source Cover parameterized by treewidth (again ignoring orientations) is FPT. As a by-product, we obtain a simple FPT algorithm for the arc-weighted and node-weighted versions of the Directed Steiner Tree problem on acyclic digraphs, whose running time is linear in the size of the input graph and exponential in the treewidth of the underlying undirected graph. To the best of our knowledge, the parameterized complexity of the general Directed Steiner Tree problem with respect to treewidth is open. For the corresponding *undirected* Steiner Tree problem, an FPT algorithm was given by Bodlaender et al. in [BCKN15].

By a “flattening” operation on the input digraph, we can transform an instance $\mathrm{I} = (D)$ of Source Cover into a Set Cover instance as follows. Let $F(D) = (V^+(D) \cup V^-(D), A')$ be an acyclic digraph, where A' is given by

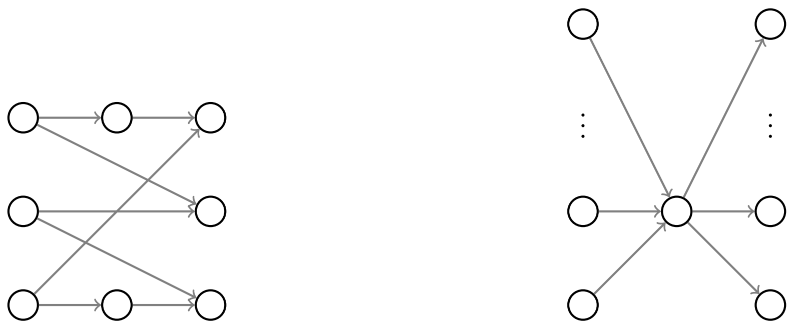
$$A' := \{st \mid s \in V^+(D),\, t \in V^-(D),\, t \text{ is reachable from } s \text{ in } D\}.$$

Then $\mathcal{U}(F(D))$ is the incidence graph of a Set Cover instance $\mathcal{A}$ on $V^-(F(D))$, such that the feasible solutions of I and $\mathcal{A}$ are in one-to-one correspondence.

As a first consequence of Theorem 3.13, Proposition 3.15, and this “flattening” we may use the classic Greedy-Algorithm for Set Cover to obtain a simple $\log_2 n$-factor approximation algorithm for Robust k-Matching Augmentation.

Corollary 3.16. Robust k-Matching Augmentation *admits a polynomial-time* $\log_2 n$*-factor approximation algorithm, where* n *is the number of vertices of the input graph.*

Proof. Let $\mathrm{I} = (G, M)$ be an instance of Robust Matching Augmentation. We use Theorem 3.13 to obtain from I in polynomial time the Source Cover instances $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $\mathrm{OPT}(\mathrm{I}) = \mathrm{OPT}(\mathrm{I}') = \max\{\mathcal{A}_1, \mathcal{A}_2\}$. For $i \in \{1, 2\}$ let S_i be the acyclic input graph of $\mathcal{A}_i$. We “flatten” the graph



(a) A digraph D such that $\mathcal{U}(D)$ is balanced, but $\mathcal{U}(F(D))$ is not.

(b) A digraph D such that $\mathcal{U}(D)$ has treewidth one, but the treewidth of $\mathcal{U}(F(D))$ is unbounded.

Figure 3.2: Examples showing that flattening does not preserve balancedness or bounded treewidth.

S_i as described before to obtain a Set Cover instance $\mathcal{B}_i$ on the incidence graph $\mathcal{U}(F(S_i))$. The classical greedy algorithm for Set Cover yields an $((\ln |M|) + 1)$-approximate cover C_i for $\mathcal{B}_i$. By Theorem 3.13, we can construct from C_1 and C_2 in polynomial time a solution L of I. By recalling that $n = |V(G)| \geq |M|/2$ and some simple calculations, we conclude that L is an $\log_2 n$-approximate solution. ■

However, as illustrated in Figure 3.2, some useful properties of the input digraph may not be preserved by the "flattening" operation. In particular, if $\mathcal{U}(D)$ has treewidth at most r, then the treewidth of $\mathcal{U}(F(D))$ cannot be bounded by a constant in general. Furthermore, the graph $\mathcal{U}(F(D))$ is not necessarily balanced (or planar) if $\mathcal{U}(D)$ is. Therefore, we cannot take advantage of polynomial-time algorithms for Set Cover on balanced incidence graphs or incidence graphs of bounded treewidth. Motivated by the example in Figure 3.2b we leave as an open question, whether Source Cover on balanced graphs admits a polynomial-time algorithm. By Theorem 3.13, the existence of such an algorithm implies a polynomial-time algorithm for Robust Matching Augmentation on balanced graphs.

Source Cover on Chordal Bipartite Graphs

We show that in contrast to the treewidth and balancedness, the property of a graph being chordal bipartite is indeed preserved by the flattening operation introduced above. From this we obtain the following result.

Theorem 3.17. Source Cover *on chordal-bipartite graphs admits a polynomial-time algorithm.*

Proof. Let (D) be a Source Cover instance such that $\mathcal{U}(D)$ is connected, has at least one arc, and $\mathcal{U}(D)$ contains no induced cycle of length at least six. If $\mathcal{U}(F(D))$ is chordal-bipartite, then we can apply the polynomial-time algorithm for Set Cover on chordal-bipartite incidence graphs, see [WN99, pp. 562-573] and [HKS85]. It remains to show that $\mathcal{U}(F(D))$ is chordal-bipartite. Suppose for a contradiction that $\mathcal{U}(F(D))$ contains an induced cycle $C_{FD} = \{s_1, t_1, \ldots, s_k, t_k, s_{k+1} = s_1\}$, where $s_1, s_2, \ldots, s_k \in V^+(F(D))$ and $t_1, t_2, \ldots, t_k \in V^-(F(D))$, and $k \geq 3$. In order to keep the notation concise, let $t_0 \coloneqq t_k$.

Since C_{FD} is a cycle in $\mathcal{U}(F(D))$ connecting sources and sinks, we have that for $1 \leq i \leq k$, there are directed paths P_i^{i-1} and P_i^i in D such that P_i^{i-1} connects s_i to t_{i-1} and P_i^i connects s_i to t_i. We now construct a cycle C in $\mathcal{U}(D)$ and then show that C is chordless and has length at least k. Let Q_1^1 be any shortest path from s_1 to t_1 in D. Let us assume we already constructed the paths Q_j^j and Q_j^{j-1} for $1 \leq j \leq i \leq k-1$. We now define the paths Q_{i+1}^i and Q_{i+1}^{i+1} in the following way: Q_{i+1}^i is a shortest path from s_{i+1} to Q_i^i in D. If there exist more than one shortest path, then we pick the one whose endpoint is closest to y_i on Q_i^i. We refer to this endpoint by t_i'. Similarly, Q_{i+1}^{i+1} is a shortest path from Q_{i+1}^i to t_{i+1} in D. If there is more than one shortest path, then we pick the one whose starting point is closest to t_i on Q_i^i. We refer to this starting point by s_i'. Finally Q_1^k $(= Q_1^0)$ is a shortest path from Q_1^1 to Q_k^k. Again, if there is more than one such shortest path, then we first pick the one whose starting point is closest to t_1 on Q_1^1 and then whose endpoint is closest to t_k on Q_k^k. We refer to these two vertices by s_1' and t_k', respectively. Now let $C = \{Q_1^1, Q_2^1, \ldots, Q_k^{k-1}, Q_k^k, Q_1^k\}$.

By construction we have that C is a cycle in $\mathcal{U}(D)$. Note that $s_i' \neq t_{i-1}'$ and $s_i' \neq t_i'$, since otherwise s_{i-1} were adjacent to t_i or s_{i+1} were adjacent to t_{i-1} in $\mathcal{U}(F(D))$. Therefore, C is simple and has length at least k. Now assume for a contradiction that C has some chord a. Observe that a connects two distinct paths Q_i^j and Q_k^l (without loss of generality, $i \leq k$ and $j \leq \ell$) only if $i = k$ and $j = \ell - 1$ or $i = k - 1$ and $j = \ell$, respectively, since otherwise C_{FD} is not chordless. On the other hand $i = k$ and $j = \ell - 1$ contradicts the choice of the starting vertex of Q_i^i on Q_i^{i-1}. Similarly, $i = k - 1$ and $j = \ell$ contradicts the choice of the endvertex of Q_{i+1}^i on Q_i^i. Therefore, C is an induced cycle in $\mathcal{U}(D)$ of length at least k, contradicting our assumption that $\mathcal{U}(D)$ has no induced cycles of length at least 6. This concludes the proof of Theorem 3.17. ■

By combining Theorem 3.13, Proposition 3.15, and Theorem 3.17 we obtain the following result.

Corollary 3.18. Robust k-Matching Augmentation *on chordal-bipartite graphs admits a polynomial-time algorithm.*

Source Cover on Graphs of Bounded Treewidth

We provide a fixed-parameter algorithm for Node Weighted Directed Steiner Tree on acyclic digraphs that is single-exponential in the treewidth of the underlying undirected graph and linear in the instance size. Since Source Cover is a restriction of Node Weighted Directed Steiner Tree on acyclic graphs, this implies a polynomial-time algorithm for Source Cover parameterized by the treewidth of the underlying undirected graph. Let us first recall some definitions related to Steiner problems and tree decompositions.

Problem 3.19 (Node Weighted Directed Steiner Tree)**.** Given an acyclic digraph $D = (V, A)$, costs $c \in \mathbb{R}^V_{\geq 0}$, terminals $T \subseteq V$, root $r \in V$, the task is to find a minimum-cost subset $F \subseteq V$, such that r is connected to each terminal in $(F, E(F))$.

Arc Weighted Directed Steiner Tree is the corresponding problem, where the costs are on the arcs of the graph.

A *tree decomposition* of a graph $G = (V, E)$ is a tree T as follows. Each node $x \in V(T)$ of T has a *bag* $B_x \subseteq V$ of vertices of G such that the following properties hold.

- $\bigcup_{x \in V(T)} B_x = V$.
- If B_x and B_y both contain a vertex $v \in V$, then the bags of all nodes of T in the path between x and y contain v as well. Equivalently, the tree nodes containing vertex v form a connected subtree of T.

- For each edge vw in G there is some bag that contains both v and w. That is, for vertices adjacent in G, the corresponding subtrees have a node in common.

The *width* of a tree decomposition is the size of its largest bag minus one. The *treewidth* $tw(G)$ of G is the minimum width among all possible tree decompositions of G.

To solve the Node Weighted Directed Steiner Tree on acyclic digraphs, we use a simple dynamic-programming algorithm over the tree decomposition of the underlying undirected graph of the input digraph D with n vertices.

Note that an optimal tree-decomposition of a graph G can be computed in time $O(2^{O(tw(G)^3)} \cdot n)$ by Bodlaender's famous theorem [Bod96]. Our algorithm intuitively works in the following way and is similar to the dynamic programming algorithm for Dominating Set (see, e.g., [CFK+15, Section 7.3.2]). We interpret a solution to Node Weighted Directed Steiner Tree as follows: each vertex of D may be *active* or not. Each active vertex needs a predecessor that is also active, unless it is the root. The cost to activate a vertex is given by the cost function of the Node Weighted Directed Steiner Tree instance. Starting with all terminals active, it is easy to see that Node Weighted Directed Steiner Tree on acyclic graphs is equivalent to the problem of finding a minimum cost active vertex set satisfying the above conditions. We compute an optimal solution in a bottom-up fashion using a so-called nice tree decomposition of the input graph.

A *nice tree decomposition* limits the structure of the difference of two adjacent nodes in the decomposition. Formally, consider a tree decomposition T of a graph G, rooted in a leaf of T. We say that T is a *nice tree decomposition* if every node $x \in V(T)$ is of one of the following types.

- **Leaf:** x has no children and $B_x = \emptyset$.

- **Introduce:** x has exactly one child y and there is a vertex $v \notin B_y$ of G with $B_x = B_y \cup \{v\}$.

- **Forget:** x has exactly one child y and there is a vertex $v \notin B_x$ of G with $B_y = B_x \cup \{v\}$.

- **Join:** x has two children y and z such that $B_x = B_y = B_z$.

Such a nice decomposition is easily computed given any tree decomposition of G. We define x^+ to be the subtree of T rooted in x: the tree of all vertices not connected to the root in the forest $T - x$, together with x. By B_x^+ we denote the set of vertices contained in all bags of nodes in x^+.

A *coloring* of a bag B_x is a mapping $f : B_x \to \{1, 1_?, 0\}^{|B_x|}$, where the individual colors have the following meaning.

- Active and already covered, represented by a 1, means that the vertex is active and that there is at least one predecessor of it that is either labeled 1 or $1_?$.

- Active and not yet covered, represented by a $1_?$, means that the vertex is active but every predecessor is labeled 0.

- Not active, represented by a 0, means that the vertex is not contained in the solution.

Note that there are $3^{|B_x|}$ colorings of the bag B_x. For a coloring f of x we denote by $\mathrm{OPT}(f, x)$ the minimum cost[1] of a coloring $B_x^+ \to \{1, 1_?, 0\}$ satisfying the following conditions.

a) every vertex of B_x is colored 1, $1_?$ or 0 according to f.

b) every vertex of $B_x^+ \setminus B_x$ is colored 0 or 1.

c) every sink $v \in V^- \cap B_x^+$ is colored either 1 or $1_?$.

d) every vertex $v \in B_x$ with $f(v) = 1$ is either a source or at least one predecessor of v in $D(B_x^+)$ is colored either 1 or $1_?$.

To present the individual steps of the algorithm, assume that we are given a nice tree decomposition of our input graph. Let us say we are currently considering the node x in T and distinguish between the type of node x.

- **Leaf:** set $\mathrm{OPT}(f, x) = 0$ if it is not the root.

[1]Here, a vertex v has a cost $c(v)$ if it is colored 1 or $1_?$ and 0 otherwise.

- **Introduce:** let y be the unique child of x and let $v \notin B_y$ such that $B_x = B_y \cup \{v\}$. The value $\mathrm{OPT}(f, x)$ depends on the type of vertex v and on the coloring g of y. By definition, sinks have to be active and therefore the optimal value is ∞ if $f(v) = 0$. The same is true for sources labeled $1_?$ in f (those do not have predecessors and need to be labeled either 1 or 0). Finally, we set the cost to be ∞ if v is labeled 1 in f and not a source, but non of its predecessors is active in f. Thus we set

$$\mathrm{OPT}(f,x) = \begin{cases} \infty, \text{ if } v \in V^- \text{ and } f(v) = 0 \ , \\ \infty, \text{ if } v \in V^+ \text{ and } f(v) = 1_? \ , \\ \infty, \text{ if } v \notin V^+ \text{ and } f(v) = 1 \text{ and } (\delta^-(v) \cap B_y) \subseteq f^{-1}(0) \ , \\ \min\{\mathrm{OPT}(g,x) : (g,y) \text{ is compatible to } (f,x)\}, \text{ if } f(v) = 0 \ , \\ \min\{\mathrm{OPT}(g,x) : (g,y) \text{ is compatible to } (f,x)\} + c(v) \ , \text{ else,} \end{cases} \tag{3.1}$$

where the pair (g, y) is compatible to (f, x) if the following conditions hold.

  – If $f(v) = 0$, then $g = f|_{B_y}$. As the introduced vertex is not considered to be part of the solution, we can simply keep the coloring of the child node.

  – If $f(v) = 1_?$, then $f^{-1}(0) = g^{-1}(0)$, $f^{-1}(1) = g^{-1}(1) \cup (g^{-1}(1_?) \cap \delta^+(v))$, and $\delta^-(v) \subseteq g^{-1}(0)$. This condition ensures that the introduced vertex can only be labeled $1_?$ if none of its predecessors is labeled 1 or $1_?$.

  – If $f(v) = 1$, then $f^{-1}(0) = g^{-1}(0)$, $f^{-1}(1) = g^{-1}(1) \cup (g^{-1}(1_?) \cap \delta^+(v))$, and, moreover, $\delta^-(v) \setminus g^{-1}(0) \neq \emptyset$ or $v \in V^+$. This conditions says that the introduced vertex can only be labeled 1 if at least one of its predecessors is labeled 1 or $1_?$, unless it is a source.

- **Forget:** let y be the unique child of x and let $v \notin B_x$ such that $B_y = B_x \cup \{v\}$. Then we put

$$\mathrm{OPT}(f,x) = \min\{\mathrm{OPT}(g,y) : f = g|_{B_x}\} \text{ if } g(v) \neq 1_?.$$

We do not allow a vertex labeled $1_?$ to be forgotten, as we can not assure to cover it in later bags. For the remaining cases we simply keep the optimal value.

- **Join:** let nodes y and z be the two children of the join node x with $B_x = B_y = B_z$. We put

$$\mathrm{OPT}(f,x) = \min\{\mathrm{OPT}(g,y) + \mathrm{OPT}(h,z) - \sum_{v \in B_X \setminus (g^{-1}(0) \cap h^{-1}(0))} c(v)\}, \tag{3.2}$$

where the minimum runs over all colorings g of y and h of z with $f^{-1}(0) = g^{-1}(0) = h^{-1}(0)$ and $f^{-1}(1) = g^{-1}(1) \cup h^{-1}(1)$.

- **Root:** as the graph is connected and the root node is a leaf, the root node is a forget node, where its child node contains exactly one vertex in its bag. The algorithm terminates with the output

$$\mathrm{OPT} = \mathrm{OPT}(f,x),$$

where f is the unique coloring of the empty bag x.

Having presented the algorithm, we need to prove Theorem 3.20 by showing the correctness and bounding the running time of the algorithm.

Theorem 3.20. Node Weighted Directed Steiner Tree *on acyclic digraphs can be solved in time $O(5^w \cdot w \cdot n)$ if a tree decomposition of $\mathcal{U}(D)$ of width w is provided.*

Proof. We need to show that the algorithm works correctly and is fixed parameter tractable when parameterized by the treewidth of the underlying graph. Let T be a nice tree decomposition of $\mathcal{U}(D)$ of width w with t nodes.

Claim 1. *The algorithm correctly computes an optimal solution to* Node Weighted Directed Steiner Tree *in acyclic graphs.*

Proof. We show the statement by a straight-forward inductive proof on the decomposition tree. The induction hypothesis states that $\mathrm{OPT}(f,x)$ is the minimum cost of a solution induced by the vertices of B_x^+, satisfying the conditions **(a)-(d)** (see p. 57). The base case are the leaf nodes where the hypothesis clearly holds. Now let the induction hypothesis be true for all descendants of x. We distinguish between the remaining three node types and argue that the induction hypothesis holds in x.

- **Introduce:** let y be the unique child of the introduce node x and let $v \notin B_y$ such that $B_x = B_y \cup \{v\}$. Clearly **(a)** holds and **(b)** holds by the induction hypothesis. By putting $\mathrm{OPT}(f, x)$ to ∞ if $f(v) = 0$ for a sink $v \in V^-$, **(c)** also holds. For **(d)** observe that the notion of compatibility is defined correctly. If $f(v) \in \{1_?, 0\}$ this is trivial. For $f(v) = 1$ note that v has to satisfy the condition that $\delta^-(v) \backslash g^{-1}(0) \neq \emptyset$. Thus Condition **(d)** holds for x. Now for a given coloring f we have to check if $\mathrm{OPT}(f, x)$ is calculated correctly. This is true for the cases in which $\mathrm{OPT}(f, x)$ is set to ∞. So it remains to show that we identify all compatible colorings g for y to calculate the minimum. The case $f(v) = 0$ is trivial. For the cases $f(v) \in \{1, 1_?\}$ observe that g has to satisfy $f^{-1}(0) = g^{-1}(0)$ and $f^{-1}(1) = g^{-1}(1) \cup (g^{-1}(1_?) \cap \delta^+(v))$. Calculating the minimum over all pairs (g, y) compatible to (f, x) is hence correct. Finally, in order to obtain $\mathrm{OPT}(f, x)$ it is clear that we have to add $c(v)$ to the minimum of all compatible colorings (g, y) for (f, x) if $f(v) \neq 0$.

- **Forget:** let y be the unique child of the forget node x and let $v \notin B_x$ such that $B_y = B_x \cup \{v\}$. For a forget node x we put $\mathrm{OPT}(f, x) = \min\{\mathrm{OPT}(g, y) : f = g|_{B_x}\}$ if $g(v) \neq 1_?$. Clearly **(a)**, **(c)** and **(d)** hold by the induction hypothesis. **(b)** also holds as we only allow colorings that satisfy $f(v) \neq 1_?$. Finally it is easily verified that the calculation of $\mathrm{OPT}(f, x)$ is correct.

- **Join:** let nodes y and z be the two children of the join node x with $B_x = B_y = B_z$. By (3.2), a vertex $v \in B_x$ may only be colored 1 if it is colored 1 either in B_y or B_z. As the induction hypothesis holds for y and z, **(a)**-**(d)** also hold for x. It remains to show that $\mathrm{OPT}(f, x)$ is calculated correctly. The considered colorings g and h of y and z have to satisfy $f^{-1}(0) = g^{-1}(0) = h^{-1}(0)$ and $f^{-1}(1) = g^{-1}(1) \cup h^{-1}(1)$. By adding $\mathrm{OPT}(g, y)+\mathrm{OPT}(h, z)$ we count the vertices in the set $B_X \setminus (g^{-1}(0) \cap h^{-1}(0))$ twice. Thus we obtain

$$\mathrm{OPT}(f, x) = \mathrm{OPT}(g, y) + \mathrm{OPT}(h, z) - \sum_{v \in B_X \setminus (g^{-1}(0) \cap h^{-1}(0))} c(v)$$

.

This proves Claim 1. □

Claim 2. *Given T, the running time of the dynamic programming algorithm is bounded by $O(5^w t)$.*

Proof. In each node x of the nice tree decomposition T we consider $O(3^{|B_x|})$ many different colorings f. We bound the running time for a bag by considering the different kinds of bags. For this, note that the interesting steps are the computation of the pairs (g, y) compatible to (f, x) for the minimum in (3.1) and the computation of the minimum in (3.2).

Consider an introduce node x with its unique child y and let $v \notin B_y$ such that $B_x = B_y \cup \{v\}$. Let f and g be colorings for x and y, respectively. For a vertex $u \in B_y$ we consider all possible combinations $(f(u), g(u))$ for the three possible values of $f(v)$ which are given by (3.1).

- In the case $f(v) = 0$ we have that $g = f|B_x$, that is,

$$(f(u), g(u)) = (g(u), g(u)).$$

- In the case $f(v) = 1_?$ we have that

$$(f(u), g(u)) \in \begin{cases} \{(0,0), (1,1), (1, 1_?)\}, & \text{if } u \in \delta^+(v), \\ \{(0,0), (1,1), (1_?, 1_?)\}, & \text{if } u \notin \delta^+(v), \end{cases} \tag{3.3}$$

  and (f, x) and (g, y) are not compatible unless $\delta^-(v) \subseteq g^{-1}(0)$.

- In the case $f(v) = 1$ we allow the same pairs $(f(u), g(u))$ like in the case $f(v) = 1_?$, but (f, x) and (g, y) are not compatible if $\delta^-(v) \subseteq g^{-1}(0)$.

We basically have three different options for the pairs (f, g). Processing through f and g at the same time leads to the total running time for an introduce node of at most $O(3^w)$.

For a join node, let y and z be the two children of x with $B_x = B_y = B_z$. Let f, g, h be colorings of x, y and z, respectively. For a vertex $u \in B_x$ we consider all possible combinations $(f(u), g(u), h(u))$ with

$$(f(u), g(u), h(u)) \in \{(0,0,0), (1_?, 1_?, 1_?), (1, 1_?, 1), (1, 1, 1_?), (1,1,1)\}. \tag{3.4}$$

Here we are given five different options for the triples (f, g, h), and so the total computation time is at most $O(5^w)$.

The overall bottleneck case is when x is a join node since we need to compute (3.2). As we just said, this can be done in $O(5^w)$ time. Since we have t nodes, the total processing time is $O(5^w t)$. This completes the proof of Claim 2. □

By storing the best current solution alongside the $\mathrm{OPT}(f, x)$-values we can compute an optimal solution together with OPT. We do not give details here since this is standard. Finally observe that the algorithm is indeed fixed parameter tractable when parameterized by the treewidth of the underlying graph. This completes the proof of Theorem 3.20. ■

By a simple reduction, we also obtain an FPT-time algorithm for Arc Weighted Directed Steiner Tree on acyclic digraphs. We just subdivide each arc and assign the cost of the arc to the corresponding new vertex. Each old vertex receives cost zero. This transformation does not increase the treewidth.

Furthermore, we can reduce Source Cover to Node Weighted Directed Steiner Tree by adding a new vertex r and connecting r to each source by an arc. The sources have cost one, while all other vertices have cost zero. The root vertex is r and the set of terminals is the set of sinks. Adding a single new vertex increases the treewidth by at most one. As a consequence of this reduction and Theorem 3.20, we obtain the following result.

Corollary 3.21. Source Cover *can be solved in time $O(5^w \cdot w \cdot n)$ if a tree-decomposition of $\mathcal{U}(D)$ of width w is provided.*

By combining Theorem 3.13, Proposition 3.15, Corollary 3.21, and the observation that treewidth is monotone under taking minors, we obtain the following result.

Corollary 3.22. Robust k-Matching Augmentation *parameterized by the treewidth of the input graph is FPT.*

3.4 Weighted Robust Matching Augmentation

In this section we consider the problem Weighted Robust Matching Augmentation. First, we show a close relationship to the problem Directed Steiner Forest in both directions, meaning that on the one hand

Weighted Robust Matching Augmentation is roughly as hard to approximate as Directed Steiner Forest and, on the other hand that using an approximation algorithm for Directed Steiner Forest we obtain an approximation algorithm for Weighted Robust Matching Augmentation.

Second, we classify the complexity of the problem Weighted Robust Matching Augmentation on minor-closed graph classes. In particular, we show that the problem is NP-hard on a minor-closed class $\mathcal{G}$ of graphs if and only if $\mathcal{G}$ contains at least one of the two graph classes $\mathcal{K}^*$ and $\mathcal{P}^*$ which we will define later.

3.4.1 Complexity and Approximation

We first demonstrate that the edge-weighted version of Robust Matching Augmentation is substantially more involved than the unit-cost version. To this end, we show that the approximability of Weighted Robust Matching Augmentation essentially corresponds to the approximability of Directed Steiner Forest. The latter problem is defined as follows:

Problem 3.23 (Directed Steiner Forest)**.** Given a directed graph $G = (V, A)$, k terminal pairs $(s_i, t_i)_{1 \leq i \leq k}$, costs $w \in \mathbb{Z}^A_{\geq 0}$ the task is to find a minimum-cost subgraph $G' \subseteq G$ such that for each $1 \leq i \leq k$, the vertex s_i is connected to t_i in G'.

We are now ready to give our main result of this subsection.

Proposition 3.24. *Let n' be the number of vertices of a* Weighted Robust Matching Augmentation *instance and n and k be the number of vertices and terminal pairs of a* Directed Steiner Forest *instance, respectively.*

A polynomial-time $f(n')$-factor approximation algorithm for Weighted Robust Matching Augmentation *implies a polynomial-time $f(4n + 2k)$-factor approximation algorithm for* Directed Steiner Forest. *Furthermore, a polynomial-time $f(n)$-factor (resp., $f(k)$-factor) approximation algorithm for* Directed Steiner Forest *implies a polynomial-time $(f(n) + 1)$-factor (resp., $(f(k) + 1)$-factor) approximation algorithm for* Weighted Robust Matching Augmentation.

Proof. We first prove the following statement: an $f(n')$-factor approximation algorithm for Weighted Robust Matching Augmentation implies an $f(n+k)$-factor approximation algorithm for Directed Steiner Forest. Let I be a feasible instance of Directed Steiner Forest with input graph $D = (V, A)$, $|V| = n$, costs $c \in \mathbb{Z}^{A}_{\geq 0}$, and terminal pairs $(s_1, t_1), \ldots, (s_k, t_k) \in V$. Without loss of generality, let $S = \bigcup_{i \in [k]} \{s_i\}$ be the set of sources and let $T = \bigcup_{i \in [k]} \{t_i\}$ be the set of sinks of D. We may also assume that $(s_i, t_i) \notin A$ for all $i \in [k]$. In the reduction it is important that each terminal is a unique vertex, i.e. $t_i \neq t_j$ for all $i \neq j$, $i, j \in [k]$. We ensure this by introducing a copy of each terminal t_i and then connect it to all neighbors of the original vertex, resulting in a graph of at most $n + k$ vertices.

To obtain an instance I′ of Weighted Robust Matching Augmentation, we create a bipartite graph $G = (U + W, E)$, a cost function $c' \in \mathbb{Z}^{\overline{E}}_{\geq 0}$, and a perfect matching M of G in the following way.

For each $v \in V$ we add the vertices u_v and w_v, and the edge $u_v w_v$ to G and M. For each $i \in [k]$ we additionally add the edge $u_{s_i} w_{t_i}$ to E. For each matching edge $u_v w_v \in M$ with $v \notin \{t_1, \ldots, t_k\}$ we add the vertices u'_v and w'_v and the path $u_v w'_v u'_v w_v$ to G, and we add the edge $w'_v u'_v$ to M. Observe that $n' = |U| + |W| \leq 4n + 2k$.

For each $a = vv' \in A$, note that $e_a = w_v u_{v'}$ is an element of $\overline{E}$ and set $c'(e_a) = c(a)$. Let $\overline{E}_A \coloneqq \{e_a \mid a \in A\}$ be this set of edges. Every remaining edge $e \in \overline{E}$ has cost $c'(e) = 1 + f(n+k) \cdot \sum_{a \in A} c(a)$ such that this edge is not contained in any $f(n')$-approximative solution. This completes the construction of G, c', and M. Observe that this transformation can be performed in polynomial time and that M is indeed a perfect matching of G. Additionally, there is a one-to-one correspondence between arcs in A to edges in $\overline{E}_A$ as stated in Fact 3.3: buying an arc in A is equivalent to buying the corresponding edge in $\overline{E}_A$.

We now show that a feasible solution to I can be transformed in polynomial-time to a feasible solution of I′ of the same cost. Let $X \subseteq A$ be a feasible solution to I of cost $c(X)$ and let X' be the corresponding edges to X in $\overline{E}$. At first observe that $c(X) = c'(X')$. We now show that X' is feasible to I′. By the one-to-one correspondence of arcs in D and edges in $\overline{E}$, we have that in $G + X'$ there is an alternating path P_i from u_{s_i} to w_{t_i} for each $i \in [k]$. Thus, by adding the edge $w_{t_i}u_{s_i}$ to P_i, we obtain an alternating cycle through $u_{t_i}w_{t_i}$ for each $i \in [k]$. It follows that X' is feasible. Now let X' be a solution to I' of cost $c'(X')$. Let $X \subseteq A$ be the edges corresponding to the edge set $X' \cap E_A$. Observe that $c(X) = c(X')$. We now show that X is a feasible solution to I. As X' is feasible to I', we have by Fact 3.4 that every vertex is contained in a directed cycle in $D(G + X, M)$. As a directed cycle through u_{t_i} has to use the edge $u_{t_i}u_{s_i}$ (since no terminal vertex appears more than once in I), the directed cycle in $D(G + X, M)$ has to go through u_{s_i}. This implies that there is a directed path from s_i to t_i in $D[X]$ for each $i \in [k]$ and therefore the feasibility of X. Finally, as $n' = |U| + |W| = O(|V| + k)$, we have proved the first part of the proposition.

We now prove the second part: an $f(n)$- or $f(k)$-factor approximation algorithm for Directed Steiner Forest implies an $(f(n) + 1)$- or $(f(k) + 1)$-factor approximation algorithm for Weighted Robust Matching Augmentation, respectively. Let I be an instance of Weighted Robust Matching Augmentation with $G = (U + W, E)$ and $c \in \mathbb{Z}^{\overline{E}}_{\geq 0}$. We define $c^* \in \mathbb{Z}^{E \cup \overline{E}}_{\geq 0}$ by $c^*(e) = c(e)$ if $e \in \overline{E}$ and $c^*(e) = 0$, otherwise. Let $M = \{u_1w_1, \dots, u_nw_n\}$ be any cost minimal perfect matching with respect to c^*, where $n = |U| = |W|$. We construct the Directed Steiner Forest instance I' with $D = (V', A)$, the terminal pairs $(s_1, t_1), \dots, (s_k, t_k)$ and the cost function $c' \in \mathbb{Z}^{A}_{\geq 0}$ in the following way. We set $V' = V$ and add an arc $a_e = uw$ to A if $e = uw \in M$ and add an arc $a_e = wu$ if $e = wu \in (E \cup \overline{E}) \setminus M$. In other words, we direct the matching edges from U to W and the non-matching edges from W to U. The terminal pairs are defined according to the matching, i.e., we set $s_i := w_i$ and $t_i := u_i$. Finally, for every $a \in A$, we let $c'(a) = 0$ if $e \in M$ and $c'(a_e) := c^*(e)$ otherwise. This completes the construction of I'.

Let X' be a feasible solution to I' of cost $c'(X')$. Observe that by the chosen orientations of the arcs in A, any path from w_i to u_i in X implies that there is an alternating path in the corresponding undirected graph with edge set X. Hence, $X \cup M$ is feasible for I. Finally, as M is a cost minimal matching with respect to c^* and $k = O(n)$, we have that $M \cup X$ is an $(f(n)+1)$- or $(f(k)+1)$-factor approximation for Weighted Robust Matching Augmentation if X' is an $f(n)$- or $f(k)$-factor approximation for Directed Steiner Forest. This concludes the proof of Proposition 3.24. ■

On the one hand, Proposition 3.24 implies an $(n^{1/2+\varepsilon}+1)$-factor approximation algorithm for Weighted Robust Matching Augmentation for every $\varepsilon > 0$, due to the results in [CEGS08, FKN09]. On the other hand, an algorithm achieving a guarantee of $n^{1/3}$ or better for Weighted Robust Matching Augmentation would imply a better approximation algorithm for Directed Steiner Forest, as the number k of distinct terminal pairs is at most $O(n^2)$ and the current best approximation factor for Directed Steiner Forest in terms of n is $n^{2/3+\varepsilon}$ due to a result of Berman et al. [BBM+13]. Additionally, by a result of Halperin and Krauthgamer [HK03], Proposition 3.24 implies the following lower bound.

Corollary 3.25. *For every $\varepsilon > 0$* Weighted Robust Matching Augmentation *does not admit a $\log^{2-\varepsilon}(n)$-factor approximation algorithm unless $\mathsf{NP} \subseteq \mathsf{ZTIME}(n^{\mathsf{polylog}(n)})$.*

Proof. Observe that by the construction in the proof of Proposition 3.24, we have that the number of vertices in the Weighted Robust Matching Augmentation instance is quadratic in the number of vertices from the Directed Steiner Forest instance. Hence, by [HK03] and Proposition 3.24, we have that for every $\varepsilon > 0$ Weighted Robust Matching Augmentation does not admit a $\log^{2-\varepsilon}(n)$-factor approximation algorithm unless $\mathsf{NP} \subseteq \mathsf{ZTIME}(n^{\mathsf{polylog}(n)})$. ■

Given this negative result we proceed to identify structural features that lead to polynomial-time algorithms for Weighted Robust Matching Augmentation.

3.4.2 Dichotomy Result

The main result of this section is a classification of the complexity of the problem Weighted Robust Matching Augmentation on minor-closed graph classes. In particular we show that the problem is NP-hard on a minor-closed class $\mathcal{G}$ of graphs if and only if $\mathcal{G}$ contains at least one of the two graph classes $\mathcal{K}^*$ and $\mathcal{P}^*$, which are defined as follows. Let $K_{1,r}$ be the star graph with r leaves and let P_r be the path on r vertices. For any graph H let H^* be the graph obtained by attaching a leaf to each vertex of H. Then

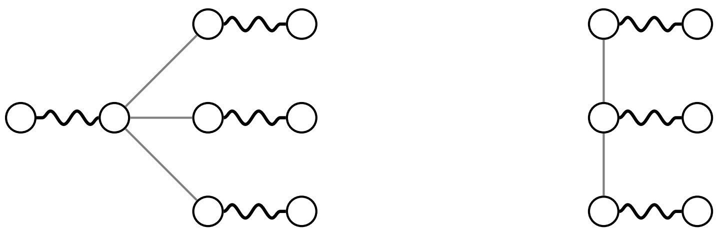


Figure 3.3: The graphs $K^*_{1,3}$ and P^*_3, each with its unique perfect matching.

$\mathcal{K}^* := \{K^*_{1,r} \mid r \in \mathbb{N}\}$ and $\mathcal{P}^* := \{P^*_r \mid r \in \mathbb{N}\}$. Note that each graph in $\mathcal{K}^*$ and $\mathcal{P}^*$ has a unique perfect matching. See Figure 3.3 for an illustration of the graphs $K^*_{1,3}$ and P^*_3.

In order to prove the hardness, we first need the following lemma.

Lemma 3.26. Weighted Robust Perfect Matching Augmentation *on independent edges is* NP*-hard.*

Proof. We give a reduction from Robust Matching Augmentation, which was proved to be NP-hard in Proposition 3.10. Let $\mathrm{I} = (G, M)$ be an instance of Robust Matching Augmentation, where $G = (V, E)$. We construct an instance $\mathrm{I}' = (G', M, c)$ of Weighted Robust Matching Augmentation as follows: Let $G' := (V, M)$ consist only of edges from the perfect matching M. Furthermore, let the costs $c \in \mathbb{Z}^{\overline{E(G')}}_{\geq 0}$ be given by

$$c(e) := \begin{cases} 0, & \text{if } e \in E(G) \setminus M, \\ 1, & \text{otherwise.} \end{cases}$$

Clearly, the construction can be performed in polynomial time. The solutions of I and I′ are in one-to-one correspondence and the costs are preserved by the transformation. ■

We are now ready to state and prove the hardness result for this section.

Lemma 3.27. Weighted Robust Matching Augmentation *is* NP*-hard on each of the classes $\mathcal{K}^*$ and $\mathcal{P}^*$.*

Proof. The result follows in large parts from Lemma 3.26. The main idea is that any instance of Weighted Robust Matching Augmentation on independent edges can be embedded in a sufficiently large member of $\mathcal{K}^*$ or $\mathcal{P}^*$. More formally, consider an instance $\mathrm{I} = (G, M, c)$ of Weighted Robust

Matching Augmentation, where G consists of independent edges. Let (U, W) be any bipartition of $V(G)$.

We first prove the statement for the class $\mathcal{K}^*$. We construct an instance $\mathrm{I}' = (G', M', c')$ of Weighted Robust Matching Augmentation from I, where $G' = K^*_{1,|M|+1}$. Let G' contain the independent edges M and a path $P = v_1, v_2, v_3, v_4$, where v_1, v_2, v_3, v_4 are new vertices. For each $u \in U$, connect v_2 to u by an edge. Observe that $M' := M \cup \{v_1v_2, v_3v_4\}$ is a perfect matching of G'. The costs $c' \in \mathbb{Z}_{\geq 0}^{\overline{E(G')}}$ are given by

$$c'(e) := \begin{cases} c(e), & \text{if } e \in \overline{E(G)}, \\ 0, & \text{if } e = v_1v_4, \\ K, & \text{otherwise,} \end{cases}$$

where K is chosen such that no optimal solution contains an edge of weight K, for example, $K := |V(G')| \cdot \max_{e \in \overline{E}} c(e)$. Since we may add v_1v_4 to any solution at no cost, we assume that it is present in any solution. Now, from the definition of c' it follows that an solution optimal solution to I is also an optimal to I$'$ and vice versa.

It remains to prove the statement for the class $\mathcal{P}^* = \{P^*_t \mid r \in \mathbb{N}\}$. In the following, let $n := |M|$. We construct an instance $\mathrm{I}'' = (G'', M'', c'')$ of Weighted Robust Matching Augmentation from I, where $G'' = P^*_{2n}$. Let G'' contain the independent edges M and join the vertices U in any order by a path $P = v_1, u_1, v_2, u_2, \ldots, v_n, u_n$, where $u_1, u_2, \ldots, u_n \in U$ and $v_1, v_2, \ldots, v_n$ are new vertices. Finally, for each $1 \leq i \leq n$, add a new vertex v'_i to G' and join it to v_i by an edge. Let $M' := M \cup \{v_iv'_i \mid 1 \leq i \leq n\}$ and let $c'' \in \mathbb{Z}_{\geq 0}^{\overline{E(G'')}}$ be given by

$$c(e)'' := \begin{cases} c(e), & \text{if } e \in \overline{E(G)} \\ 0, & \text{if } e = v'_1v_n \text{ or } e = v_iv'_{i+1},\ 1 \leq i < n \\ K, & \text{otherwise,} \end{cases}$$

where K is again chosen such that no optimal solution contains an edge of weight K, for example, $K := |V(G')| \cdot \max_{e \in \overline{E}} c(e)$. By the choice of c'', we may assume that each edge in $M'' \setminus M$ is contained in an alternating cycle. Furthermore, since no optimal solution to I$''$ connects $V(G'') \setminus V(G)$ to $V(G)$, we have that any optimal solution to I$''$ is optimal for I and vice versa. ■

We complement Lemma 3.27 by showing that Weighted Robust Matching Augmentation on a class $\mathcal{G}$ of graphs admits a polynomial-time algorithm if $\mathcal{G}$ contains neither $\mathcal{K}^*$ nor $\mathcal{P}^*$.

Theorem 3.28. *Let $\mathcal{G}$ be a class of connected graphs that is closed under connected minors. Then* Weighted Robust Matching Augmentation *on $\mathcal{G}$ admits a polynomial-time algorithm if and only if there is some $r \in \mathbb{N}$ such that $\mathcal{G}$ contains neither the graph $K^*_{1,r}$ nor P^*_r. The only if part holds under the assumption that* $\mathsf{P} \neq \mathsf{NP}$.

Before we give the proof of Theorem 3.28, we need the following key lemma. The polynomial-time algorithm described in the proof of the lemma uses the fact that Directed Steiner Forest can be solved in polynomial time if the number of terminal pairs is constant [FR06].

Lemma 3.29. *Let $r \in \mathbb{N}$ be constant and let $\mathcal{T}$ be a class of perfectly matchable trees, each with at most r leaves. Then* Weighted Robust Matching Augmentation *on $\mathcal{T}$ admits a polynomial-time algorithm.*

Proof. Let $\mathrm{I} = (G, M, c)$ be an instance of Weighted Robust Matching Augmentation, where $G = (V, E) \in \mathcal{T}$ is a tree with at most r leaves and a given bipartition (U, W). Moreover, let M be the unique perfect matching of G. We say that an arc xy is a *shortcut* if there is an additional directed path from x to y in $D(G, M)$.

Claim 1. *Let L be an optimal solution to I. Then we may assume that $D(G + L, M)$ contains no shortcut.*

Proof. By Fact 3.4, each strongly connected component of $D(G + L, M)$ is non-trivial. Suppose for a contradiction that $D(G+L, M)$ contains a shortcut arc a and let $e \in \overline{E}$ be the edge corresponding to a. Then each strongly connected component of $D(G + (L - e), M)$ is non-trivial. Since the costs c are non-negative, we conclude that $L - e$ is solution of weight at most $\mathrm{OPT}(\mathrm{I})$. This completes the proof of Claim 1. $\square$

By Claim 1 we only need to augment edges that do not correspond to shortcuts in $D(G, M)$. So let $\tilde{E} \subseteq \overline{E}$ be the subset of edges that are useful for augmentation, that is,

$$\tilde{E} \coloneqq \{uw \in \overline{E} \mid D(G + uw, M) \text{ has no shortcut}\}.$$

For $F \subseteq E$, we denote by F_{WU} the set of arcs obtained from F by directing all edges from W to U. We construct a new directed graph D' on the vertices V by directing all M-edges from U to W and making each edge in $E \setminus M$ bidirected.

Claim 2. *Let $L' \subseteq \tilde{E}$. Then $G + L'$ is robust if and only if $D' + L'_{WU}$ is strongly connected.*

Proof. First assume that $G + L'$ is robust and let $uw \in M$. Then uw is contained in some M-alternating cycle C in $G + L'$. It is readily verified that there is a corresponding directed cycle in $D' + L'_{WU}$ containing the arc uw. Therefore, there is a path from w to u in D'. Since the edges in $E \setminus M$ are undirected in D', it follows that $D' + L'_{WU}$ is strongly connected. Now suppose that $D' + L_{WU}$ is strongly connected. Thus, each M-edge is contained in some cycle. Since $L' \subseteq \tilde{E}$, each M-edge is contained in an M-alternating cycle of $G + L'$, so $G + L'$ is robust. This completes the proof of Claim 2. $\square$

Using the two claims above we finish the proof of the lemma. By Claim 2, our task is to find a minimum-weight set $L' \subseteq \tilde{E}$, such that $D' + L'$ is strongly connected. For this purpose, we construct in polynomial time an instance I' of Directed Steiner Forest with at most r terminal pairs, such that from an optimal solution of I' we obtain an optimal solution of I in a straightforward manner. Since the number of terminals r is constant, we can solve the Directed Steiner Forest instance I' in polynomial-time using the algorithm from [FR06] and obtain a solution of I in polynomial time.

The digraph of the instance I' is $D' + \tilde{E}_{WU}$ and the arc-costs c' of I' are given as follows. For each arc uw of $D' + \tilde{E}_{WU}$, let $c'(uw)$ be

$$c'(uw) := \begin{cases} c(uw), & \text{if } uw \in \tilde{E}_{WU}, \\ 0, & \text{otherwise.} \end{cases}$$

The terminal pairs of I' are given as follows. We run the algorithm Eswaran-Tarjan on $D(G, M)$ and obtain an arc-set L such that $D(G, M) + L$ is strongly connected. By Fact 3.7, we have $|L| = \max\{|V^+(D)|, |V^-(D)|\} \leq r$. Each arc $a \in L$ corresponds to a pair of terminals we wish to connect. This completes the construction of I'.

We now show that optimal solutions of I' correspond to optimal solutions of I. Let L' be an optimal solution to I'. That is, $D' + L'$ is strongly connected.

We may assume that L' contains all arcs of D', since each of them has cost 0. Since $L' \subseteq \tilde{E}_{WU}$, we invoke Claim 2 and have that $G + \mathcal{U}(L')$ is strongly connected, where $\mathcal{U}(L')$ are the undirected edges corresponding to L'. Let $L \subseteq \tilde{E}$ such that $G + L$ is robust and assume that $c(L) < c(\mathcal{U}(L'))$. Then L_{WU} is a solution of I' and $c'(L_{WU}) < c'(L')$. This contradicts the optimality of L', so $\mathcal{U}(L')$ is optimal for I. This concludes the proof of Lemma 3.29. ■

We remark that the running time of the algorithm given in Lemma 3.29 is slicewise polynomial in the number of leaves of the input graph. We can now state the proof of our main result.

Proof of Theorem 3.28. Due to Lemma 3.27, Weighted Robust Matching Augmentation is NP-hard if $\mathcal{G}$ contains the class $\mathcal{K}^* = \{K^*_{1,r} \mid r \in \mathbb{N}\}$ or the class $\mathcal{P}^* = \{P^*_t \mid r \in \mathbb{N}\}$. Assuming $\mathsf{P} \neq \mathsf{NP}$, this proves the *only if* statement of the theorem.

To see the *if* statement, let us consider $r \in \mathbb{N}$ such that $\mathcal{G}$ does not contain $K^*_{1,r}$ or P^*_r. First we will reduce the problem to the case where $\mathcal{G}$ contains only trees. For this, let $\mathcal{T}$ be the class of all trees in $\mathcal{G}$ that admit a perfect matching.

Claim 1. *There is a polynomial-time reduction from* Weighted Robust Matching Augmentation *on $\mathcal{G}$ to* Weighted Robust Matching Augmentation *on $\mathcal{T}$.*

Proof. To see this, consider an input (G, M, c) of Weighted Robust Matching Augmentation on $\mathcal{G}$, consisting of a bipartite graph $G \in \mathcal{G}$, a perfect matching M of G, and costs c of edges in the bipartite complement of G. We first compute a spanning tree T of G that contains all edges of M using, e.g. Kruskal's algorithm. We extend the costs c to all edges e in the set $E(G) \setminus E(T)$ by setting $c(e) = 0$.

Note that (T, M, c) is an instance of Weighted Robust Matching Augmentation on $\mathcal{T}$. Moreover, for every optimal solution S of the instance (T, M, c), the set $S \setminus E(G)$ is an optimal solution of the instance (G, M, c). This proves Claim 1. □

Hence we may restrict our attention to Weighted Robust Matching Augmentation on the class $\mathcal{T}$. As the next claim shows, the relevant trees contained in $\mathcal{T}$ have a bounded number of leaves.

Claim 2. *There is some number $f(r)$ depending only on r such that every tree in $\mathcal{T}$ has at most $f(r)$ many leaves.*

Proof. Let $T \in \mathcal{T}$ be arbitrary and ℓ be the number of leaves of T. First we show that the maximum degree of T is bounded by r. Fix any perfect matching M of T. Consider a vertex v of T, and let X be the set of all neighbors of v together with their matching partners. Note that $T[X]$ is isomorphic to $K^*_{1,d(v)}$, where $d(v)$ denotes the degree of v. Since $\mathcal{G}$ is closed under taking connected minors, $K^*_{1,d(v)} \in \mathcal{G}$, and hence $d(v) < r$.

Next, we show that the number of vertices of degree at least 3 is bounded. Since the maximum degree of T is bounded by r, the following holds for the number of leaves in T:

$$\ell = 2 + \sum_{j=3}^{r}(j-2)|V_j|, \text{ where } V_j = \{v \in V(T) : d(v) = j\}.$$

The above formula is a standard graph theory exercise. As r is constant, this implies $\sum_{j=3}^{r}|V_j| = \Omega(\ell)$. Again since r is constant, there is a path in T containing $\Omega(\log \ell)$ many vertices of degree at least 3 in T. Let T' be this path together with all vertices adjacent to it.

Note that P^*_t is a minor of T' where $t+2$ is the number of vertices of degree at least 3 on T. Since $\mathcal{G}$ is closed under connected minors and $P^*_r \notin \mathcal{G}$, we have $t < r$. Consequently, $t \in \Omega(\log \ell)$ implies that $\ell \leq f(r)$ for some number $f(r)$ depending only on r. This proves Claim 2. □

According to the above claims, there is a polynomial-time reduction from Weighted Robust Matching Augmentation on $\mathcal{G}$ to Weighted Robust Matching Augmentation on a class of trees with a bounded number of leaves. Hence, Lemma 3.29 implies that Weighted Robust Matching Augmentation on $\mathcal{G}$ can be solved in polynomial time. This concludes the proof of Theorem 3.28. ■

3.5 Conclusion

We presented algorithms for the task of securing matchings of a graph against the failure of a single edge. For this, we established a connection to the classical strong connectivity augmentation problem. Not surprisingly, the unit weight case is more accessible, and we were able to give a $\log_2 n$-factor approximation algorithm, as well as polynomial-time algorithms for graphs of bounded treewidth and chordal-bipartite graphs. For general non-negative weights, we showed a close relation to Directed Steiner Forest in terms of approximability and gave a dichotomy theorem characterizing minor-closed graph classes which allow a polynomial-time algorithm.

In our opinion, the case of a single edge failure is well understood now and one might go for the case of a constant number of edge failures next. Let us remark that if the number of edge failures is a part of the input, even checking feasibility is NP-hard [DMP+15, LMMP12].

Chapter 4

Robust Disjoint s-t-Paths

4.1 Introduction

In this chapter we consider the problem Bulk-robust k-Disjoint Paths. Given an undirected graph, two terminals $s, t \in V$ and a set of subsets of edges $\mathcal{F}$, find a minimum cost subset of edges that contains k disjoint s-t-paths, no matter which of the sets in $\mathcal{F}$ is deleted. More formally, in this chapter we study the following problem.

Problem 4.1 (Bulk-robust k-Disjoint Paths)**.** We are given an (undirected or directed) graph $G = (V, E)$, two vertices $s, t \in V$, costs $c \in \mathbb{Z}^E_{\geq 0}$, an integer $k \in \mathbb{N}$, and a collection of interdiction sets $\mathcal{F} = \{F_1, F_2, \dots, F_p\}$ with $F_i \subseteq E$ for all $i \in [p]$. The task is to find a set $X \subseteq E$ of minimum cost, such that the graph $G[X - F_i]$ contains k edge-disjoint s-t-paths for every $i \in [p]$.

The edges in $\bigcup_{1 \leq i \leq p} F_i$ are called *unsafe* edges. The width ℓ of the set $\mathcal{F}$ is the size of a largest interdiction set in $\mathcal{F}$, i.e. $\ell = \max_{F \in \mathcal{F}} |F|$.

We briefly discuss the approximability of the different variants of Bulk-robust k-Disjoint Paths depending on the parameters ℓ and k and whether the graph is directed or undirected.

It is not hard to see that Bulk-robust k-Disjoint Paths generalizes both Bulk-robust Shortest Path and Weighted Robust Matching Augmentation. The first part directly follows by setting $k = 1$. For the latter part we set $\ell = 1$ and use the standard reduction from s-t flows to bipartite matchings. We defer the precise reduction to Section 4.3.1. All these results hold

k \ ℓ	0	1	$O(1)$	$O(n)$
$O(1)$	P	?	?	$(1-\varepsilon)\ln n$
$O(n)$	P	$\log^{2-\varepsilon}(n)$	$\log^{2-\varepsilon}(n)$	$\log^{2-\varepsilon}(n)$

Table 4.1: Hardness of approximation for different parameters of ℓ and k on undirected graphs. The symbol 'P' represents that the problem is tractable and a '?' represents an unknown complexity status.

for directed and undirected graphs. Therefore, Bulk-robust k-Disjoint Paths does not admit a polynomial-time $(1-\varepsilon)\ln n$-approximation algorithm if $\ell = O(n)$ unless $\mathsf{P} = \mathsf{NP}$. Furthermore, if $k = O(n)$, then Bulk-robust k-Disjoint Paths does not admit a polynomial-time $\log^{2-\varepsilon}(n)$-approximation unless $\mathsf{NP} \subseteq \mathsf{ZTIME}(n^{\mathsf{polylog}(n)})$. An overview of inapproximability results can be found in Table 4.1.

Thus, if the number k of disjoint paths is large, there is not much hope for a 'good' polynomial-time approximation algorithm, since already for $\ell = 1$ the problem is very hard to approximate. Therefore, when designing approximation algorithms we (mostly) focus our attention to the case where k is at most some constant (however, if $k = O(n)$ and $\ell = 1$ we are able to design a polynomial-time $(k+1)$-approximation algorithm). Furthermore, already the approximation algorithm for Bulk-robust Shortest Path only runs in polynomial-time if ℓ is at most some constant. We generalize this approximation algorithm to the case where k is constant, but still the running time is polynomial only if ℓ is at most some constant.

To summarize, when designing approximation algorithms we (mostly) restrict ℓ and k to be at most some constants with the exception for $\ell = 1$ and $k = O(n)$.

Finally, Bulk-robust k-Disjoint Paths on *directed* graphs generalizes the same problem on undirected graphs in the following way: for each undirected edge we can add two antiparallel directed edges connecting the same endpoints. It is not hard to see that for each solution in the directed graph we can construct a corresponding solution in the undirected graph and vice versa. A solution in the directed graph has at most twice the cost of the corresponding solution in the undirected graph. Therefore, all positive results for the directed version also apply to the undirected version by multiplying the approximation factor by 2. However, we note here that in case $\ell = 1$, one can show that a solution in a directed graph does not contain antiparallel edges such that the solution in the directed graph and the corresponding solution in the undirected graph have precisely the same cost (and vice versa).

This is not true for negative results. As we will see in Section 4.3.1, for large ℓ the directed version is much harder to approximate than the undirected version.

Summary of Algorithmic Techniques. In the first part of this chapter we present an $O(\log n)$-approximation algorithm for UNDIRECTED ℓ-ROBUST k-DISJOINT PATHS where the width ℓ and the number of disjoint paths k are at most some constants. We first introduce a sequence of relaxations of the original problem with smaller interdiction sets $\mathcal{F}_j$ for $j = 1, \dots, \ell$, where the set $\mathcal{F}_j$ contains all subsets of failure sets of size at most j. Having a solution for the problem with interdiction set $\mathcal{F}_{j-1}$ at hand, we iteratively solve the problem with interdiction set $\mathcal{F}_j$. This problem is called the j-TH AUGMENTATION PROBLEM.

In order to solve these augmentation problems, we show that it suffices to add a collection of paths to S_{j-1} to obtain a feasible solution and that an optimal collection of paths has cost at most twice the global optimum. We then model the task of selecting these paths as a SET COVER problem. Using the greedy algorithm for SET COVER we obtain an $O(\log n)$-approximation in each step. Since ℓ and k are constants this implies an $O(\log n)$-approximation algorithm for UNDIRECTED ℓ-ROBUST k-DISJOINT PATHS.

In the second part of this chapter we consider the problem DIRECTED 1-ROBUST k-DISJOINT PATHS. We first present a polynomial-time $(k + 1)$-approximation algorithm for arbitrary k that uses a special min-cost-flow algorithm. The main algorithmic result is a polynomial-time algorithm for the augmentation problem for constant k. That is, given k disjoint s-t-paths X_0 at zero cost, the task is to find a set of edges $Y \subseteq E$ of minimum cost such that $X_0 \cup Y$ is feasible. Similar to the residual graph used in classic maximum flow algorithms, we define a residual graph that depends on the edges in X_0 and a solution Y. We then show that in this residual graph each unsafe edge

$k \backslash \ell$	0	1	$O(1)$	$O(n)$
$O(1)$	1	2	$O(\log n)$	$\times$
$O(n)$	1	$k+1$	$\times$	$\times$

Table 4.2: Approximation factors for different parameters k and ℓ. The symbol '$\times$' represents that we did not design an approximation algorithm for these parameters.

is contained in a strongly connected component. Having this result at hand, we are able to use a dynamic programming approach, in which we need to make certain subgraphs strongly connected. We obtain these strongly connected subgraphs by solving instances of Directed Steiner Forest with k terminal pairs. Since k is constant, we can use the algorithm of Feldman and Ruhl [FR06] to obtain an optimal solution in polynomial time. The most difficult part is to prove that the dynamic program indeed computes an optimal solution. As an immediate consequence we obtain a 2-approximation algorithm for Directed 1-robust k-Disjoint Paths for constant k. A summary of approximation factors for different parameters of ℓ and k can be found in Table 4.2.

Organization of the Chapter. In the next Section we consider the general problem ℓ-robust k-Disjoint Paths. Therein, we first establish the hardness results in Section 4.2.1 followed by the $O(\log n)$-approximation algorithm for Undirected ℓ-robust k-Disjoint Paths in Section 4.2.2 and conjecture an approximation algorithm for the directed version in Section 4.2.3. In Section 4.3 we consider the problem Directed 1-robust k-Disjoint Paths. We first show in Section 4.3.1 the close relationship between Directed 1-robust k-Disjoint Paths and Weighted Robust Matching Augmentation to prove the inapproximability result for Directed 1-robust k-Disjoint Paths for large k, complemented by an approximation algorithm in Section 4.3.2. The dynamic program for the augmentation problem is provided in Section 4.3.3. In Section 4.3.4 we give a reduction from Directed 1-robust k-Disjoint Paths with constant k to other problems of unknown complexity. Finally, Section 4.4 concludes the chapter.

4.2 ℓ-Robust k-Disjoint Paths

In this section we consider the general problem ℓ-ROBUST k-DISJOINT PATHS. As mentioned in the introduction, Bulk-robustness was introduced by Adjiashvili, Stiller and Zenklusen in [ASZ15]. Therein they considered Bulk-robust versions of the problems MINIMUM MATROID BASIS and SHORTEST PATH. They gave hardness results and approximation algorithms for both problems, including a $\log n$-inapproximability result for BULK-ROBUST SHORTEST PATH and an $O(\log n)$-approximation algorithm on instances with bounded width.

In fact, ℓ-ROBUST k-DISJOINT PATHS is a generalization of BULK-ROBUST SHORTEST PATH for instances of width at most ℓ of BULK-ROBUST SHORTEST PATH. Therefore, all inapproximability results for BULK-ROBUST SHORTEST PATH are carried over to ℓ-ROBUST k-DISJOINT PATHS. We summarize these in Section 4.2.1.

Moreover, we generalize the $O(\log n)$-approximation algorithm for BULK-ROBUST SHORTEST PATH on instances with bounded width to ℓ-ROBUST k-DISJOINT PATHS, where k and ℓ are bounded. In the approximation algorithm in [ASZ15], the authors iteratively solve a relaxation of the initial problem by specifying a certain SET COVER instance which they solve using the greedy approximation algorithm. Most of the results used to prove correctness and polynomial running time of this algorithm carry over to ℓ-ROBUST k-DISJOINT PATHS. The remaining challenge is to prove that the SET COVER instances obtained by the algorithm can still be solved in polynomial time and have the desired approximation guarantee. For convenience, throughout this section, we stick to the notation introduced in [ASZ15].

4.2.1 Complexity of ℓ-Robust k-Disjoint Paths

In this section we investigate the complexity of ℓ-ROBUST k-DISJOINT PATHS. These results directly follow from the results for BULK-ROBUST SHORTEST PATH from [ASZ15] and the fact that ℓ-ROBUST k-DISJOINT PATHS is a generalization of the former problem. In [ASZ15] the authors distinguished between the directed and undirected version of BULK-ROBUST SHORTEST PATH and proved the following results.

Proposition 4.2 ([ASZ15], Proposition 3)**.** Undirected Bulk-Robust Shortest Path *admits no* $(1-\varepsilon)\log|V|$*-approximation algorithm for any* $\varepsilon > 0$ *unless* $\mathsf{P} = \mathsf{NP}$*.*

The proposition, as stated in the original paper, was weaker than the one stated here. This is due to the fact that after the publication of [ASZ15], Dinur and Steurer proved a stronger inapproximability result for Set Cover [DS14], which in turn allows for a stronger inapproximability result for Undirected Bulk-Robust Shortest Path.

For the directed version of Bulk-Robust Shortest Path, Adjiashvili, Stiller and Zenklusen proved the following result.

Proposition 4.3 ([ASZ15], Proposition 4)**.** *There is no* $2^{\frac{1}{4}\ell^{1-\varepsilon}}$*-approximation algorithm for* Directed Bulk-Robust Shortest Path *for any* $\varepsilon > 0$*, where* ℓ *is the width of the instance unless* $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\log\log n})$*.*

Since ℓ-robust k-Disjoint Paths is a generalization of Bulk-Robust Shortest Path we directly obtain the following results.

Proposition 4.4. Undirected ℓ-robust k-Disjoint Paths *admits no* $(1-\varepsilon)\log|V|$*-approximation algorithm for any* $\varepsilon > 0$ *unless* $\mathsf{P} = \mathsf{NP}$*.*

Proposition 4.5. *There is no* $2^{\frac{1}{4}\ell^{1-\varepsilon}}$*-approximation algorithm for* Directed ℓ-robust k-Disjoint Paths *for any* $\varepsilon > 0$*, where* ℓ *is the width of the instance unless* $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\log\log n})$*.*

4.2.2 An Approximation Algorithm for Undirected ℓ-Robust k-Disjoint Paths

In this section we give a polynomial-time approximation algorithm restricted to instances of Bulk-robust k-Disjoint Paths with an undirected graph, a constant width ℓ and a constant number of disjoint paths k. We refer to this problem as Undirected ℓ-robust k-Disjoint Paths.

The algorithm presented in this section is very similar to the one in [ASZ15]. We give the full proof here including all lemmas proved in [ASZ15].

Consider an instance I of Undirected ℓ-robust k-Disjoint Paths of width at most some constant ℓ. We define a sequence of relaxations $\mathrm{I}_0, ..., \mathrm{I}_{\ell-1}$

of the instance I_ℓ. The instance I_j is defined on the same graph, terminals and cost as I but with a different scenario set

$$\mathcal{F}_j = \{R \subseteq E : |R| \leq j \text{ and } \exists F \in \mathcal{F} : R \subseteq F\}. \tag{4.1}$$

In words, the set $\mathcal{F}_j$ contains all subsets of failure sets of size at most j. Clearly, for any $0 \leq i \leq j \leq \ell$, a solution to I_j is also feasible to I_i. Note that I_ℓ is simply I.

On a very high level, the algorithm works as follows. We start by solving the instance I_0, a min-cost-flow problem. In each subsequent iteration $j > 0$ the current solution S_{j-1} is augmented to a solution S_j feasible to I_j. We solve these augmentation problems by stating them as special Set Cover problems. A simple but very useful tool is the following lemma.

Lemma 4.6 (cf. [ASZ15], Lemma 11)**.** *Let $F \in \mathcal{F}_j$ be some interdiction set such that $G[S_{j-1} - F]$ does not admit k disjoint s-t- paths. Then*

1. *$F \subseteq S_{j-1}$*
2. *There is a min-s-t-cut in $G[S_{j-1} - F]$ with value precisely $k - 1$.*

Proof. We prove the statements one by one. Assume the first statement is not true, i.e. there is some $e \in F$ such that $e \notin S_{j-1}$ and let $F' = F - e$. Then $G[S_{j-1} - F] = G[S_{j-1} - F']$ does not admit k disjoint s-t-paths. However, $|F'| = j - 1$ and S_{j-1} is feasible to F_{j-1}, a contradiction.

To prove the second statement, let A be any min-s-t-cut in $G[S_{j-1} - F]$. If $|A| \geq k$, then by the max flow min cut theorem (Chapter 2, Theorem 2.21), we have that $G[S_{j-1} - F]$ contains k disjoint s-t-paths, a contradiction. If $|A| \leq k-2$, then there is some $F' \subseteq F$ with $|F'| = j-1$ such that $G[S_{S_j-1} - F']$ contains an s-t-cut $A' \subseteq A$ of size $k - 1$. This contradicts the feasibility of S_{j-1}. ■

In the context of the j-th augmentation step we call an interdiction set $F \in \mathcal{F}_j$ *critical* if there is a min-s-t-cut in $G[S_{j-1} - F]$ of value $k - 1$. Such cuts are called *infeasible cuts.* Let $A = \delta(V_S) \subseteq E$ be such an infeasible cut and let $V_S \subseteq V$ be the vertices incident to the cut on the side containing s. The infeasible cut can be fixed in the augmentation step by a path $P = v_1, ..., v_i$ with $v_1 \in V_S$, $v_i \notin V_S$ and $E(P) \cap S_{j-1} = \emptyset$. Adding a fixing path for every infeasible cut results in a graph which is feasible for the interdiction set F.

Problem 4.7 (j-TH AUGMENTATION PROBLEM)**.** Given a feasible solution S_{j-1} to I_{j-1} the task is to find a set $X_j \subseteq E - S_{j-1}$ of minimum cost, such that the graph $G[S_{j-1} \cup X]$ is feasible to I_j, i.e. $G[(S_{j-1} - F) \cup X]$ contains k disjoint s-t-paths for every $F \in \mathcal{F}_j$.

The following lemma states an important property of the augmentation problem: restricting our search to paths incurs cost of at most twice the optimum.

Lemma 4.8 (cf. [ASZ15], Lemma 13)**.** *There is a collection of paths $P_1, ..., P_q$ such that $S_{j-1} \cup \bigcup_{i\in[q]} P_i$ is feasible to I_j and $\sum_{i\in[q]} c(P_i) \leq 2\mathrm{OPT}_j$, where OPT_j is the cost of the optimal solution of the j-th augmentation problem.*

Proof. Let X_j^* be the optimal solution to the j-th augmentation problem. X_j^* is a forest as we consider undirected graphs and inclusion-wise minimal solutions of the augmentation problem do not contain cycles. Consider any tree T of the forest X_j^*. By doubling each edge in T we obtain an Eulerian graph. Let V_{j-1} be the vertices incident to the edges in S_{j-1}. Starting from a vertex in V_{j-1} we traverse any Euler tour until we again reach a vertex from V_{j-1}. This path corresponds to our first path P_1. By repeating this until we ended the tour we obtain the paths $P_1, ..., P_q$. Repeating this for every tree in X_j^* and observing that $\sum_{i\in[q]} c(P_i) = 2\mathrm{OPT}_j$ concludes the proof. ■

The algorithm for the j-th augmentation problem works in the following way. We are given a feasible solution S_{j-1} to I_{j-1}. We first compute all shortest paths P_{uv} for every two vertices $u, v \in V[S_{j-1}]$ in the graph $(V, E \setminus S_{j-1})$. Let $\mathcal{F}'_j$ be the set of critical interdiction sets. For every $F \in \mathcal{F}'_j$ we compute all infeasible cuts. These infeasible cuts need to be fixed by a collection of paths in the augmentation problem. We model this task as an instance $\mathrm{I} = (U, \mathcal{S})$ of SET COVER and solve I using the classic greedy-algorithm, achieving a guarantee of $\log |U| + 1$. We defer the formal construction of the SET COVER instance for now but mention here that the size of U corresponds to the number of infeasible cuts.

Therefore, in order to show that the greedy-algorithm runs in polynomial time and has the desired approximation guarantee we prove in the next lemma that the number of infeasible cuts in each iteration is polynomial.

Lemma 4.9. *The number of infeasible cuts in the j-th augmentation problem is at most $O(n^{k+j-1})$.*

Proof. We prove the statement by induction. In the first iteration the set S_0 consists of k edge disjoint paths. It is easy to see that there are at most n^k many s-t-cuts in S_0. Thus the statement holds for $j = 0$.

Now let S_j be a feasible solution to I_j and let $\mathcal{F}'_{j+1}$ be the critical interdiction sets. Additionally let X_j be the edges that were added to S_{j-1} in the j-th augmentation step. We first prove the following claim.

Claim 1. *For every $F \in \mathcal{F}'_{j+1}$ we have that $|F \cap X_j| \leq 1$.*

Proof. Suppose this is not true and $F' := F \cap X_j$ has cardinality at least 2. As $F \in \mathcal{F}'_{j+1}$ we have that $G[S_j - F]$ does not contain k disjoint s-t-paths. Then also $G[(S_j - F') - (F - F')]$ does not contain k disjoint s-t-paths, implying the same for $G[S_{j-1} - (F - F')]$. But as $|F - F'| < k$ we have that S_j was not feasible to I_j, a contradiction. This proves Claim 1. □

By the induction hypothesis there are at most $O(n^{k+j-1)}$ infeasible s-t-cuts for the j-th augmentation problem. As mentioned before, X_j is a forest and thus $|X_j| \leq n$. Combining this with Claim 1 we have that the number of insfeasible s-t-cuts for the $(j+1)$-th augmentation problem is at most $O(n^{k+j-2}) + O(n^{k+j-2}) \cdot n = O(n^{k+j-1})$. This concludes the proof of Lemma 4.9. ■

We now describe the whole algorithm. We first solve I_0 and compute k disjoint paths at minimum cost. At iteration j we have already constructed a solution S_{j-1} which is feasible for I_{j-1}, i.e. feasible for every failure subset of size at most $j-1$. We then model the j-th augmentation problem as a special Set Cover problem $\mathrm{SC}_j = \{U_j, \mathcal{S}_j, c_j\}$ that is obtained as follows. For each infeasible s-t-cut Y in S_{j-1} we add an element u_Y to U_j. For each shortest path P_{uv} in $(V, E \setminus S_{j-1})$ from $u \in V[S_{j-1}]$ to $v \in V[S_{j-1}]$ we add a set S_{uv} to $\mathcal{S}_j$. The cost of this set is $c_j(S_{uv}) = c(P_{uv})$, i.e. the cost of a shortest u-v-path in $(V, E \setminus S_{j-1})$. An element $u_Y \in U_j$ is contained in a set $S_{uv} \in \mathcal{S}_j$ if and only if the path P_{uv} fixes the infeasible s-t-cut Y.

We then solve SC_j using the classic greedy algorithm with an approximation guarantee of $\log n'$, where n' is the number of critical cuts in S_{j-1}. As the width of the scenarios and the number of disjoint paths is constant, this can be done in polynomial time due to Lemma 4.9. As the approximation guarantee in each step is $O(\log n)$ we also achieve an overall approximation guarantee of $O(\log n)$, as again the width of the failure scenarios and the number of disjoint paths is constant. A complete description of the algorithm can be found below.

Algorithm 1: : A polynomial-time $O(\log n)$-approximation algorithm for UNDIRECTED ℓ-ROBUST k-DISJOINT PATHS

```
1: Solve $I_0$ by computing $k$ disjoint $s$-$t$-paths using a min-cost-flow
   algorithm to obtain $S_0$ and set $j = 1$
2: while $j \leq \ell$ do
3:   Compute the SET COVER problem $SC_j = \{U_j, \mathcal{S}_j, c_j\}$ using $S_{j-1}$
4:   Solve $SC_j$ using the greedy-algorithm to obtain $S_j$
5:   $j = j + 1$
6: return $S_\ell$
```

We are now ready to prove the main theorem of this section.

Theorem 4.10. UNDIRECTED ℓ-ROBUST k-DISJOINT PATHS *admits a polynomial-time $O(\log n)$-approximation algorithm for constant width ℓ and a constant number of disjoint paths k.*

Proof. We start by solving I_0 followed by the sequence of ℓ augmentation problems I_1, ..., I_ℓ. For $0 \leq j \leq \ell$ let X_j be the solution to I_j computed by the algorithm and let $X = \bigcup_{0 \leq j \leq \ell} X_j$ be the final solution of the algorithm. The correctness of the algorithm is a direct consequence of Lemma 4.6. The initial problem is solved to optimality using a min-cost-flow algorithm. For the ℓ augmentation problems we solve the SET COVER instances SC_j for $j \in [\ell]$ using the greedy-algorithm. By Lemma 4.8 we only loose a factor of 2 in the approximation guarantee when considering paths as the covering sets. By Lemma 4.9 we have that the number of infeasible s-t-cuts of the j-TH AUGMENTATION PROBLEM is at most $O(n^{k+j})$ and therefore at most $O(n^{k+\ell})$. Hence, the greedy algorithm runs in polynomial time. As the greedy algorithm is a $\log n' + 1$-approximation algorithm, the obtained solution X_j for the j-th augmentation problem has cost at most $c(X_j) \leq 2 \log n' \cdot \mathrm{OPT}_j$, where $n' = O(n^{k+\ell})$ is the number of infeasible s-t-cuts and OPT_j is the value of the optimal solution for the j-TH AUGMENTATION PROBLEM. As k

and ℓ are constants, we have that $c(X_j) \leq O(\log n) \cdot \mathrm{OPT}_j$. Since OPT_j is at most OPT (the value of an optimal solution) and ℓ is constant, we have that $c(X) = \sum_{1 \leq i \leq \ell} c(X_i) \leq \sum_{1 \leq i \leq \ell} O(\log n) \cdot \mathrm{OPT}_j \leq O(\log n) \cdot \mathrm{OPT}$. ■

4.2.3 Directed ℓ-Robust k-Disjoint Paths

In the last section we presented an algorithm for the undirected version of ℓ-ROBUST k-DISJOINT PATHS. In principle, the same algorithm could also be applied to the directed version. The key difference between the undirected and directed version of ℓ-ROBUST k-DISJOINT PATHS is the fact that Lemma 4.8 is not true anymore. It was shown that restricting our search to paths in the augmentation problem incurs cost of at most twice the optimal, which is not true in the directed version of the problem. Note that by Proposition 4.3 it is very unlikely that the directed version admits an approximation guarantee that is only logarithmic in the width ℓ.

Hence, for a similar statement like Lemma 4.8 for the directed version we need a much higher bound that also depends on ℓ. We conjecture the following analogon to Lemma 4.8 for the directed version.

Conjecture 4.11. *For the directed j-TH AUGMENTATION there is a collection of directed paths $P_1, ..., P_q$ such that $S_{j-1} \cup \bigcup_{i \in [q]} P_i$ is feasible to I_j and $\sum_{i \in [q]} c(P_i) \leq f(\ell) \cdot \mathrm{OPT}_j$, where OPT_j is the cost of the optimal solution of the j-th augmentation problem and $f(\ell)$ is a computable function solely depending on ℓ.*

A suitable function for f might be $f(\ell) = 2^\ell$. One potential approach to prove Conjecture 4.11 is to show that any minimal solution to the J-TH AUGMENTATION PROBLEM can be decomposed into at most 2^j (not necessarily disjoint) s-t-paths. Assuming that Conjecture 4.11 is true would prove the existence of an algorithm with the following approximation guarantee for DIRECTED ℓ-ROBUST k-DISJOINT PATHS.

Conjecture 4.12. *There is an $(f(\ell) \cdot O(\log n))$-approximation algorithm for* DIRECTED ℓ-ROBUST k-DISJOINT PATHS *for constant width ℓ and a constant number of disjoint paths k.*

Note that the conjectured approximation guarantee is essentially tight due to Proposition 4.3.

4.3 1-Robust k-Disjoint Paths

In this section we consider the restriction of ℓ-ROBUST k-DISJOINT PATHS to the special case where the width ℓ is 1. The definition of the directed version can thus be simplified as follows.

Problem 4.13 (DIRECTED 1-ROBUST k-DISJOINT PATHS)**.** Given a directed graph $D = (V, E)$, two vertices $s, t \in V$, costs $c \in \mathbb{Z}^E_{\geq 0}$, and a set $F \subseteq E$ of unsafe edges, the task is to find a set $Y \subseteq E$ of minimum cost, such that the graph $G[Y - f]$ contains k disjoint s-t-paths for every $f \in F$.

Note that for large k (i.e. $k = O(|V|)$) this problem generalizes WEIGHTED ROBUST MATCHING AUGMENTATION, which was the main problem considered in Chapter 3. Hence, the hardness results for WEIGHTED ROBUST MATCHING AUGMENTATION carry over to DIRECTED 1-ROBUST k-DISJOINT PATHS. In a nutshell, these hardness results imply that DIRECTED 1-ROBUST k-DISJOINT PATHS is (almost) as hard to approximate as DIRECTED STEINER FOREST. These results are presented in Section 4.3.1.

On the positive side, we first give a polynomial-time $(k + 1)$-approximation algorithm for any $k \in \mathbb{N}$. The algorithm uses the LP-formulation for the problem MINIMUM COST FLOW. Using properties of this LP, we show that the support of a special MINIMUM COST FLOW solution is indeed a $(k + 1)$-approximation for DIRECTED 1-ROBUST k-DISJOINT PATHS. This result is presented in Section 4.3.2.

Furthermore, in Section 4.3.3 we consider the augmentation variant of this problem, which is formally defined as follows.

Problem 4.14 (DIRECTED 1-ROBUST k DISJOINT PATHS AUGMENTATION)**.** We are given a directed graph $D = (V, E)$, the union of k disjoint s-t-paths X_0 from s to t available at zero cost, costs $c \in \mathbb{Z}^{E \setminus X_0}_{\geq 0}$, and a set $F \subseteq E$ of unsafe edges. The task is to find a set $Y \subseteq E \setminus X_0$ of minimum cost, such that $G[(X_0 - f) \cup Y]$ contains k disjoint s-t-paths for every $f \in F$.

In order to solve this problem, we first show that in any feasible solution to DIRECTED 1-ROBUST k DISJOINT PATHS AUGMENTATION, each unsafe edge is contained in a strongly connected component of a certain residual graph, which is closely related to the residual graph used in the classic algorithms for the problem Max s-t-Flow.

We then solve the augmentation problem using a dynamic programming approach. Therein, as a subroutine, we need to solve instances of Directed Steiner Forest on k terminal pairs. Using the feasibility criteria from above, we can show that the solutions obtained by the dynamic programming approach are indeed feasible and, additionally, that optimal solutions of the dynamic program correspond to optimal solutions of Directed 1-robust k Disjoint Paths Augmentation. Furthermore, we can solve the instances of Directed Steiner Forest on k terminal pairs that are obtained in the dynamic programming approach using the algorithm of Feldman and Ruhl [FR06], leading to a running time of roughly $n^{O(k)}$. Note that the running time of the algorithm is slicewise-polynomial and hence we can solve the augmentation problem in polynomial time for constant k. As a direct consequence we obtain a polynomial-time 2-approximation algorithm for Directed 1-robust k-Disjoint Paths for constant k.

Note that both, the $(k+1)$-approximation algorithm and the slicewise polynomial exact algorithm essentially match the best known bounds, either in approximability or running time of the problem: the current best known guarantee for Directed Steiner Forest is $\sqrt{k'}$, where k' is the number of terminal pairs. Since in the reduction k' corresponds to k, improving the approximation guarantee to below $\sqrt{k}$ is a very ambitious task as it implies a better approximation ratio for Directed Steiner Forest. On the other hand, it is known that Directed Steiner Forest does not admit an FPT-algorithm when parameterized by the number of terminal pairs [GNS11]. Hence, the slicewise polynomial algorithm by Feldman and Ruhl [FR06] is essentially best possible in terms of running time. Since Directed 1-robust k Disjoint Paths Augmentation generalizes Directed Steiner Forest with k terminal pairs, the slicewise polynomial algorithm presented in Section 4.3.3 is also essentially best possible in terms of running time.

However, the complexity of Directed 1-robust k-Disjoint Paths for constant k (or even for $k = 2$) is still unknown and a very interesting open question. Therefore, we relate Directed 1-robust k-Disjoint Paths to other famous problems for which the complexity status is also unknown. All of these problems are related to Directed Steiner Forest. The first problem is defined as follows.

Problem 4.15. (2-connected Directed k Steiner Tree) Given a directed graph $D = (V, A)$, a cost function $c \in \mathbb{Q}^E$, and k terminal vertices $t_1, t_2, \ldots, t_k \in V$ together with a root $s \in V$, the task is to find a minimum cost set of edges $X \subseteq E$, such that (V, X) contains 2 edge-disjoint s-t-paths for each $t \in \{t_1, t_2, \ldots, t_k\}$.

We usually set $T \coloneqq \{t_1, t_2, \ldots, t_k\}$. According to [CLNV14], even the complexity of the following special case of 2-connected Directed k Steiner Tree is still open.

Problem 4.16. (1-2-connected Directed 2 Steiner Tree) Given a directed graph $D = (V, A)$, a cost function $c \in \mathbb{Q}^E$ and two terminal vertices $t_1, t_2 \in V$ together with a root $s \in V$, the task is to find a minimum cost set of edges $X \subseteq E$, such that (V, X) contains an s-t_1-path and two edge-disjoint s-t_2-paths.

We investigate the relationship between Directed 1-robust k-Disjoint Paths and 2-connected Directed k Steiner Tree in Section 4.3.4. We show that a special case of Directed 1-robust k-Disjoint Paths corresponds to 2-connected Directed k Steiner Tree with one additional constraint and that a polynomial-time algorithm for Directed 1-robust 2-Disjoint Paths implies a polynomial-time algorithm for 1-2-connected Directed 2 Steiner Tree.

4.3.1 Complexity of Directed 1-robust k-Disjoint Paths

In this section we give a reduction from Weighted Robust Matching Augmentation to Directed 1-robust k-Disjoint Paths, showing that approximating Directed 1-robust k-Disjoint Paths is at least as hard as Weighted Robust Matching Augmentation and therefore at least as hard as Directed Steiner Forest.

Proposition 4.17. *A polynomial-time $f(k)$-factor approximation algorithm for* Directed 1-robust k-Disjoint Paths *implies an $f(\frac{n}{2})$-factor approximation algorithm for* Weighted Robust Matching Augmentation*,* *where n is the number of vertices in the* Weighted Robust Matching Augmentation *instance.*

Proof. Let I be an instance of Weighted Robust Matching Augmentation where $G = (U + W, E)$ is the bipartite graph and $c \in \mathbb{Q}^E$ is the cost function. We use the classical reduction from bipartite matching to the problem Max s-t-Flow. We construct an instance I′ of Directed 1-robust k-Disjoint Paths with a directed graph $D' = (V, A)$, cost function $c' \in \mathbb{Q}^A$, and $F \subseteq A$ in the following way. We set $V = \{U + W \cup \{s\} \cup \{t\}\}$ and $A = A_s \cup A_t \cup A_E$ where $A_s = \{su : u \in U\}$, $A_t = \{wt : w \in W\}$ and $A_E = \{uw : u \in U, w \in W, uw \in E\}$. In words, we add two vertices s and t to G together with all edges from s to U and from W to t. Furthermore, we direct all edges in E from U to W. Additionally we set $k = \frac{1}{2}n$, $F = A_E$, and

$$c'(uw) := \begin{cases} c(uw), & \text{if } uw \in E(G), \\ 0, & \text{otherwise.} \end{cases}$$

Note that there is a one-to-one correspondence between the undirected edges in E and the directed edges in A_E. Hence, for $X \subseteq E$ we refer by $q(X)$ to the corresponding directed edges of A_E. Analogously we define $q^{-1}(Y)$ for some edge set $Y \subseteq A_E$.

Now it is easy to see that for a feasible solution X to I, the edge set $q(X) \cup A_s \cup A_t$ is feasible to I′. Furthermore, a feasible solution Y to I′ corresponds to a feasible solution $q^{-1}(Y \setminus (A_s \cup A_t))$ to I. Also note that, by the definition of c', we have that the cost of these solutions are the same. Since $k = \frac{1}{2}n$, we have that any $f(k)$-factor approximation algorithm for Directed 1-robust k-Disjoint Paths implies an $f(\frac{n}{2})$-factor approximation algorithm for Weighted Robust Matching Augmentation. ■

The following proposition is a direct consequence of Proposition 3.24 from Chapter 3 and Proposition 4.17.

Proposition 4.18. *Let n be the number of vertices of an instance of* Directed 1-robust k-Disjoint Paths *and let n' and k' be the number of vertices and terminal pairs of a* Directed Steiner Forest *instance, respectively. A polynomial-time $f(k)$-factor approximation algorithm for* Directed

1-ROBUST k-DISJOINT PATHS *implies a polynomial-time* $f(2n' + k'))$*-factor approximation algorithm for* DIRECTED STEINER FOREST.

Additionally, by a result of Halperin and Krauthgamer [HK03], Proposition 4.17 implies the following lower bound.

Corollary 4.19. *For every* $\varepsilon > 0$ DIRECTED 1-ROBUST k-DISJOINT PATHS *does not admit a* $\log^{2-\varepsilon}(k)$*-factor approximation algorithm unless* $\mathsf{NP} \subseteq \mathsf{ZTIME}(n^{\mathsf{polylog}(n)})$.

Note that all results presented in this section also hold for the corresponding undirected problem.

4.3.2 A $(k+1)$-approximation Algorithm for Directed 1-Robust k-Disjoint Paths

In this section we present the $(k+1)$-approximation algorithm for DIRECTED 1-ROBUST k-DISJOINT PATHS. The idea is to transform the problem into a MIN COST s-t-FLOW with a predefined capacity function on the edges. We then add every edge to our solution which is in the support of the min-cost-flow. On the one hand, this flow has fractional value at least $\frac{1}{k}$ for every edge in the support of the min-cost-flow. On the other hand the costs are bounded by $(1 + \frac{1}{k})$ times the value of an optimal solution, such that we are able to obtain a $(k+1)$-approximation.

For a given instance I of DIRECTED 1-ROBUST k-DISJOINT PATHS we define an instance of MIN COST s-t-FLOW with capacities g on the edges. We set the capacities as follows.

$$g(e) := \begin{cases} 1, & \text{if } e \in F \\ 1 + \frac{1}{k}, & \text{otherwise} \end{cases}$$

We now want to find a min-cost-flow solution with flow value precisely $k+1$ with respect to the capacities g. Therefore, we would like to solve the following linear program.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{uv \in E} c(uv) \cdot f(uv) \\
\text{subject to} \quad & f(uv) \le g(uv) && \text{for all } uv \in E \\
& f(uv) \ge 0 && \text{for all } uv \in E \\
\sum_{w \in V} f(wu) - & \sum_{w \in V} f(uw) = 0 && \text{for all } u \in V \setminus \{s,t\} \\
& \sum_{w \in V} f(sw) = k+1 \\
& \sum_{t \in V} f(wt) = k+1
\end{aligned}
\tag{4.2}
$$

This problem can be solved in polynomial time using standard min-cost-flow algorithms. Note that the resulting solution does not have to be integral. Let $f^* \in \mathbb{Q}^E_{\geq 0}$ be an optimal solution to LP (4.2). We now claim that the support of the solution, i.e. $Y = \{e \in E : f^*(e) > 0\}$ is the desired $(k+1)$-approximation.

For this, we first show that an optimal solution always contains a flow of value $k+1$ with respect to g. On the other hand we show that the support of a flow of value $k+1$ with respect to g is always feasible.

Lemma 4.20. *Let Y^* be the optimal solution to the instance* I *of* Directed 1-robust k-Disjoint Paths *and let $f^* \in \mathbb{Q}^E_{\geq 0}$ be the min-cost-flow solution with respect to the capacities g. We then obtain the following statements.*

1. *Y^* contains a flow of value $k+1$ with respect to the capacities g.*

2. *$Y = \{e \in E \mid f^*(e) > 0\}$ is a feasible solution to* I.

Proof. We prove the statements one by one. Observe that by the max-flow-min-cut theorem in any feasible solution, every s-t-cut contains either at least k safe edges or at least $k+1$ edges. Otherwise Y^* was not feasible. On the one hand those s-t-cuts Z in $G[Y^*]$ having at least k safe edges satisfy $g(Z) \geq (1 + \frac{1}{k}) \cdot k = k+1$. On the other hand, those s-t-cuts Z' in $G[Y^*]$ containing at least $k+1$ edges satisfy $g(Z') \geq k+1$. Hence, every s-t-cut Z in $G[Y^*]$ has capacity $g(Z) \geq k+1$. The statement now follows by the max-flow-min-cut theorem.

For the second statement, we need to show that Y is feasible to I. Observe that every s-t cut Z in $G[Y]$ has value at least $g(Y) \geq k+1$. Thus, Z contains either at least k safe edges or at least $k+1$ edges. The statement now follows again by the max-flow-min-cut theorem. ∎

Thus the support of the min-cost-flow solution is always a feasible solution. We now need to bound the cost of this solution.

Theorem 4.21. *The algorithm described above is a polynomial-time* $(k+1)$*-approximation algorithm for* Directed 1-robust k-Disjoint Paths.

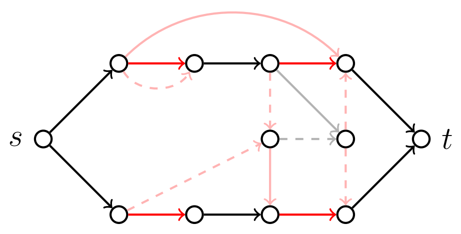
Proof. Let I be an instance of Directed 1-robust k-Disjoint Paths and let OPT be the value of an optimal solution to I. The algorithm solves the min-cost-flow problem with respect to the capacity function g. We then argue that $Y = \{e \in E \mid f^*(e) > 0\}$ is the desired solution. The feasibility of X follows from part 2 of Lemma 4.20. We now bound the cost of the solution. Let $Y_F = Y \cap F$ be the unsafe edges in Y and let $Y_S = Y - Y_F$ be the safe edges. First, we have that each edge $e \in Y$ has flow value at least $b(e) \geq \frac{1}{k}$, since $g(e')$ is either 1 or $1 + \frac{1}{k}$ for every edge $e' \in E$. This is true since one could scale g by a factor k. Then LP 4.2 is integral and one could obtain an integral solution (for the scaled capacity function g). In addition, observe that we may pay a factor $1 + \frac{1}{k}$ too much for every safe edge since the capacity of the safe edges is $1 + \frac{1}{k}$. Therefore we can bound the cost of a safe edge $e \in Y_S$ by $k \cdot (1 + \frac{1}{k}) \cdot c(e) \cdot f^*(e)$ and the cost of each unsafe edge e by $k \cdot c(e) \cdot f^*(e)$. Hence, we obtain

$$\begin{aligned} c(Y) &= c(Y_S) + c(Y_F) \\ &\leq k \cdot \left((1 + \frac{1}{k}) \cdot \sum_{e \in Y_S} c(e) \cdot f^*(e) + \sum_{e \in Y_F} c(e) \cdot f^*(e) \right) \\ &\leq k(1 + \frac{1}{k}) \cdot \left(\sum_{e \in Y_S} c(e) \cdot f^*(e) + \sum_{e \in Y_F} c(e) \cdot f^*(e) \right) \\ &\leq (k+1) \cdot \text{OPT} \ , \end{aligned}$$

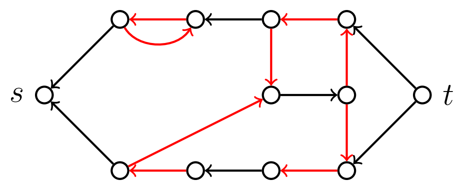
where the first inequality follows from the two arguments above and the last inequality follows from part 1 of Lemma 4.20. ∎

4.3.3 Solving the Augmentation Problem

In this section we give a polynomial-time algorithm for Directed 1-robust k Disjoint Paths Augmentation. We construct a residual graph somewhat



(a) Graph D and X_0 consisting of two disjoint paths.



(b) Residual graph $D_{X_0}(X_0 \cup Y)$.

Figure 4.1: Illustration of the structure of feasible solutions to Directed 1-robust k Disjoint Paths Augmentation. Unsafe arcs are red, safe arcs are black. In Figure a): edges of X_0 are black and red; edges of $A - X_0$ are light gray and light red. Dashed edges belong to Y.

similar to the classic residual graph used for computing maximum flows and show that in this graph all unsafe edges are contained in a strongly connected component. These strongly connected components are connected by safe edges. We reduce the problem of finding those strongly connected components to a special variant of Directed Steiner Forest on k terminal pairs, which can be solved in polynomial time since k is constant. From an optimal solution for each strongly connected component we construct another auxiliary graph. A shortest s-t-path in this auxiliary graph then corresponds to an optimal solution for Directed 1-robust k Disjoint Paths Augmentation.

Throughout this subsection we fix an instance I of Directed 1-robust k Disjoint Paths Augmentation and set $P_1, ..., P_k$ to be the k disjoint s-t-paths of X_0 given by I. We now construct the residual graph that we use as a certificate for feasibility. For a subgraph $H = (V, A^*)$ of D satisfying $X_0 \subseteq A^*$, the residual graph $D_{X_0}(A^*) = (V, A')$ consists of the same vertex set as D. For each arc $a = uv \in A^*$ we add uv to A' if $a \notin X_0$ and add the reversed arc vu to A' if $a \in X_0$. An illustration of this graph is given in Figure 4.1. We show that each unsafe arc of a feasible solution $X_0 \cup Y$ to I is contained in some strongly connected component of $D_{X_0}(X_0 \cup Y)$.

Lemma 4.22. *Let* I *be an instance of* Directed 1-robust k Disjoint Paths Augmentation *and let $Y \subseteq A \setminus X_0$. Then Y is a feasible solution to* I *if and only if each unsafe arc $f \subseteq F \cap X_0$ is contained in some strongly connected component in $D_{X_0}(X_0 \cup Y)$.*

Proof. We first prove the "if" part of the statement, so let $f = uv$ be any unsafe arc in X_0 that is contained in a strongly connected component of $D_{X_0}(X_0 \cup Y)$. As $f \in X_0$, the arc f is reversed in $D_{X_0}(X_0 \cup Y)$ and since f is on a cycle C in $D_{X_0}(X_0 \cup Y)$, there is a path P from u to v in $D_{X_0}(X_0 \cup Y)$. Let P' be the path corresponding to P in $D[X_0 \cup Y]$. Note that P' is not a directed path in D and that an arc e on P' is traversed forward if $e \in P' \cap Y$ and traversed backward if $e \in P' \cap X_0$. We partition P' into two disjoint sets $P'_{X_0} = P' \cap X_0$ and $P'_Y = P' \cap Y$ of arcs. We now argue that $(X_0 - P'_{X_0} - f) \cup P'_Y$ contains k disjoint s-t-paths in $X_0 \cup Y$. Clearly we have that $(X - P'_{X_0} - f) \cup P'_Y \subseteq X_0 \cup Y$. As X_0 is the union of k disjoint paths, we have that in $D[X_0]$, $\delta^+(v) = \delta^-(v)$ for every vertex $v \in V - \{s,t\}$ and $\delta^+(s) = \delta^-(t) = k$. Since C is a cycle in $D_{X_0}(X_0 \cup Y)$ the degree constraints also hold for $D[(X_0 - P'_{X_0} - f) \cup P'_Y]$. Hence $(X_0 - P'_{X_0} - f) \cup P'_Y$ is the union of k disjoint s-t-paths.

We now prove the "only if" part. Let $f = uv \in X_0$ be an unsafe arc and suppose it is not contained in a strongly connected component of $D_{X_0}(X_0 \cup Y)$. Let $L \subseteq V$ be the set of vertices that are reachable from u in $D_{X_0}(X_0 \cup Y)$ and let $R := V - L$. Note that $s \in L$ and $t \in R$ since otherwise there is a path from u to v in $D_{X_0}(X_0 \cup Y)$ (as every arc in X_0 is reversed). Let $L' = \{x_1, \ldots, x_k\} \subseteq V$, $x_i \in P_i$ for all $i \in [k]$, be the vertices of L that are closest to t in $D[X_0]$. We now claim that $\delta^+(L)$ is a cut containing f of size k in $D[X_0 \cup Y]$. Since f is an unsafe arc this implies that $X_0 \cup Y$ was not feasible. Clearly $f \in \delta^+(L)$ in $D[X_0 \cup Y]$, as otherwise f is contained in a strongly connected component in $D_{X_0}(X_0 \cup Y)$. There is no arc $e = u'v' \in Y \cap \delta^+(L)$ in $D[X_0 \cup Y]$ as otherwise $v' \in L$ contradicts the construction of L. Thus $\delta^+(L)$ only consists of arcs of X_0 in $D[X_0 \cup Y]$. Since X_0 is the union of k disjoint paths, $\delta(L)$ has size at most k in $D[X_0 \cup Y]$, proving our claim since this implies that $X_0 \cup Y$ is not feasible. This concludes the proof of Lemma 4.22. ■

We now construct an auxiliary digraph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ to compute an optimal solution for an instance I of Directed 1-robust k Disjoint Paths Augmentation. For clarity we call the elements in V (A) of D vertices (arcs) and elements of $\mathcal{V}$ ($\mathcal{A}$) nodes (links). We define an ordering on the vertices of P_i, $i \in k$. For two vertices $x_i^1, x_i^2 \in P_i$ we say that $x_i^1 \leq x_i^2$ if x_i^1 is closer to s on P_i than x_i^2. Note that if $x_i^1 = x_i^2$, we also have $x_i^1 \leq x_i^2$. Let us now construct the node set $\mathcal{V}$. We add a node v to $\mathcal{V}$ for every k-tuple $v = (x_1, ..., x_k)$ of vertices in $V(X_0)$ satisfying $x_i \in P_i$, for every $i \in [k]$. Note that the corresponding vertices of a node do not necessarily need to be distinct as the k disjoint paths may share vertices. We also define a (partial) ordering on the nodes in $\mathcal{V}$. For two nodes $v_1 = (x_1^1, ..., x_k^1)$ and $v_2 = (x_1^2, ..., x_k^2)$ we say that $v_1 \leq v_2$ if $x_i^1 \leq x_i^2$ for every $i \in [k]$. Additionally let Q_i be the path from x_i^1 to x_i^2 on P_i.

Let us now construct the link set $\mathcal{A}_1$. For two nodes $v_1 = (x_1^1, ..., x_k^1) \in \mathcal{V}$ and $v_2 = (x_1^2, ..., x_k^2) \in \mathcal{V}$ satisfying $v_1 \leq v_2$ we add a directed link v_1v_2 to $\mathcal{A}_1$ if $Q_i \cap F = \emptyset$ for every $i \in [k]$, i.e. if all subpaths from x_i^1 to x_i^2 are safe.

We now construct the link set $\mathcal{A}_2$. For two nodes $v_1, v_2 \in \mathcal{V}$ satisfying $v_1 \leq v_2$ we first need to solve an instance of Directed Steiner Forest on k terminal pairs in order to know if we add the link v_1v_2 and, if so, at which cost. We construct an instance $\mathrm{I}(v_1, v_2)$ of Directed Steiner Forest on v_1 and v_2 as follows. The terminal pairs are $(x_i^1, x_i^2)_{1 \leq i \leq k}$. The underlying graph for $\mathrm{I}(v_1, v_2)$ is given by $D' = (V, A')$, where $A' = A - X_0 \cup \bigcup_{i \in [k]} \overleftarrow{Q_i}$, with $\overleftarrow{Q_i}$ being the arcs of Q_i in reversed direction. The costs of the arcs are given by

$$c'(e) := \begin{cases} c(e), & \text{if } e \in A \setminus X_0 \\ 0, & \text{if } e \in \overleftarrow{Q_i} \text{ for some } i \in [k]. \end{cases}$$

That is, we reverse all paths $Q_1, ..., Q_k$ such that x_i^2 is connected to x_i^1 for every $i \in [k]$ and make the corresponding arcs available at zero cost. We then need to connect x_i^1 to x_i^2 without using arcs of X_0. Since the number of terminal pairs is at most k and thus constant, $\mathrm{I}(v_1, v_2)$ can be solved in polynomial time by the algorithm of Feldman and Ruhl given in [FR06]. Let $\mathrm{OPT}(\mathrm{I}(v_1, v_2))$ be the cost of the optimal solution to $\mathrm{I}(v_1, v_2)$. We add a link v_1v_2 if the solution of $\mathrm{I}(v_1, v_2)$ is strongly connected. The set of such links is $\mathcal{A}_2$.

Finally we set $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$. For a link $e \in \mathcal{A}$ we set the cost of the link to

$$w(e) := \begin{cases} 0, & \text{if } e \in \mathcal{A}_1 \\ \mathrm{OPT}(\mathrm{I}(v_1, v_2)), & \text{if } e \in \mathcal{A}_2. \end{cases}$$

We now argue that a shortest path $\mathcal{P}$ from node $s_1 = (s, ..., s) \in \mathcal{V}$ to node $t_1 = (t, ..., t) \in \mathcal{V}$ in $\mathcal{D}$ corresponds to an optimal solution to I. For every link $vw \in \mathcal{P}$ we add the optimal solution to the Directed Steiner Forest instance $\mathrm{I}(v, w)$ to Y. The complete algorithm is given below.

Algorithm 2: : Exact algorithm for the augmentation problem

```
1: Construct the auxiliary graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$
2: Find a shortest path $\mathcal{P}$ in $\mathcal{D}$ from $s_1$ to $t_1$
3: For every link $vw \in \mathcal{P} \cap \mathcal{A}_2$ add the arcs of an optimal solution
   to $\mathrm{I}(v, w)$ to $Y$
4: return $Y$
```

We can now prove our main result in this section.

Theorem 4.23. *Algorithm 2 computes an optimal solution to an instance* I *of* Directed 1-robust k Disjoint Paths Augmentation.

Proof. Let $\mathcal{P}$ be a shortest path in the auxiliary graph $\mathcal{D}$ and let Y be the solution computed by Algorithm 2. We first prove the feasibility of Y.

Claim 1. *The solution Y computed by Algorithm 2 is feasible to* I.

Proof. Let $Y(v, w)$ be the optimal solution to $\mathrm{I}(v, w)$ for every link $e = vw \in \mathcal{P}$. We now argue that $X_0 \cup Y$ is feasible to I. By Lemma 4.22 we need to show that each unsafe arc of X_0 is contained in some strongly connected component in $D_{X_0}(X_0 \cup Y)$. Consider an arc $z_i^1 z_i^2 \in X_0$ on an arbitrary path P_i of X_0. Two nodes v_1 and v_2 containing z_i^1 and z_i^2, respectively, cannot be connected by a link of $\mathcal{A}_1$ in $\mathcal{D}$, as $z_i^1 z_i^2$ is an unsafe arc. Let $u_1 = (x_i^1, ..., x_k^1)$ ($u_2 = (x_i^2, ..., x_k^2)$) be the node on $\mathcal{P}$ such that x_i^1 (x_i^2) is closest to z_i^1 (z_i^2) on the subpath from s to z_i^1 (z_i^2 to t) of P_i. Those two nodes exist since $\mathcal{P}$ is a path from s_1 to t_1 in $\mathcal{D}$. If there is more than one such node let x_j^1 (x_j^2) be the vertex closest to t (s) on P_j for every $j \neq i$, $j \in [k]$. By the construction of u_1 and u_2 we also have that the link $u_1 u_2 \in \mathcal{P} \cap \mathcal{A}_2$. Therefore, the optimal solution $Y(u_1, u_2)$ to $\mathrm{I}(u_1, u_2)$ has been added to Y. Since $Y(u_1, u_2)$ connects z_i^1 and z_i^2 in $D_{X_0}(X_0 \cup Y)$, we obtain that the arc $z_i^1 z_i^2$ is contained in a strongly connected component in the residual graph $D_{X_0}(X_0 \cup Y)$. Feasibility now follows from Lemma 4.22. This proves Claim 1. □

Let Y^* be an optimal solution to I of cost OPT. We now show that Y is an optimal solution, i.e. we show that $c'(\mathcal{P}) \leq c(Y^*) = \mathrm{OPT}$. The idea is the following. We first show that the different strongly connected components in $D_{X_0}(X_0 \cup Y^*)$ are partially ordered. Using this ordering we can construct a path $\mathcal{P}'$ in $\mathcal{D}$ from s_1 to t_1 of cost $c'(\mathcal{P}') = c(Y^*)$. The theorem then follows by observing that a shortest path $\mathcal{P}$ has cost at most $c'(\mathcal{P}')$.

Let Z be some strongly connected component of $D_{X_0}(X_0 \cup Y^*)$ and let $L(Z) = \{i \in [k] \mid E(P_i) \cap E(Z) \neq \emptyset\}$ be the set of indices of the paths $P_1, ..., P_k$ that have at least one common edge with Z. Additionally for each $i \in L(Z)$ let $s_i(Z)$ be the vertex of Z that is closest to s on P_i and put $S(Z) = \bigcup_{i \in L(Z)} s_i(Z)$. Similarly, for each $i \in L(Z)$ let $t_i(Z)$ be the vertex of Z that is closest to t on P_i and put $T(Z) = \bigcup_{i \in L(Z)} t_i(Z)$.

Claim 2. *Let $e_i^1, e_i^2 \in P_i \cap F$ be two unsafe arcs of P_i such that their corresponding connected components Z_1 and Z_2 are disjoint. If e_i^1 is closer to s on P_i than e_i^2, then $t_i(Z_1) < s_i(Z_2)$ for every $i \in L(Z_1) \cap L(Z_2)$.*

Proof. Suppose this is not true and there is some $i \in L_1 \cap L_2$ such that $t_i(Z_1) \geq s_i(Z_2)$. Then, by the definition of $D_{X_0}(X_0 \cup Y^*)$, $t_i(Z_1)$ is connected to $s_i(Z_2)$ in $D_{X_0}(X_0 \cup Y^*)$. But as Z_1 and Z_2, respectively, are strongly connected components this implies that $Z_1 \cup Z_2$ is a single strongly connected component, a contradiction. This proves Claim 2. □

Using this claim we can construct a path $\mathcal{P}'$ in $\mathcal{D}$ of cost at most $c(Y^*)$.

Claim 3. *There is a path $\mathcal{P}'$ in $\mathcal{D}$ of cost at most $c(Y^*)$.*

Proof. We present an algorithm that constructs a path $\mathcal{P}'$ in $\mathcal{D}$ that only uses edges of $\mathcal{A}_2$ which correspond to strongly connected components of Y^* in $D_{X_0}(X_0 \cup Y^*)$. The path starts in $s_1 = (s, ..., s) \in \mathcal{V}$. We proceed by doing the following two steps alternately until we reach $t_1 = (t, ..., t) \in \mathcal{V}$.

1. From the current node $u = (u_1, ..., u_k)$ we proceed by taking links of $\mathcal{A}_1$ until we reach the next node $v = (v_1, ..., v_k) \in \mathcal{V}$ (with respect to the ordering of the nodes in $\mathcal{V}$) satisfying that each vertex $v_i, i \in [k]$ is either t or part of some strongly connected component in $D_{X_0}(X_0 \cup Y^*)$.

2. From v we take a link $vw \in \mathcal{A}_2$ to some node $w \in \mathcal{V}$, where the link vw corresponds to a strongly connected component Z of $D_{X_0}(X_0 \cup Y^*)$.

To show that the algorithm computes such a path $\mathcal{P}'$ we need to show that in Step 2 there is always a link $vw \in \mathcal{A}_2$ such that vw corresponds to a strongly connected component of $D_{X_0}(X_0 \cup Y^*)$. Let $v = (v_1, ..., v_k) \in \mathcal{V}$ be the node as constructed in Step 1 of the algorithm. Without loss of generality we can assume that $v_i \neq t$ for every $i \in [k]$. We now need to show that there is a strongly connected component Z in $D_{X_0}(X_0 \cup Y^*)$ such that $S(Z) \subseteq \{v_1, ..., v_k\}$. Suppose for each strongly connected component Z in $D_{X_0}(X_0 \cup Y^*)$ satisfying $S(Z) \cap \{v_1, ..., v_k\} \neq \emptyset$ we have that $S(Z) \not\subseteq \{v_1, ..., v_k\}$ and let $Z_1, ..., Z_j$ be those components. By Claim 2 we have that for every component $Z_i, i \in [j]$ there is a component $Z_{i'}, i' \in [j]$, such that there is some $l \in L_i \cap L_{i'}$ with $t_l(Z_{i'}) < s_l(Z_i)$. But then the ordering of the sets $Z_1, ..., Z_j$ induces a cycle, a contradiction to the ordering.

Hence the algorithm computes a path $\mathcal{P}'$ from s_0 to t_0. Consider any link $e = vw \in \mathcal{P}' \cap \mathcal{A}_2$ with $v = (v_1, ..., v_k)$ and $w = (w_1, ..., w_k)$ and let Z_e be its corresponding strongly connected component in Y^*. For the cost of the link e we now have that $c'(e) = \text{OPT}(\text{I}(v, w)) \leq c(Z_e)$ since $\text{I}(v, w)$ computes the optimal solution connecting $(v_i, w_i)_{1 \leq i \leq k}$ in the residual graph. Hence we have $c(\mathcal{P}') = \sum_{e \in \mathcal{P}' \cap \mathcal{A}_2} c'(e) \leq \sum_{e \in \mathcal{P}' \cap \mathcal{A}_2} c(Z_e) = \text{OPT}$. This proves Claim 3. $\square$

A shortest path $\mathcal{P}$ from s_0 to t_0 in $\mathcal{D}$ now satisfies $c'(\mathcal{P}) \leq c'(\mathcal{P}') \leq \text{OPT}$. This concludes the proof of Theorem 4.23. $\blacksquare$

We complement Theorem 4.23 by bounding the running time of Algorithm 2.

Theorem 4.24. *The running-time of Algorithm 2 can be bounded by* $O(|E| \cdot |V|^{6k-2} + |V|^{6k-1} \cdot \log |V|)$.

Proof. Let I be an instance of Directed 1-robust k Disjoint Paths Augmentation with $D = (V, A)$, where $|V| = n$ and $|A| = m$. Concerning the running time, the most involving part of the algorithm is to construct the auxiliary graph and to compute the shortest path from s_1 to t_1 in $\mathcal{D}$. For the auxiliary graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ we have that $|\mathcal{V}| \leq n^k$ and $|\mathcal{A}| \leq n^{2k}$. For each link $e \in \mathcal{A}$ we need to solve an instance of Directed Steiner Forest on at most k terminal pairs, which can be done in $O(mn^{4k-2} + n^{4k-1} \log n)$. Hence, $\mathcal{D}$ can be constructed in $O(mn^{6k-2} + n^{6k-1} \log n)$. Since $\mathcal{D}$ is a directed acyclic graph, a shortest path in $\mathcal{D}$ can be found in $O(|\mathcal{V}| + |\mathcal{A}|) = O(n^{2k})$. Thus the total running time is $O(mn^{6k-2} + n^{6k-1} \log n)$. $\blacksquare$

As a direct consequence of Theorem 4.23 we obtain a polynomial-time 2-approximation algorithm for Directed 1-robust k-Disjoint Paths. The algorithm is given below.

Algorithm 3: 2-approximation Algorithm for Directed 1-robust k-Disjoint Paths

```
1: Compute $k$ disjoint $s$-$t$-paths using a min-cost-flow algorithm to
   obtain $Y$
2: Construct the auxiliary graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ using the $k$ disjoint paths $Y$
3: Find a shortest path $\mathcal{P}$ in $\mathcal{D}$ from $s_1$ to $t_1$
4: For every link $vw \in \mathcal{P} \cap \mathcal{A}_2$ add the arcs of an optimal solution
   to $\mathrm{I}(v, w)$ to $Y$
5: **return** $Y$
```

Corollary 4.25. *Algorithm 3 is a 2-approximation to* Directed 1-robust k-Disjoint Paths.

Proof. Let OPT be the cost of an optimal solution to the instance I of Directed 1-robust k-Disjoint Paths. The shortest k disjoint paths X_0 from s to t in D have cost at most OPT. By Theorem 4.23 we can augment X_0 to a feasible solution $X_0 \cup Y$ to I at cost at most OPT. Thus we have that $c(X_0 \cup Y) = c(X_0) + c(Y) \leq 2 \cdot \text{OPT}$. ∎

4.3.4 Connection to Other Famous Open Problems

In the previous section we have seen that there is a polynomial-time exact algorithm for Directed 1-robust k Disjoint Paths Augmentation which leads in a straightforward way to a 2-approximation for Directed 1-robust k-Disjoint Paths. Ideally, one would like to complement such an approximation result with a hardness or inapproximability result. However, we did not find such a reduction.

All hardness results for Bulk-robust combinatorial optimization problems where obtained by showing that even the augmentation problem is NP-hard. Consider for example the reduction for Robust Matching Augmentation from Section 3. Therein, we have shown that for a given matching, even computing an optimal augmentation is as hard to approximate as Set Cover. To the best of our knowledge, this is not only true for the Robust Matching Problem, but for all Bulk-robust combinatorial optimization problems considered so far, i.e. for 2-robust Shortest Path, Robust Matching Augmenta-

TION or FLEXIBLE GRAPH CONNECTIVITY (which is, in a sense, the Robust Spanning Tree Problem, see Section 5).

However, this is not the case for DIRECTED 1-ROBUST k-DISJOINT PATHS, since we have shown that the augmentation problem, i.e. DIRECTED 1-ROBUST k DISJOINT PATHS AUGMENTATION, is polynomial-time solvable for constant k. Hence, there is some evidence that DIRECTED 1-ROBUST k-DISJOINT PATHS might actually be polynomial-time solvable. The problem in using Algorithm 2 for a polynomial-time algorithm for DIRECTED 1-ROBUST k-DISJOINT PATHS is the following. We need to somehow combine Algorithm 2, a dynamic programming approach, with an algorithm for finding k disjoint s-t-paths. But this is precisely the difficulty. Algorithms for finding k disjoint s-t-paths usually iteratively increase the number of disjoint paths until the number of disjoint paths reaches k. While increasing the number of these paths, we need to keep track of which disjoint paths we have already selected. To the contrary, a dynamic programming approach works completely the other way around. Therein, we precisely do not want to keep track of all the edges that we have selected so far since this would lead to an exponential running time. Combining these two kinds of algorithms would be a very nice result and could lead to a breakthrough for many related problems for which the complexity status is still unknown for a long time.

Even though we did not find a reduction to an NP-hard problem, we show that DIRECTED 1-ROBUST k-DISJOINT PATHS is closely related to two other problems for which the complexity status is unknown. Even though we do not give a reduction, we show that a special case of DIRECTED 1-ROBUST k-DISJOINT PATHS corresponds to 2-CONNECTED DIRECTED k STEINER TREE with one additional constraint.

Note that 2-CONNECTED DIRECTED k STEINER TREE is a generalization of DIRECTED STEINER TREE and therefore, by [HK03], does not admit a $\log^{2-\varepsilon}(n)$-factor approximation algorithm unless $\mathsf{NP} \subseteq \mathsf{ZTIME}(n^{\mathsf{polylog}(n)})$. However, there is a big gap in the complexity between DIRECTED STEINER TREE and 2-CONNECTED DIRECTED k STEINER TREE if the number k of terminals is bounded. While it is known that DIRECTED STEINER TREE is FPT in the number of terminals (and therefore polynomial-time solvable for constant k), it is still unknown if 2-CONNECTED DIRECTED k STEINER TREE can be solved in polynomial-time, even for $k = 2$ (note that for $k = 1$ an optimal solution is simply a min-cost-2-flow). We now show that a special case of DIRECTED 1-ROBUST k-DISJOINT PATHS corresponds to 2-CONNECTED DIRECTED k STEINER TREE with the additional constraint that any cut between s and the terminal vertices $t_1, \dots, t_k$ contains at least $k+1$ edges.

Proposition 4.26. *A polynomial-time algorithm for* DIRECTED 1-ROBUST k-DISJOINT PATHS *implies a polynomial-time algorithm for* 2-CONNECTED DIRECTED k STEINER TREE *with the additional constraint that every s-T cut contains at least $k+1$ edges.*

Proof. Consider an instance I of 2-CONNECTED DIRECTED k STEINER TREE with graph $G = (V, E)$, cost $c \in \mathbb{Q}^E$, root $s \in V$, and terminals $T = \{t_1, \dots, t_k\}$. We now construct an instance I$'$ of DIRECTED 1-ROBUST k-DISJOINT PATHS as follows. We add to G a vertex t and all edges between T and t, that is we add the edge set $E_t = \{(t_i, t) : t_i \in T\}$. The resulting graph is called G'. The cost of the new edges E_t are zero, while the edges in E have the same cost as before. Thus we have

$$c'(e) := \begin{cases} c(e), & \text{if } e \in E(G), \\ 0, & \text{otherwise.} \end{cases}$$

Finally we set $F := E$, that is, all original edges are unsafe, while the new edges E_t are safe.

Consider any feasible solution X to I$'$. Clearly we have $E_t \subseteq X$, since otherwise X is not feasible. We now show that the feasibility of X implies that there are 2 disjoint s-t_i paths in $G[X]$ for every $t_i \in T$. Assume this is not true and there are no two disjoint path between s and, say $t_k \in T$ in $G[X]$. By the max flow min cut theorem it follows that there is a cut edge $e \in E$ (or no edge at all) between s and t_k in $G[X]$. But then $Y := \{E_t \cup e \setminus (t_k, t)\}$ is a cut in $G'[x]$ of size k containing an unsafe edge. This contradicts the feasibility of X. Furthermore, note that every s-T cut in $G[X]$ contains at least $k+1$ edges since all edges in E are unsafe, as otherwise X is not feasible to I$'$.

Now observe that there is a one-to-one correspondence between feasible solutions to 2-CONNECTED DIRECTED k STEINER TREE on I with the property that every s-T cut contains at least $k+1$ edges and feasible solutions for

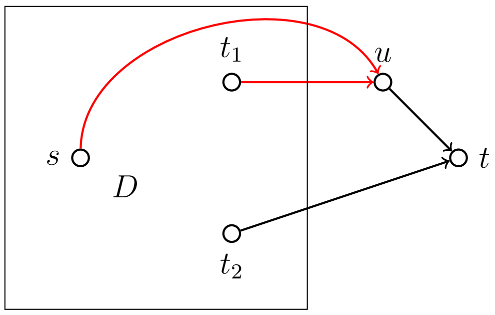


Figure 4.2: Illustration of the construction in Proposition 4.27. The rectangle indicates the graph D in which all edges are unsafe. The black edges are safe and the red edges are unsafe.

DIRECTED 1-ROBUST k-DISJOINT PATHS on I′. This completes the proof of Proposition 4.26. ∎

Furthermore, we show that 1-2-CONNECTED DIRECTED 2 STEINER TREE is a special case of DIRECTED 1-ROBUST 2-DISJOINT PATHS. Note that according to [CLNV14] even the complexity of 1-2-CONNECTED DIRECTED 2 STEINER TREE is still open.

Proposition 4.27. *A polynomial-time algorithm for* DIRECTED 1-ROBUST 2-DISJOINT PATHS *implies a polynomial-time algorithm for* 1-2-CONNECTED DIRECTED 2 STEINER TREE.

Proof. Let I be an instance of 1-2-CONNECTED DIRECTED 2 STEINER TREE with graph $G = (V, E)$, cost $c \in \mathbb{Q}^E$, root $s \in V$, and terminals $T = \{t_1, t_2\}$. Similar to the proof of Proposition 4.26 we construct an instance I′ of DIRECTED 1-ROBUST 2-DISJOINT PATHS as follows. We add to G two vertices u and t and four directed edges $\hat{E} = \{(s, u), (t_1, u), (u, t), (t_2, t)\}$. The resulting graph is called G'. The costs of the the new edges $\hat{E}$ are zero, while the edges in E have the same cost as before. Thus we have

$$c'(e) := \begin{cases} c(e), & \text{if } e \in E(G), \\ 0, & \text{otherwise.} \end{cases}$$

Finally, we set $F := E \cup \{(s, u), (t_1, u)\}$, that is, the edges incident to t are safe while all other edges are unsafe. An illustration of the construction is given in Figure 4.2.

Consider any feasible solution X to I'. Clearly we have $\hat{E} \subseteq X$, since otherwise X is not feasible. We now show that the feasibility of X implies that there is at least one s-t_i path and at least 2 disjoint s-t_2 paths in $G[X]$. Assume the first part is not true and that there is no path from s to t_1 in $G[X]$. But then $\{(s,u),(v,t)\}$ is a cut of size two in G', where (s,u) is an unsafe edge. This contradicts the feasibility of X.

Now assume that there are no two disjoint s-t_2 paths in $G[X]$. Similar to the proof of Proposition 4.26, we then have that X is not feasible.

Now observe that there is a one-to-one correspondence between feasible solutions to 1-2-connected Directed 2 Steiner Tree on I and feasible solutions to Directed 1-robust 2-Disjoint Paths on I'. This concludes the proof of Proposition 4.27. ■

According to [CLNV14], the undirected version of 1-2-connected Directed 2 Steiner Tree can be solved by a polynomial reduction to the following problem.

Problem 4.28 (Min-Cost Cycle Through 3 Vertices)**.** Given an undirected graph $D = (V, A)$, a cost function $c \in \mathbb{Q}^E$, and three vertices $u, v, w \in V$, the task is to find a minimum cost cycle $C \subseteq E$ containing u, v and w.

To the best of our knowledge, the complexity status of Min-Cost Cycle Through 3 Vertices is still unknown, even though Björklund, Husfeldt and Taslaman [BHT12] showed that there is a polynomial-time randomized algorithm that finds an optimal solution with one-sided errors of exponentially small probability in the number of vertices.

4.4 Conclusion

In this chapter we considered the robust disjoint s-t-paths problem and provided algorithms and hardness results. The main achievement is a polynomial-time exact algorithm for Directed 1-robust k Disjoint Paths Augmentation for constant k which immediately implies a 2-approximation for Directed 1-robust k-Disjoint Paths.

As already mentioned in Section 4.3.4, the main open problem of this chapter is whether Directed 1-robust k-Disjoint Paths admits a polynomial-time exact algorithm or not. On the one hand, all hardness results for Bulk-robust combinatorial optimization problems were obtained by showing that even the augmentation problem is NP-hard. This is not true for Directed 1-robust k-Disjoint Paths since we designed an efficient exact algorithm for this task. Therefore, there is some evidence that the problem is tractable. However, showing tractability of Directed 1-robust k-Disjoint Paths seems to be a difficult task, since it encapsulates several other problems for which the complexity status is unknown. As a first step one might look at the problem 1-2-connected Directed 2 Steiner Tree, which is a special case of Directed 1-robust 2-Disjoint Paths.

Chapter 5

Robust Spanning Trees

5.1 Introduction

The problem studied in this chapter, which we call Flexible Graph Connectivity (FGC), lies in the intersection of classical network design and robust optimization. It encapsulates several well studied problems that have received significant attention from the research community, such as the problems Minimum Spanning Tree (MST), 2-Edge Connected Spanning Subgraph (2-ECSS) [FJ81, Jai01, GGP$^+$94, SV14], Weighted Tree Augmentation (WTAP) [Adj18, FGKS18, FJ81, GKZ18, KN18, KN16, Nut17], and Matching Augmentation [CDG$^+$19]. As such, FGC is APX-hard and it encompasses all of the technical challenges associated with approximating these problems simultaneously. In a sense, MST and 2-ECSS represent two far ends of a spectrum of possible network design tasks that can be modeled with FGC and the other mentioned problems lie in between. We argue that by translating attributes of a real-life network design problem, one is much more likely to encounter a problem from the aforementioned spectrum, as opposed to one of its more famous extreme cases. The problem Flexible Graph Connectivity (FGC) is formally defined as follows.

Problem 5.1 (Flexible Graph Connectivity). We are given an undirected connected graph $G = (V, E)$, non-negative edge weights $w \in \mathbb{Q}^E_{\geq 0}$, and a set of edges $F \subseteq E$ called *unsafe* edges. Let $\overline{F} := E \setminus F$ be the *safe* edges.

The task is to compute a minimum-weight edge set $S \subseteq E$ with the property that $(V, S - f)$ is a connected spanning graph for every $f \in F$.

For an example of FGC together with 2-ECSS and WTAP please refer to Figure 5.1. We now briefly illustrate why MST, 2-ECSS, and WTAP are all special cases of FGC. Clearly, if all edges of the input graph are safe, then an optimal solution is a minimum-weight spanning tree. If, on the other hand, all edges of the input graph are unsafe, then an optimal solution is a minimum-weight 2-edge-connected spanning subgraph. Finally, if the unsafe edges form a weight-zero spanning tree T of the input graph, then an optimal solution is a minimum-cost tree augmentation of T. The goal of this chapter is to provide approximation algorithms for FGC and our main result is the following theorem.

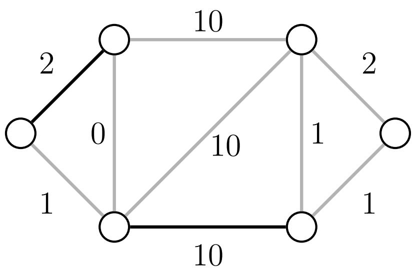
Theorem 5.2. Flexible Graph Connectivity *admits a polynomial-time 2.523-approximation algorithm.*

In the spirit of recent advancements for WTAP our results also extend to bounded-weight versions of the problem. A bounded-weight FGC instance is one whose weights all range between 1 and some fixed constant $M \in \mathbb{N}$. For bounded-weight FGC we obtain the following result.

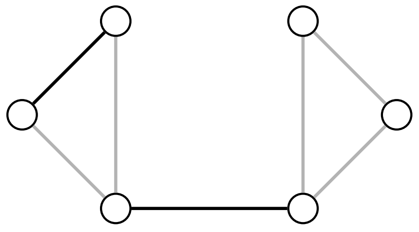
Theorem 5.3. Flexible Graph Connectivity *admits a polynomial-time 2.404-approximation algorithm on bounded-weight instances.*

Note that the algorithm stated in Theorem 5.3 has exponential running-time in the ratio $\frac{w_{\max}}{w_{\min}}$ between the maximum weight and minimum weight. We elaborate on the techniques used to prove these theorems as well as their connection to results on 2-ECSS and WTAP later on. We start by giving our motivation for studying this problem.

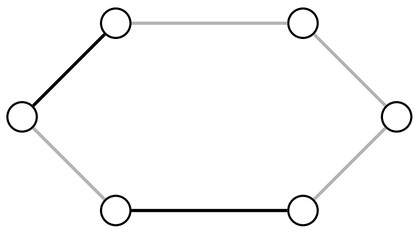
The relationship between FGC, 2-ECSS and WTAP. As was pointed out before, some classical and well studied network design problems are special cases of FGC, including 2-ECSS and WTAP. Thus, approximating FGC is at least as challenging as approximating the latter two problems. In this paragraph we present some evidence that FGC might actually be significantly harder. For the general case of both 2-ECSS and WTAP the iterative rounding algorithm of Jain [Jai01] provides an approximation factor of 2, which is best known. It is important to note that 2-ECSS subsumes WTAP in the



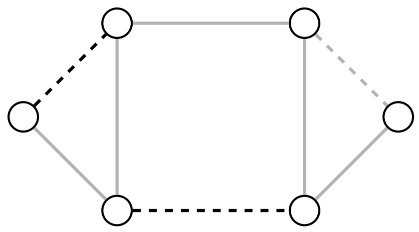
(a) FGC instance. Dark edges are safe and light edges are unsafe.



(b) Optimal Solution to FGC



(c) Optimal 2-edge-connected spanning subgraph



(d) Optimal MST Augmentation

Figure 5.1: The versatility of FGC. In (a) an instance of FGC with unsafe edges colored gray. In (b) the optimal solution. In (c) the optimal solution to the same instance, except that all edges are unsafe. The problem coincides with 2-ECSS on the same graph. In (d) the dashed edges are an optimal FGC solution for the instance where the unsafe edges have cost zero. It is equivalent to the tree augmentation problem for the gray minimum spanning tree.

general (weighted) case. Nevertheless, it is both instructive and useful to relate FGC to both problems. In particular, this enables us to improve the approximation ratio for the bounded-weight version of FGC.

In contrast, for the unweighted versions of both problems (and in the case of WTAP more generally for bounded weights) a long line of results has generated numerous improvements beyond ratio 2, leading to the currently best known bounds of 4/3 for unweighted 2-ECSS [SV14], 1.46 for unweighted tree augmenation [GKZ18] and 1.5 for WTAP with bounded weights [FGKS18, GKZ18]. The case of unit (or bounded) weights is where techniques for approximating 2-ECSS and WTAP start to differ significantly. In both cases, the best known bounds are achieved by combining LP-based techniques with clever combinatorial tools. Nevertheless, there seems to be very little intersection in both the nature of used LPs and the overall approaches, as techniques suitable for one problem do not seem to provide competitive ratios for the other.

Consequently, there are several implications for approximating FGC. First, achieving an approximation factor better than 2 is an ambitious task, as it would simultaneously improve the long-standing best known bounds for both 2-ECSS and WTAP. At the same time, for achieving a factor 2, it may be possible to use classical tools for survivable network design [Jai01]. We show that, at least with natural LPs, this is impossible, as the integrality gap of such LPs can be significantly larger than 2. Consider the following natural generalization of the cut-based formulation for survivable network design to FGC.

$$\begin{aligned} \text{minimize} \quad & w^T x \\ \text{subject to} \quad & \sum_{f \in \delta(S) \cap F} x_f + \sum_{e \in \delta(S) \cap \overline{F}} 2x_e \geq 2 \quad \text{for all } \emptyset \subsetneq S \subsetneq V \\ & x_e \in \{0,1\} \quad \text{for all } e \in E \end{aligned} \tag{5.1}$$

In essence, the IP formulation (5.1) states that each cut in the graph needs to contain at least one safe edge or at least two edges, which indeed is the feasibility condition for FGC. One can show that many important properties that are central in Jain's [Jai01] analysis still hold, e.g., the possibility to perform uncrossing for tight constraints at a vertex LP solution. These properties might become useful for devising a pure LP-based algorithm for FGC. However, for the instance shown in Figure 5.2, the integrality gap of formulation (5.1) is at least 8/3.

Finally, the recent advances achieved for unweighted and bounded-weight versions of 2-ECSS and WTAP seem to be unsuitable to directly tackle FGC, as they were not found to provide good ratios for both 2-ECSS and WTAP simultaneously. It is natural to conclude that a good ratio for FGC can only be achieved by a combination of techniques suitable for both 2-ECSS and WTAP.

To summarize, it seems that while the only known techniques for simultaneously approximating both 2-ECSS and WTAP within a factor two rely on rounding natural linear programming relaxations of a more general network de-

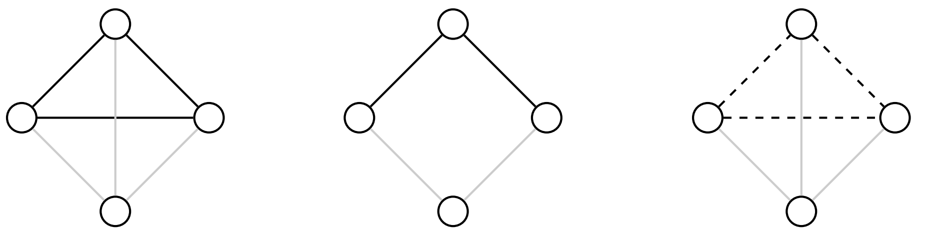

(a) FGC instance.
(b) Optimal integral solution of cost 2.
(c) Optimal fractional solution of cost 3/4.

Figure 5.2: FGC instance for which ILP (5.1) has integrality gap 8/3. Gray edges are unsafe and have cost 0. Black edges are safe and have cost 1. The dashed (safe) edges are fractional with value 1/4.

sign problem (such as the survivable network design problem [GGP$^+$94, Jai01]), the integrality gaps of such natural LPs for FGC are considerably larger than two. At the same time, due to its strong motivation, it is desirable to achieve a factor close to two, which is the state of the art for both 2-ECSS and WTAP.

In this chapter we show that this goal can be achieved by properly combining algorithms for 2-ECSS and WTAP. Our algorithms are simple and black-box to such an extent that results for restricted versions of WTAP (e.g., bounded cost) can be directly applied to FGC with the same restrictions, thus leading to improved bounds for these restricted versions of FGC as well. At the same time, the analysis is complex and requires careful charging arguments, generalizations of the notion of exchange bijections of spanning trees, and factor revealing optimization problems, as we elaborate next.

Summary of Algorithmic Techniques. We present here a high-level overview of some of the technical ingredients that go into our algorithm and analysis used to prove Theorem 5.2. The algorithm carefully combines the following three rather simple algorithms for FGC, each having an approximation ratio significantly worse than the one exhibited in Theorem 5.2. A more detailed description of the algorithms will be given later on. Each of the algorithms establishes 2-edge-connectivity in a modified graph, where a subset of safe edges is contracted. In order to be able to establish 2-edge connectivity, we need to add a parallel unsafe edge e' for each safe edge e of the same cost. It can easily be observed that optimal solutions for the new instance are also optimal solutions for the old instance and vice versa. The three algorithms are stated below.

Algorithm A	Compute a 2-edge-connected spanning subgraph.
Algorithm B	Compute a minimum spanning tree, contract its safe edges and make it 2-edge-connected by solving the corresponding WTAP instance.
Algorithm C	Compute a minimum spanning tree, contract its safe edges, then compute a 2-edge-connected spanning subgraph on the resulting graph. Return the union of this solution and the safe edges of the spanning tree.

It is not hard to show that Algorithms A, B, and C are polynomial-time approximation algorithms for FGC, with approximation ratios of $4, 3$, and 5, respectively, given that 2-ECSS and WTAP admit polynomial-time 2-approximation algorithms. We defer the details to the later sections and instead present a road map for proving the main result.

Our approximate solution is obtained from returning the best of many solutions, each computed by one of the above three algorithms on an instance that is computed from the original instance by appropriately scaling the costs of the safe edges. The motivation for making safe edges cheaper is that buying a similarly priced unsafe edge instead likely incurs extra costs, since one safe edge or at least two edges have to cross each cut. The technical challenge is to determine the most useful scaling factors.

The main idea in the analysis is to relate the costs of edges in an optimal solution to the costs of edges in the computed solutions based on a proper generalization of *exchange bijections* between spanning trees. Exchange bijections $\varphi : A \to B$ are bijections between bases A, B of a matroid (e.g., spanning trees in a connected graph), such that for any $a \in A$, the set $A \setminus \{a\} \cup \{\varphi(a)\}$ is again a basis of the matroid. It is well known that an exchange bijection always exists. We introduce our generalized notion of *α-monotone exchange bijections*, where α is a scaling factor used in the algorithm, and prove that they always exist between a spanning tree of the optimal and a spanning tree of a computed solution. We then combine the properties of such bijections with additional technical ideas to derive an upper bound on the cost of the computed solution. The bound is expressed in terms of several parameters

problem	general weights	bounded weights	unweighted
FGC	2.523 (Thm. 5.2)	2.404 (Thm. 5.3)	3/2 (Thm. 5.4)
2-ECSS	2 [Jai01]	2 [Jai01]	4/3 [SV14]
WTAP	2 [FJ81]	3/2 [GKZ18]	1.46 [GKZ18]

Table 5.1: Approximation ratios for FGC and some special cases.

that represent proportions of costs associated with parts of the computed and an unknown optimal solution, defined through the exchange bijections.

The final step is to combine all obtained upper bounds. Since we have the choice of selecting the scaling factors, but have no control over the parameters appearing in the upper bounds, we can compute a conservative upper bound on the approximation ratio by solving a three-stage *factor-revealing min-max-min optimization problem.* The inner minimum is taken over the upper bounds on the values of the solutions computed by Algorithms A, B, and C. The maximum is taken over the parameters that depend on an unknown optimal solution. Finally, the outer minimum is taken over the choice of scaling factors. One interesting aspect of our factor-revealing optimization problem is that its solution gives not only a bound on the approximation ratio of the algorithm (as in, e.g., [JMM$^+$03, ABWZ14]), but it also suggests optimal instance-independent scaling factors to be used by the algorithm itself. One can show that by computing the scaling factors it is possible to further constrain the unknown parameters in the problem, thus obtaining better instance-specific bounds. While we can compute these factors in polynomial time, we will not elaborate on this approach in this work.

Since the overall factor-revealing optimization problem is a three-stage min-max-min program, we can only provide analytic proof for its optimal value for very small sizes. However, we are still able to give an analytic bound of 2.523 in order to prove Theorem 5.2 by combining only Algorithms A and C, but using the optimal choices of scaling factors for a given instance. To achieve the factor 2.404 for bounded-weight instances we use all three Algorithms A, B, and C to bound the optimal solution of the min-max-min problem. Clearly, using all three algorithms yields better bounds, but we cannot give an analytic upper bound. Instead we give a computational upper bound on the min-max-min problem using the BARON solver [The19], which is a global optimization software and is based on branch-and-bound techniques to compute optimal solutions. An overview of our approximation guarantees along with some related results can be found in Table 5.1.

Further Results. We also consider several generalizations and special cases of FGC. First, we show that the unweighted version of the problem admits a 3/2-approximation algorithm. We note that Unweighted FGC does not contain unweighted WTAP as special case, and hence, our result does not imply a 3/2-approximation algorithm for unweighted WTAP. In particular, we prove the following theorem for a generalization of FGC, which we call k-FGC. The problem k-FGC asks for the minimum weight connected subgraph of a given graph that can withstand the failure of at most k unsafe edges. Note that 1-FGC is simply FGC.

Theorem 5.4. Unweighted k-Flexible Graph Connectivity *admits a polynomial-time* $\left(\vartheta_{k+1} \cdot \frac{2k+1}{2k+2} + \frac{1}{k+1}\right)$*-approximation algorithm.*

By ϑ_{k+1} we denote the approximation ratio of an approximation algorithm for the minimum $(k+1)$-edge-connected spanning subgraph problem. In particular this implies a polynomial-time 3/2-approximation algorithm for Unweighted FGC. Note that the approximation guarantee in Theorem 5.4 tends to 1 as k tends to infinity. We prove this result in Section 5.2.

We contrast our algorithmic results by showing in Section 5.4 that a natural generalization of FGC to matroid optimization is NP-hard to approximate within any sublogarithmic factor.

Notation. We denote by ϑ_k the ratio of an approximation algorithm for the problem of finding a minimum-cardinality k-edge-connected spanning subgraph. Similarly, we denote by $c_{2\text{EC}}$ (c_{TAP}, resp.) the ratio of an approximation algorithm for 2-ECSS (WTAP, resp.).

For the remainder of this chapter we consider an instance $\mathrm{I} = (G, w, \overline{F})$ of FGC and some optimal solution $Z^* \subseteq E(G)$ of I. To avoid technicalities, we add to each safe edge a parallel unsafe edge of the same cost. It is readily seen that this modification preserves optimal solutions (a solution that uses both edges of a parallel pair is not optimal and we may always just pick the safe edge in an optimal solution). Observe that the solution Z^* has the following structure: for some $r \in \mathbb{N}$, the graph $(V(G), Z^*)$ consists of r 2-edge-connected components $C_1, C_2, \ldots, C_r$ that are joined together by safe edges $E' := \{\overline{f}_1, \overline{f}_2, \ldots, \overline{f}_{r-1}\} \subseteq \overline{F}$ in a tree-like fashion. That is, if we contract each component C_i to a single vertex, the remaining graph is a tree T^* with edge set E'. On the other hand, if we contract E' , we obtain a 2-edge-connected spanning subgraph of the resulting graph. We let $\delta := w(E')/\text{OPT(I)}$. That is, the value δ is the proportion of the cost of the safe cut edges E' relative to the total cost of the optimal solution Z^*.

Organization of the Chapter. In the next section we consider the unweighted case and prove the result for Unweighted k-FGC. Section 5.3 deals with the weighted case. Therein, we first give in Section 5.3.1 the three basic algorithms mentioned in the introduction followed by the overall algorithm (Algorithm 4) for FGC in Section 5.3.2. In Section 5.3.3 we present the technical tools to bound the cost of Algorithm 4. Section 5.3.4 contains the rather easy proof that Algorithm 4 is a 2.8-approximation. In Sections 5.3.5 and 5.3.6 we prove Theorem 5.2 and Theorem 5.3. To complement the approximation result, we give an $o(\log n)$ inapproximability result in Section 5.4. Finally, Section 5.5 concludes the chapter.

5.2 Unweighted k-Flexible Graph Connectivity

In this section we give an approximation algorithm for Unweighted k-FGC, which asks for a minimum-cardinality subgraph of a given graph, such that the removal of any k *unsafe* edges results in a connected graph. Note that 1-FGC is simply FGC.

Recall that we denote by ϑ_k the approximation ratio of a polynomial-time algorithm for the problem Unweighted k-ECSS. Currently, the best known values for ϑ_k are 4/3 for $k = 2$ due to a result of Sebő and Vygen [SV14], $1 + 2/(k + 1)$ for $3 \leq k \leq 6$ due to Cheriyan and Thurimella [CT00], and $1 + 1/2k + O(1/k^2)$ for $k \geq 7$ due to Gabow and Gallagher [GG12]. Note that if k tends to infinity then ϑ_k tends to 1. In the remainder of this section we prove the following theorem.

Theorem 5.4. Unweighted k-Flexible Graph Connectivity *admits a polynomial-time* $\left(\vartheta_{k+1} \cdot \frac{2k+1}{2k+2} + \frac{1}{k+1}\right)$*-approximation algorithm.*

Note that with the current best value for ϑ_2 from [SV14], Theorem 5.4 gives a 3/2-approximation guarantee for Unweighted FGC.

The approximation algorithm proceeds as follows. First, we compute a maximum forest X on the safe edges $\overline{F}$. We then compute a ϑ_{k+1}-approximate $(k+1)$-edge-connected spanning subgraph Y of G/X and output $X \cup Y$.

Lemma 5.5. *Let* $H \subseteq G$ *be a feasible solution to* I*. Then* $H/(\overline{F} \cap E(H))$ *is* $(k+1)$*-edge-connected.*

Proof. Suppose for a contradiction that $G' := H/(\overline{F} \cap E(H))$ is at most k-edge-connected. That is, there are two vertices $v, w \in V(G')$ that are connected by at most k edge-disjoint paths. Therefore, there is a cut $F' \subseteq E(G')$ of size at most k separating v and w. But then F' is also a cut of size at most k in H and F' consists only of unsafe edges. Therefore, H is not feasible, a contradiction. ■

We are now ready to prove Theorem 5.4.

Proof of Theorem 5.4. Let $H = X \cup Y$ be a solution computed by the algorithm described above. Suppose that a maximum forest in $G - F$ has size $\ell = |E(X)|$. Let Y^* be a minimum $(k+1)$-edge-connected spanning subgraph of $G/E(X)$. We may compute in polynomial-time a $(k+1)$-ECSS of $G/E(X)$ of size $\vartheta_{k+1} \cdot |E(Y^*)|$. Therefore, the solution $X \cup Y$ output by the algorithm has size

$$|E(X)| + |E(Y)| \leq \ell + \vartheta_{k+1} \cdot |E(Y^*)| \leq \ell + \vartheta_{k+1} \cdot \mathrm{OPT(I)}\ .$$

On the other hand, we have that

$$|E(X)| + |E(Y)| \leq \ell + 2(k+1)(n-\ell) \leq 2\mathrm{OPT(I)} - (2k+1)\ell\ .$$

Hence,

$$\begin{aligned} |X| + |Y| &\leq \min\{\ell + \vartheta_{k+1} \cdot \mathrm{OPT(I)},\ 2\mathrm{OPT(I)} - (2k+1)\ell\} \\ &\leq \frac{2 + \vartheta_{k+1}(2k+1)}{2k+2} \cdot \mathrm{OPT(I)} \\ &= \left(\vartheta_{k+1} \cdot \frac{2k+1}{2k+2} + \frac{1}{k+1}\right) \cdot \mathrm{OPT(I)}\ , \end{aligned}$$

as claimed. This concludes the proof of Theorem 5.4. ■

5.3 Weighted Flexible Graph Connectivity

In this section we present our two main results: a polynomial-time 2.523-approximation algorithm for FGC (Theorem 5.2) and a polynomial-time 2.404-approximation algorithm for bounded-weight FGC (Theorem 5.3).

As pointed out in the introduction of this chapter, we first state three simple algorithms which turn out to perform poorly on their own. Then, we show how we can improve on these simple algorithms by applying them to a large amount of modified instances, where we scale the cost of some of the edges. The algorithm then simply outputs the best among all these solutions. After introducing some useful technical properties used throughout this section, we then show how we can bound the cost of each solution computed by the algorithm. Finally, we combine all these bounds and obtain the proposed approximation factors.

5.3.1 Three Simple Approximation Algorithms for FGC

We introduce three basic approximation algorithms for FGC that use in a black-box fashion approximation algorithms for 2-ECSS and WTAP. With the best known algorithms for these problems, it is readily seen that the first has an approximation guarantee of $(2+2\delta)$ and the second has a guarantee of 3. Using our technical tools from Section 5.3.3 one can show that the third algorithm is a 5-approximation algorithm for FGC. Interestingly, it performs much better precisely when the other two algorithms exhibit their worst-case behavior. Note that none of the three algorithms attains the integrality gap of 8/3 exhibited by the instance shown in Figure 5.2.

Algorithm A: a $(2+2\delta)$-approximation algorithm. Algorithm A computes (in polynomial time) a $c_{2\mathrm{EC}}$-approximate 2-edge-connected spanning subgraph, e.g., by Jain's algorithm [Jai01]. Algorithm A then removes all unsafe edges that are parallel to safe edges of the spanning subgraph and returns the resulting edge set. Observe that the returned solution is feasible. We now argue that Algorithm A is a $(2+2\delta)$-approximation algorithm. By adding a copy of each edge in E' to the optimal solution Z^* we obtain a 2-edge-connected spanning subgraph H_1^* of G. The cost $w(H_1^*)$ of H_1^* is given by

$$w(H_1^*) = \sum_{e \in Z^* \setminus E'} w(e) + 2 \cdot \sum_{e \in E(Z^*) \cap E'} w(e) \ .$$

Since $w(H_1) \leq 2w(H_1^*)$, it follows that

$$w(H_1) \leq 2w(H_1^*) = 2(1-\delta) \cdot \mathrm{OPT(I)} + 4\delta \mathrm{OPT(I)} = (2+2\delta) \cdot \mathrm{OPT(I)}$$

as claimed. Observe that Algorithm A performs best if the weight of the safe edges E' is small.

Algorithm B: a 3-approximation algorithm. Algorithm B computes a minimum spanning tree T of G and then computes a c_{TAP}-approximate solution to the WTAP instance (G, T', w), where the tree T' is obtained by contracting each safe edge of T. The solution of the WTAP instance together with the tree T is a feasible solution for I by Lemma 5.5. Since $w(T) \leq \mathrm{OPT(I)}$, the best available algorithms for WTAP (see Table 5.1) give a 3-approximation for FGC, and a 5/2-approximation for FGC on bounded-weight instances.

Algorithm C: a 5-approximation algorithm. Algorithm C first computes a minimum spanning tree T of G. Let G' be the graph obtained from G by contracting the safe edges of T, that is, $G' := G/(E(T) \cap \overline{F})$. Algorithm C then computes a $c_{2\mathrm{EC}}$-approximate 2-edge-connected spanning subgraph $H' \subseteq E(G')$ and returns the edge set $(E(T) \cap \overline{F}) \cup H'$. It is readily seen that Algorithm C computes a feasible solution and that the safe edges of T have cost at most $\mathrm{OPT(I)}$. Using exchange bijections from Section 5.3.3, it can be shown that $w(H') \leq 4\mathrm{OPT(I)}$, so Algorithm C is a 5-approximation algorithm for FGC.

5.3.2 An Improved Approximation Algorithm

In this section we describe our improved approximation algorithm for FGC, which hybridizes the three basic algorithms for FGC discussed in Section 5.3.1. We first illustrate why Algorithm B, which has the best approximation guarantee of the three simple algorithms, may perform poorly. Recall that Algorithm B first computes a minimum spanning tree of G and then approximates

Algorithm 4: Improved Approximation Algorithm for FGC

```
input : Instance $\mathrm{I} = (G, w, \overline{F})$ of FGC
compute threshold values $W := \{\alpha_e \mid e \in E(G)\} \cup \{0, 1\}$
run Algorithm A on I to obtain solution $Z^A$
for threshold value $\alpha \in W$ do
    run Algorithms B and C with scaling factor $\alpha$ to obtain solutions
      $Z^B_\alpha$ and $Z^C_\alpha$, respectively
return solution with lowest cost among $Z^A$ and $\{Z^B_\alpha, Z^C_\alpha \mid \alpha \in W\}$
```

the WTAP instance (G, T', w), where the tree T' is obtained from the minimum spanning tree by contracting its safe edges. Consider a cut edge $e \in \overline{F}$ of an optimal solution to I. If Algorithm B chooses instead of e a slightly cheaper unsafe edge f across the same cut in the MST computation, then to make this cut "safe" in the second step we need to buy either the edge e or another edge of similar cost across the cut. However, we can only c_{TAP}-approximate this step. Hence, if Algorithm B chooses f instead of e, it may result in cost of up to $3w(e)$. We try to avoid such a situation by scaling the weight of all safe edges by a suitable factor $\alpha \in [0, 1]$, hence making safe edges more attractive.

Algorithm 4, our improved approximation algorithm for FGC, proceeds as follows. It first computes suitable scaling factors $W \subseteq [0, 1]$ (called "threshold values") for the costs of the safe edges. Then, Algorithm 4 runs Algorithm A using the original weights w to obtain solution Z^A. We say that we run Algorithm B (resp., C) *with scaling factor* α if the minimum spanning tree in Algorithm B (resp., C) is computed with respect to weights obtained from w by scaling the costs of the safe edges by α. Algorithm 4 runs Algorithms B and C with each scaling factor $\alpha \in W \cup \{0, 1\}$ and returns a solution of minimal weight among all the different solutions computed by Algorithms A, B, and C.

From the discussion in Section 5.3.1 it follows that Algorithm 4 returns a feasible solution. As we will later see (and is pointed out in Algorithm 4) we use only $|E|$ many threshold values. Since we can compute the threshold values in polynomial-time it follows that Algorithm 4 runs in polynomial-time. We defer the proofs of the running-time to Section 5.3.3.

Let us denote by $\mathcal{A}(\mathrm{I})$ the weight of the solution returned by Algorithm 4.

Using properties of the threshold values we show that the selection of the scaling factors in Algorithm 4 is best possible.

Lemma 5.6. *Let $W' \subseteq [0,1]$ and let $\mathcal{A}'(\mathrm{I})$ be the weight of the solution returned by Algorithm 4 run with W set to W'. Then $\mathcal{A}(\mathrm{I}) \leq \mathcal{A}'(\mathrm{I})$.*

The proof of Lemma 5.6 will be given in Section 5.3.3. A detailed analysis of the approximation ratio of Algorithm 4 is deferred to Section 5.3.5. Here, we give a high-level overview. Our starting point is Lemma 5.6, which allows us to assume that Algorithm 4 tries *all* scaling factors in $[0,1]$. We show that the approximation ratio of Algorithm 4 is bounded from above by the optimal value of a min-max-min optimization problem. For an instance I of FGC and some $N \in \mathbb{N}$, the optimization problem has the following data.

- Scaling factors $\alpha_1, \alpha_2, \ldots, \alpha_N \in [0,1]$. Due to the discussion above, in our analysis of Algorithm 4 we are free to choose these values.
- Parameters $\beta_1, \beta_2, \ldots, \beta_N, \gamma_1, \gamma_2, \ldots, \gamma_N \in [0,1]$, which depend on the structure of an optimal solution. These parameters additionally satisfy $\sum_{j=1}^N \beta_j + \sum_{j=1}^N \gamma_j = 1$.
- Functions f^A, as well as $f_1^B, f_2^B, \ldots, f_N^B$ and $f_1^C, f_2^C, \ldots, f_N^C$ that bound from above in terms of α_i, β_i, γ_i, $1 \leq i \leq N$, c_{2EC} and c_{TAP}, the cost of the solutions.

Precise definitions of the parameters and the functions will be given in Section 5.3.5. We note that Algorithm B is only used in the proof of Theorem 5.3. Technically, we show that for a proper choice of functions f^A, f_i^B, and f_i^C the approximation guarantee of Algorithm 4 is bounded by the optimal value of the following optimization problem.

$$\begin{aligned} \min_{\alpha_i \in [0,1]\,:\,1 \leq i \leq N} \; \max_{\substack{\beta_i \in [0,1]\,:\,1 \leq i \leq N \\ \gamma_i \in [0,1]\,:\,1 \leq i \leq N}} \; \min_{1 \leq i \leq N} \quad & \{f^A(\cdot),\, f_i^B(\cdot),\, f_i^C(\cdot)\} \\ \text{subject to} \quad & \sum_{j=1}^N \beta_j + \sum_{j=1}^N \gamma_j = 1 \end{aligned} \tag{5.2}$$

Our goal is to provide suitable functions f^A, f_i^B, and f_i^C for our analysis. In Section 5.3.5 we will give an analytic upper bound of 2.523 on the optimal value of (5.2).

5.3.3 α-MSTs, Thresholds, and Exchange Bijections

In this section we present our main technical tools that are needed for the analysis of Algorithm 4.

We first show that safe edges and unsafe edges exhibit a "threshold" behavior with respect to MSTs if the costs are scaled by some $\alpha \in [0,1]$. Furthermore, we show that i) the used threshold values can be computed in polynomial time, which is essential to ensure that Algorithm 4 runs in polynomial time, and ii) these are the best choice of scaling factors for Algorithm 4, which allows us to assume in our analysis that we execute Algorithm 4 for *all* scaling factors $\alpha \in [0,1]$. For $\alpha \in [0,1]$, we denote by

$$w_\alpha(e) = \begin{cases} \alpha \cdot w(e), & \text{if } e \in \overline{F}, \text{ and} \\ w(e), & \text{otherwise} \end{cases}$$

the weight function obtained from w by scaling the costs of the safe edges by α. A spanning tree T is called *α-minimum spanning tree* (α-MST) if $E(T)$ has minimal weight with respect to w_α.

Consider changing the scaling factor α smoothly from 0 to 1. We observe that for any safe edge e, if there is an α-MST containing e, then there is also an α'-MST containing e for any $\alpha' \leq \alpha \leq 1$. On the other hand, if there is an α-MST containing an unsafe edge f then there is also an α'-MST containing f for any $\alpha \leq \alpha' \leq 1$. We formally capture this notion in the following definition.

Definition 5.7. Let $e \in E$ and $\alpha_e \in [0,1]$. We say that α_e is a *lower threshold for* e if for any $\alpha \in [0,1]$ there is an α-MST containing e if and only if $\alpha \geq \alpha_e$. If e is in no α-MST for $0 \leq \alpha \leq 1$, we define the lower threshold value of e to be ∞. Similarly, α_e is an *upper threshold for* e if for $\alpha \in [0,1]$ there is an α-MST containing e if and only if $\alpha \leq \alpha_e$. The threshold values of an instance $\mathrm{I} = (G, w, \overline{F})$ are the collection of all threshold values for all edges, i.e. $\{\alpha_e \mid e \in E(G)\}$.

The following technical lemma ensures the existence of threshold values for safe and unsafe edges.

Lemma 5.8. *For each unsafe edge $f \in F$ there is a lower threshold $\alpha_f \in [0,1] \cup \{\infty\}$. For each safe edge $e \in \overline{F}$ there is an upper threshold $\alpha_e \in [0,1]$.*

Proof. We first prove the following claim.

Claim 1. *Let $f \in F$ be an unsafe edge. If f is in some α-MST then for any $\alpha' \geq \alpha$, there is some α'-MST containing f.*

Proof. Let $\alpha' \geq \alpha$ and let us fix some α'-MST $T_{\alpha'}$. Suppose for a contradiction that f is in T_α but not in $T_{\alpha'}$ and assume that f has smallest weight among all such edges. Consider the edges $e_1, e_2, \ldots, e_m$ of G ordered non-decreasingly by their weight w_α and suppose $f = e_i$. Similarly, let $e'_1, e'_2, \ldots, e'_m$ be the edges of G ordered non-decreasingly by $w_{\alpha'}$ and suppose that $f = e'_{i'}$. Note that due to the construction of the weight function, the weights of all edges in $\overline{F}$ are scaled by the same factor α and the weights of the edges in F are the same for w_α and $w_{\alpha'}$. Therefore, we have that $i' \leq i$ and that $\{e'_1, \ldots, e'_{i'}\} \subseteq \{e_1, \ldots, e_i\}$.

For $1 \leq i \leq m$, let T^i_α ($T^i_{\alpha'}$, resp.) be the restriction of T_α ($T_{\alpha'}$, resp.) to $e_1, e_2, \ldots, e_i$ ($e'_1, e'_2, \ldots, e'_i$, resp.). Since $f = e_{i'}$ is not in $T^{i'}_{\alpha'}$, the graph $T^{i'}_{\alpha'} + f$ contains a unique cycle C_f. Let $S \coloneqq E(C_f) \setminus E(T^i_\alpha)$. The set S is non-empty since $f = e_i$ is in T^i_α. For each $e \in S$, the graph $T^i_\alpha + e$ contains a unique cycle C_e. Hence, the edge set

$$C' \coloneqq (E(C_f) \setminus S) \cup \bigcup_{e \in S} E(C_e) - e$$

contains a cycle, but $C' \subseteq E(T^i_\alpha)$, which contradicts our assumption that T_α is a tree. This proves Claim 1. □

Let $f \in F$ be an unsafe edge. If f is not contained in some 1-MST, then it is not contained in any α-MST for $0 \leq \alpha \leq 1$. Therefore, the edge f has a lower threshold value $\alpha_f = \infty$. Consider the case that f is contained in some α-MST, where $0 \leq \alpha \leq 1$. We choose α_f to be the smallest value $\alpha \in [0, 1]$, such that there is an α-MST containing f. By Claim 1, we have that α_f is a lower threshold value for f.

We now prove the existence of an upper threshold value for safe edges. The proof of the following claim is analogous to the proof of Claim 1.

Claim 2. *Let $e \in \overline{F}$ be a safe edge. If e is an edge of some α-MST then for any $\alpha' \leq \alpha$, there is some α'-MST containing e.*

Let $e \in \overline{F}$ be a safe edge. Observe that for $\alpha = 0$, there is an α-MST containing e. We let α_e be the largest value of $\alpha \in [0, 1]$, such that there is an α-MST containing f. By Claim 2, we have that α_e is an upper threshold value for e. This concludes the proof of Lemma 5.8. ■

It is easily seen that there are $O(|V(G)|^2)$ threshold values. This implies in particular that Algorithm 4 runs in polynomial time. In fact, according to the next proposition there are at most $|V(G)| - 1$ different threshold values.

Proposition 5.9. *For each safe edge $e \in \overline{F}$ (unsafe edge $f \in F$, resp.), the upper threshold α_e (lower threshold α_f, resp.) can be computed in polynomial time. Furthermore, there are at most $|V(G)| - 1$ threshold values.*

Proof. Let T_1 be a 1-MST of G and let $F_1 \coloneqq \{f_1, f_2, \ldots, f_\ell\} = F \cap E(T_1)$ be the unsafe edges of T_1. For each $e \in \overline{F}$ and $f_i \in F_1$, we initialize $\alpha_i^e \coloneqq w(f_i)/w(e)$. Thus, for $\alpha = \alpha_i^e$ we have $w_\alpha(e) = w_\alpha(f_i)$. We keep a set W of all such α_i^e (actually triples (e, i, α_i^e)). Since $|\overline{F}| \leq |E|$ and $|E(T_1)| < |V|$, we have that W has cardinality at most $|E| \cdot |V|$. Additionally, we define a mapping $\varrho : F_1 \to \overline{F} \setminus E(T_1)$. Whenever for some $\alpha \in [0, 1]$, an edge $f \in F_1$ is replaced in some α-MST by a safe edge $\overline{f} \in \overline{F} \setminus E(T_1)$, we set $\varrho(f) \coloneqq \overline{f}$.

We perform at most $|V| \cdot |E|$ iterations. In iteration j we do the following. We pick some $\alpha_i^e \in W$ of largest value and check whether the safe edge e is in some α_i^e-MST $T_{\alpha_i^e}$. If this is not the case, we remove α_i^e from W and continue with the next iteration. Otherwise, we set $\varrho(f_i) \coloneqq e$ and $\alpha_{f_i} \coloneqq \alpha_i^e$ and for each $1 \leq i \leq \ell$ we delete α_i^e from W and for each $e' \in \overline{F} \setminus E(T_1)$ we delete $\alpha_i^{e'}$ from W. Note that we can distinguish between the two cases in polynomial time by computing minimum-weight spanning trees.

Observe that after the above algorithm terminates, the mapping ϱ assigns to each safe edge $e \in \overline{F} \setminus E(T_1)$ at most one partner $f \in F_1$, and to each unsafe edge $f_i \in F_1$ at most one partner $e' \in \overline{F} \setminus E(T_1)$. For each $e \in \overline{F} \cap E(T_1)$, we let $\alpha_e \coloneqq 1$ and for each $f \in F \setminus E(T_1)$, we let $\alpha_f \coloneqq \infty$. Now, for each $f_i \in F_1 = F \cap E(T_1)$ with $\varrho(f_i) = e$, we let $\alpha_e = \alpha_{f_i} \coloneqq \alpha_i^e$. Finally, for each $e \in \overline{F} \setminus T_1$ that is not in the image of ϱ (for each $f \in F \cap E(T_1)$ that is not in the domain of ϱ, resp.), we let $\alpha_e \coloneqq 0$ ($\alpha_f \coloneqq 0$, resp.). It is readily verified that these choices are in accordance with the definition of lower and upper threshold values (Definition 5.7). Finally, since there are at most $|V(G)| - 1$ many unsafe edges in T_1, there are at most $|V(G)| - 1$ many threshold values. ■

We now prove that threshold values are optimal scaling factors for Algorithm 4, as claimed in Lemma 5.6.

Proof of Lemma 5.6. Let $\alpha \in W'$ and let α_L (α_R, resp.) be the largest (resp., smallest) item in W, such that $\alpha_L \leq \alpha$ ($\alpha_R \geq \alpha$, resp.). Then, since W contains a threshold value for each edge of G and by the properties of the threshold values given in Definition 5.7, the tree T_α is either an α_L-MST or an α_R-MST of G. Therefore, Algorithm 4 has computed an α-MST and a corresponding augmentation for each $\alpha \in W'$. We conclude that $\mathcal{A}(\mathrm{I}) \leq \mathcal{A}'(\mathrm{I})$, proving Lemma 5.6. ■

In our analysis of Algorithm 4 we will use a charging argument based on the notion of monotone exchange bijections, which we now introduce. Let G be a connected graph and let T and T' be spanning trees of G. A bijection $\varphi : E(T') \to E(T)$ is called *exchange bijection*, if for each $e \in E(T')$, the graph $T' - e + \varphi(e)$ is a spanning tree of G. An exchange bijection φ is *monotone* if for each edge $e \in E(T')$ we have $w(e) \leq w(\varphi(e))$. For any two spanning trees T and T' a canonical exchange bijection exists: note that the edge sets of spanning trees of G are the bases of the graphic matroid $M(G)$ of G. By the strong basis exchange property of matroids there is a bijection between $E(T) \setminus E(T')$ and $E(T') \setminus E(T)$ with the required properties, which can be extended to an exchange bijection by mapping each item in $E(T) \cap E(T')$ to itself. Furthermore, if T' is an MST then for any spanning tree T, a canonical exchange bijection is monotone.

We generalize monotone exchange bijections as follows.

Definition 5.10. Let $\alpha \in [0,1]$ and let T, T' be spanning trees of G. An exchange bijection $\varphi : E(T') \to E(T)$ is *α-monotone* if for each edge $e \in E(T')$ we have

1. $w(e) \leq \frac{1}{\alpha} w(\varphi(e))$, if $e \in \overline{F}$ and $\varphi(e) \in F$,
2. $w(e) \leq w(\varphi(e))$, if either $e, \varphi(e) \in \overline{F}$ or $e, \varphi(e) \in F$, and
3. $w(e) \leq \alpha w(\varphi(e))$, if $e \in F$ and $\varphi(e) \in \overline{F}$.

For any spanning tree T of G, there is an α-monotone exchange bijection from an α-MST to T.

Lemma 5.11. *Let $\alpha \in [0, 1]$, let T_α be an α-MST of G and let T be any spanning tree of G. Then there is an α-monotone exchange bijection $\varphi : E(T_\alpha) \to E(T)$.*

Proof. By the discussion above we have that there is a monotone exchange bijection $\varphi : T_\alpha \to T$ between an α-MST T_α and any spanning tree T of G with respect to the weight function w_α. By substituting w_α with w we observe that φ is α-monotone with respect to w. ■

The following technical lemma is key to our charging argument in the analysis of Algorithm 4 in Sections 5.3.5 and 5.3.6.

Lemma 5.12. *Let $\alpha, \alpha' \in [0, 1]$, let T be a spanning tree contained in an optimal solution to* I*, and let T_α (resp., $T_{\alpha'}$) be an α-MST (resp., α'-MST) of G. Then, for an α-monotone exchange bijection $\varphi : E(T_\alpha) \to E(T)$ there is an α'-monotone exchange bijection $\varphi' : E(T_{\alpha'}) \to E(T)$, such that $\varphi(e) = \varphi'(e)$ for each $e \in E(T_\alpha) \cap E(T_{\alpha'})$.*

Proof. Without loss of generality, let $\alpha \leq \alpha'$ and let $\varphi : E(T_\alpha) \to E(T)$ be an α-monotone exchange bijection. Let $0 \leq q_1 < q_2 < \ldots < q_n \leq 1$ be the threshold values for $E(G)$ with respect to the weights w. By the definition of threshold values, for each threshold value q there are at least two q-MSTs. Furthermore, we may assume without loss of generality that for each threshold value q, there are precisely two q-MSTs. If this is not the case, then the reason is that there are at least two different pairs of edges, such that the two edges of each pair have the same scaled cost. We may break ties in an arbitrary but consistent way by slightly perturbing the weights and hence obtain two different thresholds, one for each pair. By iterating this argument, we have the claimed property that there are at most two q-MSTs for a threshold value q and furthermore, that T_α (resp., $T_{\alpha'}$) is an α-MST (resp., α'-MST) and φ is an α-monotone exchange bijection.

Observe that T_α is a q_i-MST, where q_i is the smallest threshold value such that $\alpha \leq q_i$. Similarly, the tree $T_{\alpha'}$ is a q_j-MST, where q_j is the largest threshold value such that $q_j \leq \alpha'$. We will reduce the task of constructing an α'-monotone exchange bijection $\varphi' : E(T_{\alpha'}) \to E(T)$ with the desired properties to the case that the symmetric difference of T_α and $T_{\alpha'}$ has size at most two. If $q_i = q_j$ then this is the case. Otherwise, note that by our assumption above there is precisely one q_{i+1}-MST T_{i+1} that is also a q_i-MST. Furthermore, the size of the symmetric difference of T_{i+1} and T_α is at most two. We construct a q_{i+1}-monotone exchange bijection $\varphi_{i+1} : E(T_{i+1}) \to E(T)$ that agrees with φ on $E(T_\alpha) \cap E(T_{i+1})$. We then replace T_α by T_{i+1} and φ by φ_{i+1} and iterate our argument. In each step we reduce the size of the symmetric difference with $T_{\alpha'}$ by two.

We now show how to construct an exchange bijection φ_2 with the desired properties, given two q_i-MSTs T_1 and T_2 and a q_i-monotone exchange bijection $\varphi_1 : E(T_1) \to E(T)$, such that the symmetric difference of $E(T_1)$ and $E(T_2)$ has size exactly two. Note that if there is a threshold value q_{i+1}, then without loss of generality the tree T_2 is also a q_{i+1}-MST. Let $E(T_1) \setminus E(T_2) = \{e\}$ and $E(T_2) \setminus E(T_1) = \{e'\}$. Consider the bijection $\varphi_2 : E(T_2) \to E(T)$ such that $\varphi_2(f) := \varphi_1(f)$ for each $f \in E(T_1) \cap E(T_2)$ and $\varphi_2(e') := \varphi_1(e)$ otherwise. We first show that φ_2 is an exchange bijection. By the definition of φ_2 it suffices to consider the edge e'. Since $e \notin E(T_2)$ but $e \in E(T_1)$ we have that e' and e are contained in a cycle of $T_2 + e$. Since φ_1 is an exchange bijection, the edge $\varphi_1(e)$ is on a cycle of $T_1 + \varphi_1(e)$. Therefore, the graph $T_2 + \varphi_1(e) = T_1 - e + e' + \varphi_1(e)$ contains a cycle visiting e'. We conclude that φ_2 is an exchange bijection.

It remains to show that φ_2 is q_i-monotone. Since e' is contained in the q_i-MST T_2, we have that $w_{q_i}(e') \leq w_{q_i}(\varphi_1(e))$. Therefore φ_2 is a q_i-monotone exchange bijection such that φ_1 and φ_2 agree on each edge in $E(T_i) \cap E(T_{i+1})$. Furthermore, if there is a threshold value q_{i+1}, then the exchange bijection φ_2 is also q_{i+1}-monotone. This concludes the proof of Lemma 5.12. ■

It will be more convenient for us to apply the following corollary rather than Lemma 5.12.

Corollary 5.13. *Let $0 \leq \alpha_1 \leq \alpha_2 \leq \ldots \leq \alpha_N \leq 1$ and for $1 \leq i \leq N$ let T_i be an α_i-MST. Furthermore let T be a spanning tree in an optimal solution to* I *and let $\varphi_1 : E(T_1) \to E(T)$ be an α_1-monotone exchange bijection. Then for $1 < i \leq N$ there are α_i-monotone exchange bijections $\varphi_i : E(T_i) \to E(T)$, such that $\varphi_{i-1}(e) = \varphi_i(e)$ for each $e \in E(T_{i-1}) \cap E(T_i)$.*

Proof. For $1 < i \leq N$ inductively apply Lemma 5.12 in order to obtain α_i-monotone exchange bijections $\varphi_i : E(T_i) \to E(T)$ with the desired properties. ■

5.3.4 Simple Analysis of Algorithm 4: A 2.8-approximation

In this section we give an analytic upper bound of 14/5 on the approximation ratio of Algorithm 4. In our analysis we consider just the contribution of Algorithms A and B and ignore the contribution of Algorithm C. We obtain an analytic upper bound on the value of the min-max-min optimization problem (5.2) in terms of c_{TAP} and c_{2EC}, which gives in turn an upper bound on the approximation ratio of Algorithm 4. Note that the combined worst-case of Algorithm A and C is much better than their individual ratios. One of the key insights is that the two algorithms exhibit their worst-case performance if Algorithm B runs with a scaling factor of 1/2.

Let us fix some $\alpha \in [0, 1]$ and let T_α be an α-MST of G. We consider the graph $G_\alpha = (V', E_\alpha)$ obtained from T_α by identifying for $1 \le j \le r$ the vertex set of the 2-edge-connected component C_j of the optimal solution Z^* with a single vertex, discarding loops but not parallel edges. Since T_α is a tree, the graph G_α is connected. Let T be a spanning tree of Z^* and let $\varphi : T_\alpha \to T$ be an α-monotone exchange bijection, which exists according to Lemma 5.11. Since every spanning tree of Z^* contains E' we have that $E' \subseteq \varphi(E_\alpha)$.

We partition the edge set of T_α into four parts D_α, O_α, F_α, and S_α as follows.

- $D_\alpha := \{e \in E(T_\alpha) \cap F \mid \varphi(e) \in E'\}$
- $O_\alpha := \{e \in E(T_\alpha) \cap \overline{F} \mid \varphi(e) \in E'\}$
- $F_\alpha := \{e \in E(T_\alpha) \cap F \mid \varphi(e) \in E(T) \setminus E'\}$
- $S_\alpha := \{e \in E(T_\alpha) \cap \overline{F} \mid \varphi(e) \in E(T) \setminus E'\}$

Note that the partition of $E(T_\alpha)$ depends on the optimal solution Z^* and the spanning tree T of Z^*, so we cannot expect to compute it efficiently. We now define the following variables.

$$
\begin{aligned}
b_\alpha &:= \frac{w(\varphi(O_\alpha))}{\mathrm{OPT(I)}} \text{ and } c_\alpha := \frac{w(\varphi(S_\alpha))}{\mathrm{OPT(I)}} \\
b_0 &:= \frac{w(E') - w(\varphi(O_\alpha))}{\mathrm{OPT(I)}} \text{ and } c_0 := \frac{w(T - E') - w(\varphi(S_\alpha))}{\mathrm{OPT(I)}} \\
\xi &= \frac{\mathrm{OPT(I)} - w(T)}{\mathrm{OPT(I)}}
\end{aligned}
$$

So b_α (c_α, resp.) is the fraction of the weight of OPT(I) of the safe edges in T_α that are mapped to safe cut edges in T (resp., edges of $E(T) - E'$). The value b_0 (resp., c_0) represents the fraction of the weight of OPT(I) of unsafe edges in T_α that are charged to safe cut edges in T (resp., edges of $E(T) - E'$). Finally, ξ is the fraction of the weight of OPT(I) that is not contributed by $E(T)$.

Observe that the following holds.

1. $b_\alpha, c_\alpha, b_0, c_0, \xi \in [0, 1]$,
2. $b_\alpha + b_0 = w(E')/\mathrm{OPT(I)}$,
3. $c_\alpha + c_0 = w(T - E')/\mathrm{OPT(I)}$, and
4. $b_0 + b_\alpha + c_0 + c_\alpha + \xi = 1$

We use the properties of α-monotone exchange bijections to show that the minimum weight of the solution computed by Algorithm B run with scaling factor α and the solution computed by Algorithm A can be bounded in terms of b_α, b_0, c_α, c_0, ξ, and OPT(I) as follows.

Lemma 5.14. *Suppose we run Algorithm 4 with a single scaling factor* $\alpha \in [0, 1]$ *(i.e.,* $W = \{\alpha\}$*) and let* Z^B_α *be the solution computed by Algorithm B with scaling factor* α *in Algorithm 4. Then*

$$
\begin{aligned}
w(Z^B_\alpha) \le \Big(&(c_{\mathrm{TAP}} + \alpha) b_0 + b_\alpha + (c_{\mathrm{TAP}} + 1) c_0 \\
&+ (c_{\mathrm{TAP}} + 1/\alpha) c_\alpha + c_{\mathrm{TAP}} \cdot \xi \Big) \cdot \mathrm{OPT(I)} \ .
\end{aligned}
$$

Proof. Let T_α be the α-MST computed by Algorithm B and let H_α be the solution to the WTAP instance (G, T'_α, w) computed by Algorithm B. The following bound on $w(T_\alpha)$ follows in a straight-forward manner from Definition 5.7.

Claim 1. *We have* $w(T_\alpha) \leq (\alpha b_0 + b_\alpha + c_0 + c_\alpha/\alpha) \cdot \mathrm{OPT(I)}$.

Proof. Since T_α is an α-MST and $\varphi : T_\alpha \to T$ is an α-monotone exchange bijection, we have that for each edge e of T_α exactly one of the following cases applies.

1. If $e \in D_\alpha$, then we have $w(e) \leq \alpha \cdot \varphi(e)$, or
2. if $e \in O_\alpha \cup F_\alpha$, then we have $w(e) \leq w(\varphi(e))$, or
3. if $e \in S_\alpha$, then $w(e) \leq \frac{1}{\alpha} w(\varphi(e))$.

By summing over the above inequalities and applying the definition of $b_0, b_\alpha, c_0, c_\alpha$ we have

$$\begin{aligned} w(T_\alpha) &\leq (\alpha \cdot w(\varphi(D_\alpha)) + w(E' \setminus \varphi(D_\alpha)) + \varphi(S_\alpha)/\alpha + \varphi(F_\alpha)) \cdot \mathrm{OPT(I)} \\ &\leq (\alpha \cdot b_0 + b_\alpha + c_0 + c_\alpha/\alpha) \cdot \mathrm{OPT(I)} \end{aligned}$$

as claimed. This proves Claim 1. □

It remains to bound the cost of H_α. Note that H_α is a c_{TAP}-approximate solution of an optimal solution for the augmentation problem. To bound the cost of an optimal augmentation we demonstrate the existence of a feasible augmentation. We show that $Y_\alpha := \varphi(D_\alpha) \cup \varphi^{-1}(Z^* - E')$ is such a feasible augmentation.

Claim 2. $T_\alpha \cup Y_\alpha$ *is a feasible solution to* I.

Proof. Clearly $E(T_\alpha) \cup Y_\alpha$ is a connected spanning subgraph since it contains the tree T_α. It remains to show that $E(T_\alpha) \cup Y_\alpha - f$ is connected for each $f \in F \cap (E(T_\alpha) \cup Y_\alpha)$. This is clearly true for the edges in F_α since we added $\varphi^{-1}(Z^* - E')$. Thus let $f \in D_\alpha$. Since $f \in F$ and $\varphi(f) \in \overline{F}$, we have that $f \neq \varphi(f)$. By the definition of exchange bijections there is a cycle in $T_\alpha \cup \varphi(f)$ going through f. This concludes the proof of Claim 2. □

By the definition of $b_0, b_\alpha, c_0, c_\alpha$, and ξ we have

$$\begin{aligned} w(Y_\alpha) &\leq w(\varphi(D_\alpha)) + w(Z^* - E') \\ &\leq w(\varphi(D_\alpha)) + w(\varphi(F_\alpha)) + w(\varphi(S_\alpha)) + \xi \\ &\leq (b_0 + c_0 + c_\alpha + \xi) \cdot \mathrm{OPT(I)} . \end{aligned}$$

Since Algorithm 4 uses a c_{TAP}-approximation for WTAP, we have that $w(H_\alpha) \leq c_{\text{TAP}} \cdot w(Y_\alpha)$.

Using these bounds and Claims 1 and 2, we obtain

$$\begin{aligned} w(Z^B_\alpha) \leq \Big(&(c_{\text{TAP}} + \alpha) b_0 + b_\alpha + (c_{\text{TAP}} + 1) c_0 \\ &+ (c_{\text{TAP}} + 1/\alpha) c_\alpha + c_{\text{TAP}} \cdot \xi \Big) \cdot \text{OPT(I)} \ . \end{aligned}$$

as claimed. This proves Lemma 5.14. ■

Next, we provide a bound on the cost of the solution returned by Algorithm A.

Lemma 5.15. *Let $\alpha \in [0,1]$ and let Z^A be a solution computed by Algorithm A in Algorithm 4. Then*

$$w(Z^A) \leq (c_{2\text{EC}} + c_{2\text{EC}} \cdot (\alpha \cdot b_0 + b_\alpha)) \cdot \text{OPT(I)} \ .$$

Proof. We demonstrate the existence of a feasible solution $Y \subseteq E$ to I. Recap that Algorithm A computes a 2-ECSS solution to I where each safe edge $e \in \overline{F}$ has a parallel edge e'. Let $Y := Z^* \cup \varphi(D_\alpha) \cup \{e' : e \in E' \setminus \varphi(D_\alpha)\}$. Since $Z^* \subseteq Y$, each edge in $Z^* \setminus E'$ is contained in a 2-edge-connected component. Furthermore, since Y additionally contains $\varphi(D_\alpha) \cup \{e' : e \in E' \setminus \varphi(D_\alpha)\}$, each edge in E' is also contained in a 2-edge-connected component. Thus, Y is a 2-edge connected spanning subgraph.

It remains to bound the cost of Y. We bound the cost of each part individually. Clearly we have $w(Z^*) = \text{OPT(I)}$. By the definition of b_0 and b_α, we have $w(\varphi(D_\alpha)) \leq \alpha \cdot b_0 \cdot \text{OPT(I)}$ and $w(\{e' : e \in E' \setminus \varphi(D_\alpha)\}) \leq b_\alpha \cdot \text{OPT(I)}$.

Since Algorithm A uses a $c_{2\text{EC}}$-approximation for the 2-ECSS problem, we obtain

$$w(Z^A) \leq (c_{2\text{EC}} + c_{2\text{EC}} \cdot (\alpha \cdot b_0 + b_\alpha)) \cdot \text{OPT(I)} \ .$$

■

Proposition 5.16. *Algorithm 4 is an approximation algorithm for* FGC *with ratio*

$$\min\left\{1+c_{\text{TAP}}, \frac{c_{2\text{EC}}(4c_{\text{TAP}}^2 + \sqrt{1+4c_{\text{TAP}}} - 2c_{\text{TAP}} - 1)}{(1-c_{2\text{EC}})\sqrt{1+4c_{\text{TAP}}} + (c_{\text{TAP}} + c_{2\text{EC}} - 1)2c_{\text{TAP}} - 1 + c_{2\text{EC}}}\right\}.$$

Proof. The guarantee of Algorithm 4 is clearly bounded by the weight of the solution computed by Algorithm B. This weight is at most $1+c_{\text{TAP}}$, which is the first part of the minimum. For the second part we know that $w(Z^A) = w(Z^B_\alpha)$. By Lemmas 5.14 and 5.15 we have

$$w(Z^B_\alpha) \leq \Big((c_{\text{TAP}} + \alpha)b_0 + b_\alpha + (c_{\text{TAP}} + 1)c_0 + (c_{\text{TAP}} + \frac{1}{\alpha})c_\alpha + c_{\text{TAP}} \cdot \xi\Big) \cdot \text{OPT(I)}$$

and

$$w(Z^A) \leq (c_{\text{2EC}} + c_{\text{2EC}} \cdot (\alpha \cdot b_0 + b_\alpha)) \cdot \text{OPT(I)} \ .$$

Let $\mathcal{A}(\text{I})$ be the cost of a solution computed by Algorithm 4. An upper bound on the approximation ratio of Algorithm 4 is

$$\begin{aligned} \mathcal{A}(\text{I}) \leq \max_{b_0, b_\alpha, c_0, c_\alpha, \xi \in [0,1]} \ \min_{\alpha \in [0,1]} \quad & \{w(Z^A), w(Z^B_\alpha)\} \\ \text{subject to} \quad & b_0 + b_\alpha + c_0 + c_\alpha + \xi = 1. \end{aligned} \tag{5.3}$$

It is easy to see that the maximum is obtained if $c_0 = \xi = 0$. By substituting $c_\alpha = 1 - b_0 - b_\alpha$ we obtain

$$w(Z^B_\alpha) \leq ((c_{\text{TAP}} + \alpha)b_0 + b_\alpha + (c_{\text{TAP}} + 1/\alpha)(1 - b_0 - b_\alpha)) \cdot \text{OPT(I)} \ .$$

By putting $w(Z^A) = w(Z^B_\alpha)$ we obtain

$$b_0 = \frac{(-b_\alpha \cdot c_{\text{2EC}} - c_{\text{2EC}}) \cdot \alpha - \alpha \cdot b_\alpha \cdot c_{\text{TAP}} + b_\alpha \cdot \alpha + c_{\text{TAP}}\alpha - b_\alpha + 1}{\alpha^2 \cdot c_{\text{2EC}} - \alpha^2 + 1} \ .$$

Plugging this into $w(Z^A)$ and setting $\alpha = \frac{-1+\sqrt{1+4c_{\text{TAP}}}}{2c_{\text{TAP}}}$ (recall that we are free to choose α) we obtain

$$\mathcal{A}(\text{I}) \leq \frac{c_{\text{2EC}}(4c_{\text{TAP}}^2 + \sqrt{1 + 4c_{\text{TAP}}} - 2c_{\text{TAP}} - 1)}{(1 - c_{\text{2EC}})\sqrt{1 + 4c_{\text{TAP}}} + (c_{\text{TAP}} + c_{\text{2EC}} - 1)2c_{\text{TAP}} - 1 + c_{\text{2EC}}} \ .$$

■

By setting $c_{\text{2EC}} = c_{\text{TAP}} = 2$ we directly obtain the following result.

Corollary 5.17. *Algorithm 4 is a polynomial-time* 14/5*-approximation algorithm for* FGC *for* $c_{\text{TAP}} = c_{\text{2EC}} = 2$.

5.3.5 Refined Analysis of Algorithm 4: A 2.523-approximation

In this section we give an analytic upper bound of 2.523 on the approximation ratio of Algorithm 4. For our analysis it suffices to run Algorithm A together with Algorithm C. Using α-monotone exchange bijections from Section 5.3.3, we determine bounds $f^A(\cdot)$ and $f^C(\cdot)$ for the optimization problem (5.2), where $f^C(\cdot)$ depends on a selection of scaling factors and some other parameters to be introduced shortly. We then transform problem (5.2) into a maximization problem, which we solve analytically. Recall that according to Lemma 5.6, the selection of scaling factors in Algorithm 4 is optimal. Surprisingly, a worst-case instance for our bounds $f^A(\cdot)$ and $f^C(\cdot)$ in fact has a single threshold value which is $1/\sqrt{c_{2\mathrm{EC}}}$. However, to obtain the approximation ratio of 2.523 it is crucial to execute Algorithm 4 with all threshold values of the given instance.

Let $\mathcal{I}(N)$ be a class of instances of FGC with at most N threshold values in the sense of Definition 5.7. In the following, suppose that $\mathrm{I} \in \mathcal{I}(N)$ and recall that an optimal solution $Z^* \subseteq E(G)$ of I consists of r 2-edge-connected components $C_1, C_2, \dots, C_r$ that are joined together by safe edges $E' \coloneqq \{\overline{f}_1, \overline{f}_2, \dots, \overline{f}_{r-1}\} \subseteq \overline{F}$ in a tree-like fashion. Moreover, for any spanning tree T contained in the optimal solution Z^* we have $E' \subseteq T$.

Observe that since there is an unsafe edge for each safe edge of the same weight in G, we have that each threshold value of the safe edges is in $[0, 1]$. Let $0 \le \alpha_1 \le \alpha_2 \le \ldots \le \alpha_N \le 1$ be the N threshold values of I in non-decreasing order. To prepare our analysis, we consider for $i \in \{1, 2, \dots, N\}$ an α_i-MST T_i, an α_i-monotone exchange bijection $\varphi_i : T_i \to T$ and a weight $w_i \coloneqq w_{\alpha_i}$. For $2 \le i \le N$ we choose φ_i such that for each $e \in E(T_{i-1}) \cap E(T_i)$ we have $\varphi_{i-1}(e) = \varphi_i(e)$ (in accordance with Corollary 5.13). In order to define the parameters of the optimization problem (5.2), for $1 \le i \le N$, we partition the edge set of the α_i-MST T_i into four parts D_i, O_i, F_i, and S_i as follows. This is similar to the partition described in Section 5.3.4.

- $D_i \coloneqq \{e \in E(T_i) \cap F \mid \varphi_i(e) \in E'\}$
- $O_i \coloneqq \{e \in E(T_i) \cap \overline{F} \mid \varphi_i(e) \in E'\}$
- $F_i \coloneqq \{e \in E(T_i) \cap F \mid \varphi_i(e) \in E(T) \setminus E'\}$

- $S_i \coloneqq \{e \in E(T_i) \cap \overline{F} \mid \varphi_i(e) \in E(T) \setminus E'\}$

The parameters of problem (5.2) are given as follows. For $1 \leq i \leq N$ we let $E_i^{\bar{F}}$ (resp., E_i^F) be the set of edges in E' (resp., $E(T) - E'$) that have threshold value α_i, i.e. $E_i^{\bar{F}} \coloneqq \{e \in E' \mid \alpha_e = \alpha_i\}$ and $E_i^F \coloneqq \{e \in E(T) - E' \mid \alpha_e = \alpha_i\}$. For $1 \leq i \leq N$ we let $\beta_i = w(E_i^{\bar{F}})/\mathrm{OPT(I)}$ and $\gamma_i = w(E_i^F)/\mathrm{OPT(I)}$ be the fraction of the weight of the optimal solution that is contributed by the edges in $E_i^{\bar{F}}$ (resp., E_i^F). Finally, let $\xi \in [0,1]$ be the the fraction of the weight of the optimal solution that is not contributed by the tree T; e.g., $\xi \coloneqq \frac{w(Z^*)-w(T)}{\mathrm{OPT(I)}}$. The following properties of β_i, γ_i, $1 \leq i \leq N$, and ξ are readily verified:

1. $\beta_1, \beta_2, \ldots \beta_N, \gamma_1, \gamma_2, \ldots \gamma_N, \xi \in [0,1]$,
2. $\sum_{j=1}^N \beta_j = \frac{w(E')}{\mathrm{OPT(I)}}$,
3. $\sum_{j=1}^N \gamma_j = \frac{w(T-E')}{\mathrm{OPT(I)}}$, and
4. $\xi = 1 - \sum_{j=1}^N \beta_j - \sum_{j=1}^N \gamma_j$.

We now bound the cost of the solutions Z_i^C and Z^A returned by Algorithm C (Algorithm A, resp.) in terms of the parameters.

Lemma 5.18. *Suppose we run Algorithm 4 with the optimal threshold values $W = \{\alpha_i\}_{1\leq i\leq N}$. Let Z_i^C be the solution computed by Algorithm C with scaling factor α_i in Algorithm 4. Then*

$$w(Z_i^C) \leq \Bigg(1 + \sum_{j=1}^{i-1}(c_{2\mathrm{EC}} + c_{2\mathrm{EC}}\alpha_j - 1)\beta_j + (c_{2\mathrm{EC}} - 1) \cdot \sum_{j=1}^{N} \gamma_j \\ + \sum_{j=i}^{N} \frac{1}{\alpha_j}\gamma_j + (c_{2\mathrm{EC}} - 1) \cdot \xi \Bigg) \cdot \mathrm{OPT(I)} \ .$$

Proof. Let T_i^S be the safe edges of the tree T_i and let $\varphi_i : E(T_i) \to E(T)$ be an α_i-monotone exchange bijection, where T is a spanning tree of the optimal solution Z^* to the instance $(G, w, \overline{F})$.

By contracting each edge of T_i^S in G we obtain the graph $G_i^S \coloneqq G/E(T_i^S)$. Algorithm C computes a $c_{2\mathrm{EC}}$-approximate solution to the instance (G_i^S) of 2-ECSS.

Claim 1. *The set*

$$Y_i := \left(\bigcup_{e \in E' \setminus \varphi_i(E(T_i^S))} \{e, \varphi_i^{-1}(e)\} \right) \cup \bigcup_{1 \leq j \leq r} E(C_j)$$

*of edges is a feasible solution to the 2-*ECSS *instance* (G_i^S).

Proof. Clearly (V, Y_i) is a connected graph. It remains to argue that each edge $e \in Y_i$ is contained in some cycle. This is certainly true for each edge e of a component C_j, $1 \leq j \leq r$. It remains to show that the edges in $\bigcup_{e \in E' \setminus \varphi_i(E(T_i^S))} \{e, \varphi_i^{-1}(e)\}$ are contained in some cycle of G_i^S. Since φ_i is an exchange bijection, an edge $e \in E' \setminus \varphi_i(E(T_i^S))$ and its preimage $\varphi^{-1}(e)$ are on a cycle in $E(T) \cup \{e\}$. Since G_i^S is obtained from G by contracting the edges of the safe forest T_i^S, the edges e and $\varphi^{-1}(e)$ are also on a cycle in G_i^S. This proves Claim 1. $\square$

We now bound the cost of Z_i^C. By Claim 1 we have that $T_i^S \cup Y_i$ is a feasible solution to the FGC instance $(G, w, \overline{F})$. The algorithm then returns in polynomial time a solution Z_i^C of cost at most $w(Z_i^C) \leq w(T_i^S) + c_{2\text{EC}} w(Y_i)$. We first bound the cost of each edge of T_i^S as follows.

Claim 2. *Let* $e \in E(T_i^S)$ *and let* α_e *be its threshold value. Then we have*

$$w(e) \leq \begin{cases} \frac{1}{\alpha_e} \cdot w(\varphi_i(e)) & \text{if } \varphi_i(e) \notin E', \text{ and} \\ w(\varphi_i(e)) & \text{otherwise .} \end{cases}$$

Proof. First suppose that $\varphi_i(e) \in E'$. Since $\varphi_i(e) \in \overline{F}$ and φ_i is an α_i-monotone exchange bijection it follows that $w(e) \leq w(\varphi_i(e))$. Now let $\varphi_i(e) \notin E'$. Since φ_i is an α_i-monotone exchange bijection and since the threshold value of e is α_e, we have that $w(e) \leq \frac{1}{\alpha_e} w(\varphi_i(e))$. This concludes the proof of Claim 2. $\square$

Observe that if $e \in T_i^S$, we also have $e \in T_j^S$ for each $j \leq i$. Since the exchange bijections $\varphi_1, \varphi_2, \dots, \varphi_N$ are in accordance with Corollary 5.13, we then have $\varphi_j(e) = \varphi_i(e)$ for every $j \leq i$. Thus, according to Claim 2 we have

$$\begin{aligned}
w(T_i^S) &= \sum_{e \in T_i^S} w(e) = \sum_{e \in T_i^S} \sum_{j:\alpha_e=\alpha_j} w(e) \\
&= \sum_{e \in T_i^S:\varphi_i(e)\in E'} \sum_{j:\alpha_e=\alpha_j} w(e) + \sum_{e \in T_i^S:\varphi_i(e)\notin E'} \sum_{j:\alpha_e=\alpha_j} w(e) \\
&\leq \sum_{e \in T_i^S:\varphi_i(e)\in E'} \sum_{j:\alpha_e=\alpha_j} w(\varphi_i(e)) + \sum_{e \in T_i^S:\varphi_i(e)\notin E'} \sum_{j:\alpha_e=\alpha_j} \frac{1}{\alpha_e} \cdot w(\varphi_i(e)) \\
&\leq \left(\sum_{j=i}^{N} \beta_j + \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j} \right) \cdot \mathrm{OPT(I)} \\
&= \left(1 - \sum_{j=1}^{i-1} \beta_j - \sum_{j=1}^{N} \gamma_j - \xi + \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j} \right) \cdot \mathrm{OPT(I)} \ ,
\end{aligned}$$

where the last equality holds due to $\xi + \sum_{j=1}^{N} \beta_j + \sum_{j=1}^{N} \gamma_j = 1$. We additionally bound the cost of Y_i.

Claim 3.

$$w(Y_i) \leq \left(\sum_{j=1}^{i-1} (1+\alpha_j)\beta_j + \sum_{j=1}^{N} \gamma_j + \xi \right) \cdot \mathrm{OPT(I)}$$

Proof. We need to bound the cost of

$$Y_i = \left(\bigcup_{e \in E' \setminus \varphi_i(E(T_i^S))} \{e, \varphi_i^{-1}(e)\} \right) \cup \bigcup_{1 \leq j \leq r} E(C_j) \ .$$

We first bound the cost of $\bigcup_{e \in E' \setminus \varphi_i(E(T_i^S))} \{e, \varphi_i^{-1}(e)\}$. According to the definition of $\beta_1, \beta_2, \ldots, \beta_N$ we can bound the cost of $\{e \mid e \in E' \setminus \varphi_i(E(T_i^S))\}$ by $\sum_{j=1}^{i-1} \beta_j \cdot \mathrm{OPT(I)}$. Additionally, for each $e \in E' \setminus \varphi_i(E(T_i^S))$ we have $w(\varphi_i^{-1}(e)) \leq \alpha_e w(e)$. Therefore we can bound the cost of the edge-set $\{\varphi_i^{-1}(e) \mid e \in E' \setminus \varphi_i(E(T_i^S))\}$ by $\sum_{j=i}^{i-1} \alpha_j \beta_j \cdot \mathrm{OPT(I)}$. Finally we bound the cost of $\bigcup_{1 \leq j \leq r} E(C_j)$ by $(\xi + \sum_{j=1}^{N} \gamma_j) \cdot \mathrm{OPT(I)}$. Putting things together we obtain

$$w(Y_i) \leq \left(\sum_{j=1}^{i-1} (1+\alpha_j)\beta_j + \sum_{j=1}^{N} \gamma_j + \xi \right) \cdot \mathrm{OPT(I)} \ .$$

This concludes the proof of Claim 3. □

Finally, since the algorithm computes a $c_{2\mathrm{EC}}$-approximate solution to the 2-ECSS instance, we have

$$\begin{aligned}
w(Z_i^C) \leq & w(T_i^S) + c_{2\mathrm{EC}} \cdot w(Y_i) \\
\leq & \Bigg(\Bigg(1 - \sum_{j=1}^{i-1} \beta_j - \sum_{j=1}^{N} \gamma_j - \xi + \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j} \Bigg) \\
& + \Bigg(\sum_{j=1}^{i} (c_{2\mathrm{EC}} + c_{2\mathrm{EC}}\alpha_j)\beta_j + \sum_{j=1}^{N} c_{2\mathrm{EC}}\gamma_j + c_{2\mathrm{EC}}\xi \Bigg) \Bigg) \cdot \mathrm{OPT(I)} \\
\leq & \Bigg(1 + \sum_{j=1}^{i-1} (c_{2\mathrm{EC}} + c_{2\mathrm{EC}}\alpha_j - 1)\beta_j + (c_{2\mathrm{EC}} - 1) \cdot \sum_{j=1}^{N} \gamma_j \\
& + \sum_{j=i}^{N} \frac{1}{\alpha_j} \gamma_j + (c_{2\mathrm{EC}} - 1) \cdot \xi \Bigg) \cdot \mathrm{OPT(I)} \ ,
\end{aligned}$$

which concludes the proof of Lemma 5.18. ■

Lemma 5.19. *Suppose we run Algorithm 4 with the optimal threshold values $W = \{\alpha_i\}_{1 \leq i \leq N}$. Let Z^A be the solution computed by Algorithm A with scaling factor α_i in Algorithm 4. Then*

$$w(Z^A) \leq \Bigg(c_{2\mathrm{EC}} + c_{2\mathrm{EC}} \cdot \sum_{j=1}^{N} \alpha_j \beta_j \Bigg) \cdot \mathrm{OPT(I)} \ .$$

Proof. The algorithm computes a 2-edge-connected spanning subgraph. Recall that for each safe edge $e \in \overline{F}$ there is a parallel unsafe copy e' of the same cost. We construct a feasible solution Y_A to the 2-ECSS instance to bound the cost of Z^A.

Claim 1. $Y_A := Z^* \cup \{\varphi_N^{-1}(e)) \mid e \in E'\}$ *is a feasible solution to the 2-ECSS instance of cost at most*

$$w(Y_A) \leq \Bigg(1 + \sum_{j=1}^{N} \alpha_j \beta_j \Bigg) \cdot \mathrm{OPT(I)} \ .$$

Proof. We first show the feasibility. Clearly (V, Y_A) is connected since it contains Z^*. We now show that each $e \in E'$ is contained in some cycle in Y_A. Since for each safe edge there is an unsafe edge of the same cost, we can assume that $\varphi_N^{-1}(e) \neq e$. Then, by the definition of φ_N, the edge e and its preimage $\varphi_N^{-1}(e)$ are contained in a cycle.

It remains to bound the cost of Y_A. We partition Y_A into two disjoint sets $Y_A = Z^* \cup X_1$, where $X_1 = \{\varphi_N^{-1}(e) \mid e \in E'\}$, and bound the cost of each part individually. Clearly we have $w(Z^*) = \mathrm{OPT(I)}$.

To bound the cost of X_1, observe that for some $\varphi_N^{-1}(e) \in X_1$ we have $w(\varphi_N^{-1}(e)) \leq \alpha_e w(e)$. Thus, we may bound the cost of X_1 by

$$\begin{aligned} w(X_1) = \sum_{\varphi_N^{-1}(e)\in X_1} w(e) &= \sum_{\varphi_N^{-1}(e)\in X_1} \sum_{j:\alpha_e=\alpha_j} w(\varphi_N^{-1}(e)) \\ &\leq \sum_{\varphi_N^{-1}(e)\in X_1} \sum_{j:\alpha_e=\alpha_j} \alpha_e w(e) = \sum_{j=1}^{N} \alpha_j \beta_j \cdot \mathrm{OPT(I)}\ . \end{aligned}$$

Combining both bounds we obtain

$$w(Y_A) \leq w(Z^*) + w(X_1) \leq \left(1 + \sum_{j=1}^{N} \alpha_j \beta_j\right) \cdot \mathrm{OPT(I)}\ .$$

This proves Claim 1. □

Since the algorithm computes a $c_{2\mathrm{EC}}$-approximation, we have that

$$w(Z^A) \leq c_{2\mathrm{EC}} w(Y_A) \leq \left(c_{2\mathrm{EC}} + c_{2\mathrm{EC}} \cdot \sum_{j=1}^{N} \alpha_j \beta_j\right) \cdot \mathrm{OPT(I)}\ .$$

This concludes the proof of Lemma 5.19. ■

With the bounds from Lemmas 5.18 and 5.19 we show in the next theorem that Problem (5.2) can be simplified to the following maximization problem.

$$\begin{aligned} &\max\ c_{2\mathrm{EC}} \cdot \left(1 + \sum_{j=1}^{N} \alpha_j \hat{\beta}_j\right) \\ &\text{subject to } \sum_{j=1}^{N} \hat{\beta}_j \cdot (1 + \alpha_j(c_{2\mathrm{EC}} - 1 + c_{2\mathrm{EC}}\alpha_j)) = 1\ , \\ &\qquad 0 \leq \alpha_1 \leq \alpha_2 \leq \ldots \leq \alpha_N \leq 1\ , \\ &\qquad \hat{\beta}_j \in [0,1] \text{ for all } j \in \{1,\ldots,N\}\ . \end{aligned} \tag{5.4}$$

Theorem 5.20. *The approximation guarantee of Algorithm 4 for instances with at most N threshold values is upper bounded by the optimal value of Problem* (5.4)*.*

Proof. With the bounds from Lemmas 5.18 and 5.19 and Properties 1–4 above, for instances with at most N threshold values, we can rewrite Problem (5.2) as the following max-min optimization problem.

$$\begin{aligned} \max_{\substack{\xi\in[0,1]\\ \alpha_i\in[0,1]\,:\,1\le i\le N\\ \beta_i\in[0,1]\,:\,1\le i\le N\\ \gamma_i\in[0,1]\,:\,1\le i\le N}} \quad \min_{1\le i\le N} \quad & \{w(Z^A),\, w(Z_i^C)\} \\ \text{subject to} \quad & \sum_{j=1}^{N} \beta_j + \gamma_j + \xi = 1 \;, \\ & \alpha_1 \le \alpha_2 \le \ldots \le \alpha_N \;. \end{aligned} \tag{5.5}$$

We obtain from (5.5) a simple maximization problem as follows. Since $c_{\text{2EC}} \ge 1$, each $\beta_j, \gamma_j, j \in [N]$ as well as ξ has a positive coefficient in the bounds of the Lemmas 5.18 and 5.19. Moreover, since we maximize over $\beta_1, \beta_2, \ldots, \beta_N$, $\gamma_1, \gamma_2, \ldots, \gamma_N$, and ξ we may assume $\xi = 0$. To see this, suppose we have an optimal choice of the variables where $\xi > 0$. Then, consider the following new variables. Let $\beta'_j = \beta_j$ for $j \in [N], \gamma'_j = \gamma_j$ for $j \in [N-1]$, $\gamma'_N = \gamma_N + \xi$ and $\xi' = 0$. Now observe that the value of the minimum over $w(Z^A)$ and $w(Z_i^C), i \in [N]$ for the new variables is at least as large as for the old ones. Thus we can assume that $\xi = 0$ and have

$$\sum_{j=1}^{N} \beta_j + \sum_{j=1}^{N} \gamma_j = 1 \;.$$

Hence, it follows that the approximation guarantee of Algorithm 4 is upper bounded by the following optimization problem

$$\begin{aligned} \max_{\substack{\alpha_i\in[0,1]\,:\,1\le i\le N\\ \beta_i\in[0,1]\,:\,1\le i\le N\\ \gamma_i\in[0,1]\,:\,1\le i\le N}} \quad \min_{1\le i\le N} \quad & \{w(Z^A),\, w(Z_i^C)\} \\ \text{subject to} \quad & \sum_{j=1}^{N} \beta_j + \gamma_j = 1 \;, \\ & \alpha_1 \le \alpha_2 \le \ldots \le \alpha_N \;, \end{aligned}$$

where

$$w(Z_i^C) = 1 + \sum_{j=1}^{i-1} (c_{\text{2EC}} - 1 + c_{\text{2EC}}\alpha_j)\beta_j + (c_{\text{2EC}} - 1) \cdot \sum_{j=1}^{N} \gamma_j + \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j}$$

for each $i \in [N]$ and

$$w(Z^A) = c_{2\mathrm{EC}} + c_{2\mathrm{EC}} \cdot \sum_{j=1}^{N} \alpha_j \beta_j \ .$$

Let us assume that the optimal value for this optimization problem is $\rho \in [c_{2\mathrm{EC}}, 2c_{2\mathrm{EC}}]$ and let $\beta_1^*, \ldots, \beta_N^*$ and $\gamma_1^*, \ldots, \gamma_N^*$ be the optimal values of the respective variables. Since Algorithm C gives a 5-approximation only, we know that the minimum of the optimization problem above is equal to $w(Z^A)$. Thus we have

$$\rho = c_{2\mathrm{EC}} \cdot \left(1 + \sum_{j=1}^{N} \alpha_j \beta_j^*\right) .$$

Therefore we have that

$$\begin{aligned}
w(Z_N^C) - w(Z^A) &= 1 + \sum_{j=1}^{N-1} (c_{2\mathrm{EC}} - 1 + c_{2\mathrm{EC}}\alpha_j)\beta_j^* + (c_{2\mathrm{EC}} - 1) \cdot \sum_{j=1}^{N} \gamma_j^* \\
&\quad + \frac{\gamma_N^*}{\alpha_N} - (c_{2\mathrm{EC}} + c_{2\mathrm{EC}} \cdot \sum_{j=1}^{N} \alpha_j \beta_j^*) \\
&= 1 + \sum_{j=1}^{N} (c_{2\mathrm{EC}} - 1 + c_{2\mathrm{EC}}\alpha_j)\beta_j^* - (c_{2\mathrm{EC}} - 1 + c_{2\mathrm{EC}}\alpha_N)\beta_N^* \\
&\quad + (c_{2\mathrm{EC}} - 1) \cdot \sum_{j=1}^{N} \gamma_j^* + \frac{\gamma_N^*}{\alpha_N} - c_{2\mathrm{EC}} - c_{2\mathrm{EC}} \cdot \sum_{j=1}^{N} \alpha_j \beta_j^* \\
&= \frac{\gamma_N^*}{\alpha_N} - (c_{2\mathrm{EC}} - 1 + c_{2\mathrm{EC}}\alpha_N)\beta_N^* \\
&\geq 0 \ ,
\end{aligned}$$

where the third equality follows from $\sum_{j=1}^{N} \beta_j + \gamma_j = 1$. In fact, we may assume that the last inequality is an equality. If not, then we can reduce γ_N^* by some fraction and increase γ_{N-1} by the same fraction. This yields a feasible solution but may increase the weight of the maximum. Recursively, for $1 \leq i \leq N - 1$

we obtain

$$\begin{aligned}
w(Z_i^C) - w(Z^A) &= 1 + \sum_{j=1}^{i-1} (c_{2\mathrm{EC}} - 1 + c_{2\mathrm{EC}}\alpha_j)\beta_j + (c_{2\mathrm{EC}} - 1) \cdot \sum_{j=1}^{N} \gamma_j \\
&\quad + \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j} - c_{2\mathrm{EC}} + c_{2\mathrm{EC}} \cdot \sum_{j=1}^{N} \alpha_j \beta_j \\
&= \frac{\gamma_i^*}{\alpha_i} - (c_{2\mathrm{EC}} - 1 + c_{2\mathrm{EC}}\alpha_i)\beta_i^* \\
&= 0 \ .
\end{aligned}$$

Thus we obtain $\gamma_i^* = \alpha_i \cdot (c_{2\mathrm{EC}} - 1 + c_{2\mathrm{EC}}\alpha_i) \cdot \beta_i^*$ for $1 \le i \le N$. Substituting this for γ_j^* in $\sum_{j=1}^N \beta_j^* + \gamma_j^* = 1$, we obtain

$$\sum_{j=1}^{N} (1 + \alpha_i(c_{2\mathrm{EC}} - 1 + c_{2\mathrm{EC}}\alpha_i))\beta_i^* = 1 \ .$$

Hence, in summary the optimization problem (5.2) with bounds $w(Z^A)$ and $w(Z_i^C)$ from Lemmas 5.18 and 5.19 can be written as maximization Problem (5.4). This proves Theorem 5.20. ■

Next, we give an analytic solution to problem (5.4) in terms of $c_{2\mathrm{EC}}$ which yields a ratio of $(8 + 4\sqrt{2})/(4 + \sqrt{2}) < 2.523$ for Algorithm 4 for $c_{2\mathrm{EC}} = 2$, the best known approximation guarantee for 2-ECSS. This is the bound claimed in Theorem 5.2.

Theorem 5.21. *Algorithm 4 has an approximation guarantee of*

$$\frac{c_{2\mathrm{EC}} \cdot (c_{2\mathrm{EC}} + 2\sqrt{c_{2\mathrm{EC}}})}{2\sqrt{c_{2\mathrm{EC}}} + c_{2\mathrm{EC}} - 1} \ .$$

Proof. Consider an optimal solution to Problem (5.4) with optimal values $\alpha_1^* < \alpha_2^* < \ldots < \alpha_N^*$ and $\hat{\beta}_1^*, \hat{\beta}_2^*, \ldots, \hat{\beta}_N^*$. It is easy to see that an optimal solution has only one $\hat{\beta}_k^* \neq 0$, $k \in [N]$. We then have

$$\hat{\beta}_k^* = \frac{1}{1 + \alpha_k(c_{2\mathrm{EC}} - 1 + c_{2\mathrm{EC}}\alpha_k)} \ ,$$

where $\alpha_k \in [0,1]$. Optimizing over $\alpha_k \in [0,1]$ yields the optimal value $\alpha_k = 1/\sqrt{c_{2\mathrm{EC}}}$. Thus we obtain $\hat{\beta}_k^* = \frac{\sqrt{c_{2\mathrm{EC}}}}{2\sqrt{c_{2\mathrm{EC}}} + c_{2\mathrm{EC}} - 1}$ and the optimal value for Problem (5.4) is $\frac{c_{2\mathrm{EC}} \cdot (c_{2\mathrm{EC}} + 2\sqrt{c_{2\mathrm{EC}}})}{2\sqrt{c_{2\mathrm{EC}}} + c_{2\mathrm{EC}} - 1}$. Now observe that the solution does not depend on the number N of threshold values. Hence the bound holds for instances with any number of threshold values. ■

5.3.6 Improved Analysis of Algorithm 4 for Bounded-weight Instances: A 2.404-approximation

In this section we give a computational proof for an upper bound of 2.404 on the approximation ratio of Algorithm 4 for bounded-weight instances. Similarly to Section 5.3.5 we give an upper bound on the value of the min-max-min optimization problem (5.2), which gives in turn an upper bound on the approximation ratio of Algorithm 4. In contrast to our previous analysis, we also use Algorithm B in order to exploit recent progress in the approximation of WTAP on bounded-weight instances. Hence, we consider the simultaneous worst-case behavior of all three algorithms. However, this prevents us from giving an improved *analytic* upper bound on the approximation ratio of Algorithm 4 in this setting. Instead, we give a computational upper bound on the approximation ratio of Algorithm 4, which is obtained by a non-linear programming (NLP) solver. Since such problems tend to be hard to solve, we relax the first min in the Problem (5.2) by selecting appropriate scaling factors and transform the relaxation into a quadratic maximization problem, which we solve computationally. A proof that our claimed upper bound indeed holds is given by the branch-and-bound tree of the solver. Nevertheless, we encourage the reader to independently verify the bound by solving (5.6). Finding an analytic proof of the improved approximation guarantee is an interesting open problem.

We start similarly to Section 5.3.5. Let $N \in \mathbb{N}$ be any natural number and let $0 = \alpha_0 \leq \alpha_1 \leq \alpha_2 \leq \ldots \leq \alpha_{N+1} = 1$ (not necessarily the threshold values of the given instance) be $N+2$ values in $[0,1]$ in non-decreasing order. For $i \in \{0,1,\ldots,N+1\}$, let T_i be an α_i-MST, let $\varphi_i : T_i \to T$ be an α_i-monotone exchange bijection, where T is a spanning tree of Z^* and let $w_i := w_{\alpha_i}$. For $1 \leq i \leq N+1$, we choose φ_i in accordance with Corollary 5.13, that is, we have $\varphi_{i-1}(e) = \varphi_i(e)$ for each $e \in E(T_{i-1}) \cap E(T_i)$. This will be useful later on for our charging argument. For $0 \leq i \leq N+1$, we partition the edge set of T_i into four parts D_i, O_i, F_i, and S_i as in Section 5.3.5.

- $D_i := \{e \in E(T_i) \cap F \mid \varphi_i(e) \in E'\}$
- $O_i := \{e \in E(T_i) \cap \overline{F} \mid \varphi_i(e) \in E'\}$

- $F_i := \{e \in E(T_i) \cap F \mid \varphi_i(e) \in E(T) \setminus E'\}$
- $S_i := \{e \in E(T_i) \cap \overline{F} \mid \varphi_i(e) \in E(T) \setminus E'\}$

Similarly to Section 5.3.5 we define the parameters of Problem (5.2). Since we consider arbitrary values of α in our analysis instead of the threshold values of the given instance, these parameters are defined slightly differently. The key difference is that in Section 5.3.5 we defined the β and γ values according to the precise threshold values that we also used in the analysis. Since we do not consider the threshold values in the analysis of this section, we define the parameters according to the closest upper and lower α values to the threshold value instead.

More formally, for $0 \leq i \leq N$, we let $E_i^{\bar{F}}$ (E_i^F, resp.) be the set of edges in E' ($E(T)-E'$, resp.) that have threshold values between α_i and α_{i+1}. That is, $E_i^{\bar{F}} := \{e \in E' \mid \alpha_i < \alpha_e \leq \alpha_{i+1}\}$ and $E_i^F := \{e \in E(T)-E' \mid \alpha_i < \alpha_e \leq \alpha_{i+1}\}$. For $0 \leq i \leq N$, we let $\beta_i = w(E_i^{\bar{F}})/\mathrm{OPT(I)}$ and $\gamma_i = w(E_i^F)/\mathrm{OPT(I)}$ be the fraction of the weight of the optimal solution that is contributed by the edges in $E_i^{\bar{F}}$ (resp., E_i^F). Finally, let $\xi \in [0,1]$ be the fraction of the weight of the optimal solution that is not contributed by the tree T; e.g., $\xi := \frac{w(Z^*)-w(T)}{\mathrm{OPT(I)}}$. The following properties of β_i, γ_i, $0 \leq i \leq N$, are readily verified:

1. $\beta_1, \beta_2, \dots \beta_N, \gamma_1, \gamma_2, \dots \gamma_N, \xi \in [0,1]$,
2. $\sum_{j=0}^N \beta_j = \frac{w(E')}{\mathrm{OPT(I)}}$,
3. $\sum_{j=0}^N \gamma_j = \frac{w(T-E')}{\mathrm{OPT(I)}}$, and
4. $\xi = 1 - \sum_{j=0}^N \beta_j - \sum_{j=0}^N \gamma_j$.

With all these parameters we can now bound the cost of the solutions computed by Algorithms A, B, and C. We start with the solution Z_i^C returned by Algorithm C.

Lemma 5.22. *Suppose we run Algorithm 4 with threshold values $W = \{\alpha_i\}_{0 \leq i \leq N}$. For $1 \leq i \leq N$, let Z_i^C be the solution computed by*

Algorithm C with scaling factor α_i in Algorithm 4. Then

$$w(Z_i^C) \leq \Bigg(1 + \sum_{j=0}^{i-1} (c_{2\mathrm{EC}} - 1 + c_{2\mathrm{EC}}\alpha_{j+1})\beta_j + (c_{2\mathrm{EC}} - 1) \cdot \sum_{j=0}^{N} \gamma_j \\ + \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j} + (c_{2\mathrm{EC}} - 1) \cdot \xi \Bigg) \cdot \mathrm{OPT(I)} \ .$$

Proof. The proof is similar to the one of Lemma 5.18. Let T_i^S be the safe edges of the tree T_i and let $\varphi_i : E(T_i) \to E(T)$ be an α_i-monotone exchange bijection, where T is a spanning tree of the optimal solution Z^* to the instance $(G, w, \overline{F})$.

By contracting each edge of T_i^S in G we obtain the graph $G_i^S \coloneqq G/E(T_i^S)$. Algorithm C then computes a $c_{2\mathrm{EC}}$-approximate solution to the instance (G_i^S) of 2-ECSS. The next claim is identical to Claim 1 in the proof of Lemma 5.18.

Claim 1. *The set*

$$Y_i \coloneqq \left(\bigcup_{e \in E' \setminus \varphi_i(E(T_i^S))} \{e, \varphi_i^{-1}(e)\} \right) \cup \bigcup_{1 \leq j \leq r} E(C_j)$$

*of edges is a feasible solution to the 2-*ECSS *instance (G_i^S).*

We now bound the cost of Z_i^C. The algorithm then returns in polynomial time a solution Z_i^C of cost at most $w(Z_i^C) \leq w(T_i^S) + c_{2\mathrm{EC}} w(Y_i)$. The next claim is very similar to Claim 2 of Lemma 5.18. The only difference is that we consider arbitrary values of α in our analysis instead of the threshold values of the given instance. Thus we need to bound the cost of $w(e)$ in terms of the largest $\alpha \in W$ such that $\alpha \leq \alpha_e$. The rest of the proof is analogous. We bound the cost of each edge of T_i^S as follows.

Claim 2. *Let $e \in E(T_i^S)$ and let α_e be its threshold value. Then we have*

$$w(e) \leq \begin{cases} \frac{1}{\alpha_j} \cdot w(\varphi_i(e)) & \text{if } \varphi_i(e) \notin E', \text{ and} \\ w(\varphi_i(e)) & \text{otherwise} \ , \end{cases}$$

where α_j is the largest $\alpha \in W$ satisfying $\alpha \leq \alpha_e$.

Using the above claim we now bound the cost of the edges in T_i^S. For $1 \leq i \leq N$ we have

$$\begin{aligned}
w(T_i^S) &= \sum_{e \in T_i^S} w(e) = \sum_{e \in T_i^S} \sum_{\arg\max_j\{\alpha_e \geq \alpha_j\}} w(e) \\
&= \sum_{e \in T_i^S : \varphi_i(e) \in E'} \sum_{\arg\max_j\{\alpha_e \geq \alpha_j\}} w(e) + \sum_{e \in T_i^S : \varphi_i(e) \notin E'} \sum_{\arg\max_j\{\alpha_e \geq \alpha_j\}} w(e) \\
&\leq \sum_{e \in T_i^S : \varphi_i(e) \in E'} \sum_{\arg\max_j\{\alpha_e \geq \alpha_j\}} w(\varphi_i(e)) \\
&\quad + \sum_{e \in T_i^S : \varphi_i(e) \notin E'} \sum_{\arg\max_j\{\alpha_e \geq \alpha_j\}} \frac{1}{\alpha_j} \cdot w(\varphi_i(e)) \\
&\leq \left(\sum_{j=i}^{N} \beta_j + \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j} \right) \cdot \mathrm{OPT(I)} \\
&= \left(1 - \sum_{j=0}^{i-1} \beta_j - \sum_{j=0}^{N} \gamma_j - \xi + \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j} \right) \cdot \mathrm{OPT(I)} \ ,
\end{aligned}$$

where the first inequality follows from Claim 2 and the last equality holds due to $\xi + \sum_{j=1}^{N} \beta_j + \sum_{j=1}^{N} \gamma_j = 1$.

Furthermore, we bound the cost of the solution Y_i to the 2-ECSS instance (G_i^S) as follows.

Claim 3.

$$w(Y_i) \leq \left(\sum_{j=0}^{i-1} (1 + \alpha_{j+1}) \beta_j + \sum_{j=0}^{N} \gamma_j + \xi \right) \cdot \mathrm{OPT(I)} \ .$$

The proof of Claim 3 is analogous to the proof of Claim 3 in Lemma 5.18. However, for each $e \in E' \setminus \varphi_i(E(T_i^S))$, we have $w(\varphi_i^{-1}(e)) \leq \alpha_{j+1} w(e)$. Here, α_j is the largest $\alpha_j \in W$ satisfying $\alpha_j \leq \alpha_e$. Thus we can bound the cost of $\{\varphi_i^{-1}(e) \mid e \in E' \setminus \varphi_i(E(T_i^S))\}$ by $\sum_{j=i}^{i-1} \alpha_{j+1} \beta_j$.

Finally, since the algorithm computes a $c_{2\mathrm{EC}}$-approximate solution to the

2-ECSS instance, we have

$$\begin{aligned} w(Z_i^C) &\le w(T_i^S) + c_{2\mathrm{EC}} \cdot w(Y_i) \\ &\le \Bigg(\left(1 - \sum_{j=0}^{i-1} \beta_j - \sum_{j=0}^{N} \gamma_j - \xi + \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j}\right) \\ &\quad + \left(\sum_{j=1}^{i} (c_{2\mathrm{EC}} + c_{2\mathrm{EC}} \cdot \alpha_{j+1})\beta_j + \sum_{j=0}^{N} c_{2\mathrm{EC}}\gamma_j + c_{2\mathrm{EC}} \cdot \xi \right) \Bigg) \cdot \mathrm{OPT(I)} \\ &\le \Bigg(1 + \sum_{j=0}^{i-1} (c_{2\mathrm{EC}} + c_{2\mathrm{EC}} \cdot \alpha_{j+1} - 1)\beta_j + (c_{2\mathrm{EC}} - 1) \cdot \sum_{j=0}^{N} \gamma_j \\ &\quad + \sum_{j=i}^{N} \frac{1}{\alpha_j}\gamma_j + (c_{2\mathrm{EC}} - 1) \cdot \xi \Bigg) \cdot \mathrm{OPT(I)} \ , \end{aligned}$$

which concludes the proof of Lemma 5.22. ■

Next, we bound the cost of the solution Z^A computed by Algorithm A.

Lemma 5.23. *Suppose we run Algorithm 4 with threshold values* $W = \{\alpha_i\}_{0 \le i \le N}$. *Let* Z^A *be the solution computed by Algorithm A in Algorithm 4. Then*

$$w(Z^A) \le \left(c_{2\mathrm{EC}} + c_{2\mathrm{EC}} \cdot \sum_{j=0}^{N} \alpha_{j+1}\beta_j \right) \cdot \mathrm{OPT(I)} \ .$$

Proof. The proof is similar to the one of Lemma 5.19. The only difference is an index shift in the bound of the following claim, compared to Claim 1 of Lemma 5.19.

Claim 1. $Y_A \coloneqq Z^* \cup \{\varphi_N^{-1}(e)) \mid e \in E'\}$ *is a feasible solution to the 2-ECSS instance of cost at most*

$$w(Y_A) \le \left(1 + \sum_{j=0}^{N} \alpha_{j+1}\beta_j \right) \cdot \mathrm{OPT(I)} \ .$$

Proof. The proof for feasibility can be found in Claim 1 of Lemma 5.19.

It remains to bound the cost of Y_A. We partition Y_A into two disjoint sets

$$Y_A = Z^* \cup X_1 \ ,$$

where $X_1 = \bigcup_{e \in E'} \varphi_N^{-1}(e)$ and bound the cost of each part individually. Clearly we have $w(Z^*) = \mathrm{OPT(I)}$. To bound the cost of X_1, observe that for some

$\varphi_N^{-1}(e) \in X_1$ we have $w(\varphi_N^{-1}(e)) \leq \alpha_{j+1} w(e)$, where α_j is the largest $\alpha \in W$ satisfying $\alpha_j \leq \alpha_e$. We can bound the cost of X_1 by

$$\begin{aligned} w(X_1) = \sum_{\varphi_N^{-1}(e) \in X_1} w(e) &= \sum_{\varphi_N^{-1}(e) \in X_1} \sum_{\arg\max_j \{\alpha_e \geq \alpha_j\}} w(\varphi_N^{-1}(e)) \\ &\leq \sum_{\varphi_N^{-1}(e) \in X_1} \sum_{\arg\max_j \{\alpha_e \geq \alpha_j\}} \alpha_{j+1} w(e) = \sum_{j=0}^{N} \alpha_{j+1} \beta_j \cdot \mathrm{OPT(I)} \ . \end{aligned}$$

Combining both bounds we obtain

$$w(Y_A) \leq w(Z^*) + w(X_1) \leq \left(1 + \sum_{j=0}^{N} \alpha_{j+1} \beta_j\right) \cdot \mathrm{OPT(I)} \ .$$

This proves Claim 1. □

Since the algorithm computes a c_{2EC}-approximation, we have that

$$w(Z^A) \leq c_{\mathrm{2EC}} w(Y_A) \leq \left(c_{\mathrm{2EC}} + c_{\mathrm{2EC}} \cdot \sum_{j=0}^{N} \alpha_{j+1} \beta_j\right) \cdot \mathrm{OPT(I)} \ .$$

This proves Lemma 5.23. ■

Finally, we bound the cost of the solution output by Algorithm B.

Lemma 5.24. *Suppose we run Algorithm 4 with threshold values $W = \{\alpha_i\}_{0 \leq i \leq N}$. For $1 \leq i \leq N$ let Z_i^B be the solution computed by Algorithm B with scaling factor α_i in Algorithm 4. Then*

$$\begin{aligned} w(Z_i^B) \leq \Bigg(1 &+ \sum_{j=0}^{i-1} (c_{\mathrm{TAP}} - 1 + \alpha_{j+1}) \beta_j + (c_{\mathrm{TAP}} - 1) \cdot \sum_{j=0}^{N} \gamma_j \\ &+ \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j} + \sum_{j=0}^{i-1} \gamma_j + (c_{\mathrm{TAP}} - 1) \cdot \xi \Bigg) \cdot \mathrm{OPT(I)} \ . \end{aligned}$$

Proof. Let T_i^S be the safe edges of the tree T_i and let $\varphi_i : E(T_i) \to E(T)$ be an α_i-monotone exchange bijection, where T is a spanning tree of the optimal solution Z^* to the instance $(G, w, \overline{F})$.

By contracting each edge of T_i^S in T_i and G we obtain the tree T_i' and the graph $G_i^S := G/E(T_i^S)$. Algorithm B computes a c_{TAP}-approximate solution to the instance (G_i^S, T_i') of WTAP.

Claim 1. *The set*

$$Y_i := (E' \setminus \varphi_i(E(T_i^S))) \cup \bigcup_{1 \le j \le r} E(C_j)$$

of edges is a feasible solution to the WTAP *instance* $(G_i^S.T_i')$.

Proof. Clearly (V, Y_i) is a connected graph. It remains to argue that each edge $e \in Y_i$ is contained in some cycle. This is certainly true for each edge e of a component C_j, $1 \le j \le r$. It remains to show that the edges in $E' \setminus \varphi_i(E(T_i^S))$ are contained in some cycle of G_i^S. Since φ_i is an exchange bijection, an edge $e \in E' \setminus \varphi_i(E(T_i^S))$ and its preimage $\varphi^{-1}(e)$ are on a cycle in $E(T) \cup \{e\}$. This concludes the proof of Claim 1. $\square$

We now bound the cost of Z_i^B. By Claim 1 we have that $T_i \cup Y_i$ is a feasible solution to the FGC instance $(G, w, \overline{F})$. The algorithm then returns in polynomial time a solution Z_i^B of cost at most $w(Z_i^B) \le w(T_i) + c_{\text{TAP}} w(Y_i)$. We first bound the cost of each edge of T_i as follows.

Claim 2. *Let $e \in E(T_i)$ and let α_e be its threshold value. Then we have*

$$w(e) \le \begin{cases} \frac{1}{\alpha_j} \cdot w(\varphi_i(e)) & \text{if } \varphi_i(e) \notin E', \\ \alpha_{j+1} \cdot w(\varphi_i(e)) & \text{if } \varphi_i(e) \in E' \text{ and } e \in F, \text{ and} \\ w(\varphi_i(e)) & \text{otherwise }, \end{cases}$$

where α_j is the largest $\alpha \in W$ satisfying $\alpha \le \alpha_e$.

Proof. It remains to prove $w(e) \le \alpha_{j+1} w(\varphi_i(e))$ if $\varphi_i(e) \in E'$ and $e \in F$ since the other part is proven in Claim 2 of Lemma 5.22. Since e is an unsafe edge and $\varphi_i(e) \in E'$, it holds that $w(e) \le \alpha_e w(\varphi_i(e))$. Since e is contained in T_j but not contained in T_{j+1} (by the definition of j), we have that $\alpha_e \le \alpha_{j+1}$. This proves Claim 2. $\square$

Recall that by Lemma 5.22 we have

$$w(T_i^S) \le \left(\sum_{j=i}^{N} \beta_j + \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j} \right) \cdot \text{OPT(I)} .$$

According to Claim 2 we obtain for $1 \le i \le N$ that

$$\begin{aligned}
w(T_i \cap F) &= \sum_{e \in T_i \cap F} w(e) = \sum_{e \in T_i \cap F} \sum_{\arg\max_j \{\alpha_e \ge \alpha_j\}} w(e) \\
&= \sum_{e \in T_i \cap F : \varphi_i(e) \in E'} \sum_{\arg\max_j \{\alpha_e \ge \alpha_j\}} w(e) \\
&\quad + \sum_{e \in T_i \cap F : \varphi_i(e) \notin E'} \sum_{\arg\max_j \{\alpha_e \ge \alpha_j\}} w(e) \\
&\le \sum_{e \in T_i \cap F : \varphi_i(e) \in E'} \sum_{\arg\max_j \{\alpha_e \ge \alpha_j\}} \alpha_{j+1} w(\varphi_i(e)) \\
&\quad + \sum_{e \in T_i \cap F : \varphi_i(e) \notin E'} \sum_{\arg\max_j \{\alpha_e \ge \alpha_j\}} w(\varphi_i(e)) \\
&\le \left(\sum_{j=0}^{i-1} \alpha_{j+1} \beta_j + \sum_{j=0}^{i-1} \gamma_j \right) \cdot \mathrm{OPT(I)} \ .
\end{aligned}$$

Thus, we obtain

$$\begin{aligned}
w(T_i) &= w(T_i^S) + w(T_i \cap F) \\
&\le \left(\sum_{j=0}^{i-1} \alpha_{j+1} \beta_j + \sum_{j=i}^{N} \beta_j + \sum_{j=0}^{i-1} \gamma_j + \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j} \right) \cdot \mathrm{OPT(I)} \\
&= \left(1 + \sum_{j=0}^{i-1} (\alpha_{j+1} - 1) \beta_j - \sum_{j=0}^{N} \gamma_j - \xi + \sum_{j=0}^{i-1} \gamma_j + \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j} \right) \cdot \mathrm{OPT(I)} \ ,
\end{aligned}$$

where the last equality holds due to $\xi + \sum_{j=0}^{N} \beta_j + \sum_{j=0}^{N} \gamma_j = 1$. We additionally bound the cost of Y_i.

Claim 3.

$$w(Y_i) \le \left(\sum_{j=0}^{i-1} \beta_j + \sum_{j=0}^{N} \gamma_j + \xi \right) \cdot \mathrm{OPT(I)}$$

Proof. We need to bound the cost of

$$Y_i = \big(E' \setminus \varphi_i(E(T_i^S))\big) \cup \bigcup_{1 \le j \le r} E(C_j) \ .$$

We first bound the cost of $E' \setminus \varphi_i(E(T_i^S))$. According to the definition of $\beta_0, \beta_1, \dots, \beta_N$ we can bound $\bigcup_{e \in E' \setminus \varphi_i(E(T_i^S))} w(e)$ by $\sum_{j=1}^{i-1} \beta_j$. Additionally, we bound $\bigcup_{1\leq j\leq r} w(E(C_j))$ by $(\xi + \sum_{j=1}^{N}\gamma_j)\cdot \mathrm{OPT}$. Putting things together, we obtain

$$w(Y_i) \leq \left(\sum_{j=0}^{i-1}\beta_j + \sum_{j=0}^{N}\gamma_j + \xi\right)\cdot \mathrm{OPT(I)}\ .$$

This concludes the proof of Claim 3. □

Finally, since the algorithm computes a c_{TAP}-approximate solution for the WTAP instance, we have

$$\begin{aligned}
w(Z_i^B) &\leq w(T_i) + c_{\mathrm{TAP}}\cdot w(Y_i)\\
&\leq \Bigg(1 + \sum_{j=0}^{i-1}(\alpha_{j+1}-1)\beta_j - \sum_{j=0}^{N}\gamma_j - \xi + \sum_{j=0}^{i-1}\gamma_j + \sum_{j=i}^{N}\frac{\gamma_j}{\alpha_j}\\
&\qquad + \sum_{j=0}^{i-1}\beta_j + \sum_{j=0}^{N}\gamma_j + \xi\Bigg)\cdot \mathrm{OPT(I)}\\
&\leq \Bigg(1 + \sum_{j=0}^{i-1}(c_{\mathrm{TAP}} - 1 + \alpha_{j+1})\beta_j + (c_{\mathrm{TAP}}-1)\cdot\sum_{j=0}^{N}\gamma_j\\
&\qquad + \sum_{j=i}^{N}\frac{\gamma_j}{\alpha_j} + \sum_{j=0}^{i-1}\gamma_j + (c_{\mathrm{TAP}}-1)\cdot\xi\Bigg)\cdot \mathrm{OPT(I)}\ ,
\end{aligned}$$

which concludes the proof of Lemma 5.24. ■

With a similar argument as in Section 5.3.5 we can argue that $\xi = 0$. For the sake of completeness, we state it here again. Since $c_{2\mathrm{EC}}, c_{\mathrm{TAP}} \geq 1$, each $\beta_j, \gamma_j, j \in [N]$ as well as ξ has a positive coefficient in the bounds of the Lemmas 5.22, 5.23 and 5.24. Moreover, since we maximize over $\beta_1, \beta_2, \ldots, \beta_N$, $\gamma_1, \gamma_2, \ldots, \gamma_N$, and ξ, we may assume $\xi = 0$. To see this, suppose we have an optimal choice of the variables, where $\xi > 0$. Then, consider the following new variables. Let $\beta'_j = \beta_j$ for $j \in [N], \gamma'_j = \gamma_j$ for $j \in [N-1]$, $\gamma'_N = \gamma_N + \xi$, and $\xi' = 0$. Now observe that the value of the minimum over $w(Z^A)$ as well as $w(Z_i^C)$ and $w(Z_i^B), i \in [N]$ for the new variables is at least as large as for the old ones. Thus, we can assume that $\xi = 0$ and have

$$\sum_{j=1}^{N}\beta_j + \sum_{j=1}^{N}\gamma_j = 1\ .$$

Hence we found suitable functions for $f^A(\cdot)$, $f_i^B(\cdot)$, $f_i^C(\cdot)$. These are given in Lemmas 5.23, 5.24 and 5.22, respectively (with ξ set to 0).

Next, using these bounds, we will provide a computational proof of Theorem 5.3. Note that even though we use the optimal threshold values in our algorithm, in our analysis we cannot use instance specific scaling factors. However, since we minimize over the number and choice of scaling factors, we can relax the minimization problem by simply choosing some "good" scaling factors. Eventually, we can turn the resulting max min problem into a maximization problem. This is presented in the following proof.

Computational proof of Theorem 5.3. We choose $N = 58$ and let $c_{\mathrm{TAP}} = 1.5$ and $c_{\mathrm{2EC}} = 2$ and

$$\begin{aligned}(\alpha_i)_{0\le i\le N+1} = {}& 0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.51, 0.52, 0.53,\\ & 0.54, 0.55, 0.56, 0.57, 0.58, 0.59, 0.60, 0.61, 0.62, 0.63, 0.64, 0.65,\\ & 0.67, 0.68, 0.69, 0.7, 0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78,\\ & 0.79, 0.8, 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9,\\ & 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1\ .\end{aligned}$$

With these choices we obtain the following optimization problem from (5.2).

$$\begin{aligned}
\text{maximize } & z \\
\text{subject to } & z \le c_{\mathrm{2EC}} + c_{\mathrm{2EC}} \cdot \sum_{j=0}^{N} \alpha_{j+1}\beta_j \\
& z \le 1 + \sum_{j=0}^{i-1} (c_{\mathrm{TAP}} - 1 + \alpha_{j+1})\beta_j \\
& \qquad + (c_{\mathrm{TAP}} - 1) \cdot \sum_{j=0}^{N} \gamma_j + \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j} + \sum_{j=0}^{i-1} \gamma_j \qquad \text{for } 1 \le i \le N \\
& z \le 1 + \sum_{j=0}^{i-1} (c_{\mathrm{2EC}} - 1 + c_{\mathrm{2EC}}\alpha_{j+1})\beta_j \\
& \qquad + (c_{\mathrm{2EC}} - 1) \cdot \sum_{j=0}^{N} \gamma_j + \sum_{j=i}^{N} \frac{\gamma_j}{\alpha_j} \qquad \text{for } 1 \le i \le N \\
& \sum_{j=1}^{N} \beta_j + \gamma_j = 1 \\
& \beta_j, \gamma_j \in [0,1] \qquad \text{for } 1 \le j \le N
\end{aligned} \tag{5.6}$$

The constraints are given by the bounds from Lemmas 5.22 – 5.24. The solution of the problem is an upper bound on the approximation ratio of Algorithm 4 due to Lemma 5.6. We obtain a computational upper bound of 2.404 and lower bound of 2.4035 on the optimal value of the non-linear program above using the NLP solver baron [The19]. ■

Note that we did not investigate the complexity of Problem 5.6, since we were mainly interested in its solution. However, for a larger number N of scaling factors, the time it took BARON to solve Problem 5.6 grew fast. Hence, it might be true that BARON does not solve this problem in polynomial-time.

However, we are able to solve instances with about 60 scaling factors in roughly 30 minutes on a standard computer. As one might expect, when refining the choice of scaling factors, the solution became better, but converged very fast to roughly 2.4. It would be interesting to find an exact upper bound similar to the one presented in Section 5.3.5.

5.4 Approximation Hardness on Transversal Matroids

Our main technical tools in the analysis of our approximation algorithm for FGC are exchange bijections, which are based on a matroid basis exchange argument. So it is natural to ask whether our results can be transferred to a matroid setting entirely. Let us consider a generalization of FGC as follows.

Problem 5.25 (Flexible Matroid Basis)**.** Given a matroid M on a ground set X, weights $w \in \mathbb{Z}^X_{\geq 0}$, and unsafe items $F \subseteq X$, the task is to find a minimum-weight set of elements $E \subseteq X$ such that for each $f \in F$, the set $E - f$ contains a basis of M.

Observe that for graphic matroids this problem corresponds to FGC. Note also that α-MSTs and their threshold properties as well as α-monotone exchange bijections (Section 5.3.3) generalize in a natural way to matroids by replacing "spanning tree" by "matroid basis" in the respective lemmas. We show that despite these promising observations, under some standard complexity assumption, we cannot hope for a polynomial-time constant-factor approximation algorithm for Flexible Matroid Basis. Let $G = (U, V, E)$ be a bipartite graph and let $\mathcal{I} := \{F \subseteq U \mid \text{there is a matching } M \text{ of } G \text{ that covers } F\}$. Then $\mathcal{I}$ is the set of independent sets of a matroid. Matroids that can be obtained in this manner are called *transversal matroids*, see for instance [Oxl06] for an introduction to the theory of transversal matroids. The next theorem shows that Flexible Matroid Basis on transversal matroids is as hard to approximate as Set Cover.

Theorem 5.26. Flexible Matroid Basis *on transversal matroids admits no polynomial-time* $(1-\varepsilon)\log|X|$*-approximation algorithm for any* $\varepsilon > 0$ *unless* $\mathsf{P} = \mathsf{NP}$.

Proof. Consider an instance $\mathrm{I} = (U, \mathcal{S})$ of Set Cover with ground set $U = \{u_1, u_2, \ldots, u_n\}$ and a family of subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$. We construct an instance $\mathrm{I}' := (M(G), w, F)$ of Flexible Matroid Basis as follows. Let $G = (A, B, E)$ be a bipartite graph given by

$$\begin{aligned} A :=& \{u_1, u_2, \ldots, u_n\} \cup \{S_1, S_2, \ldots, S_m\} \ , \\ B :=& \{v_1, v_2, \ldots, v_n\} \ , \\ E :=& \{S_i v_j \mid u_j \in S_i,\ 1 \le i \le m,\ 1 \le j \le n\} \ . \end{aligned}$$

Let $M(G)$ be the transversal matroid that arises from the subsets of A that are covered by a matching of G. Clearly, the set U is a basis of $M(G)$. Let $F := A \setminus U$ be the set of unsafe items and let the weights $w \in \mathbb{Z}^A_{\ge 0}$ be given by $w(e) = 0$ if $e \in U$ and $w(e) = 1$ otherwise. This concludes the construction of the instance $\mathrm{I}' = (M(G), w, F)$ of Flexible Matroid Basis.

Let $Z^* \subseteq A$ be an optimal solution to the instance I'. We may assume that Z^* contains U. We claim that $Z^* \setminus U$ is an optimal solution to the instance I. First, suppose that $Z^* \setminus U$ is not a cover. Then there is an item $u \in U$ that is not covered by $Z^* \setminus U$. But then $Z^* - u$ does not contain a basis of $M(G)$. Now suppose that $Z^* \setminus U$ is not optimal for I. Then there is a family $C \subseteq \mathcal{S}$ of sets that cover U which is cheaper than $Z^* \setminus U$. But then $U \cup C$ is a cheaper solution to I' than Z^*, contradicting the optimality of Z^*. We conclude that the values of optimal solutions to I and I' are the same. Therefore, any polynomial-time ρ-approximation algorithm for Robust Matroid Basis gives a polynomial-time ρ-approximation algorithm for Set

Cover. Hence, the approximation hardness result for Set Cover by Dinur and Steurer [DS14] implies that there is no polynomial-time $(1 - \varepsilon)\log|X|$-approximation algorithm for Robust Spanning Tree unless $\mathsf{P} = \mathsf{NP}$. ■

By a theorem of Piff and Welsh [PW70], any transversal matroid is representable over any sufficiently large field. Therefore, Flexible Matroid Basis is Set Cover-hard to approximate on vector matroids. Note that in the proof of Theorem 5.26, we essentially show approximation hardness of a generalization of WTAP to matroids. It is an interesting open question whether this generalization admits constant-factor approximation algorithms for subclasses of transversal or vector matroids, for instance, regular, binary, and bicircular matroids.

5.5 Conclusion

We studied Flexible Graph Connectivity, a problem that encapsulates the problems Minimum Spanning Tree, 2-Edge Connected Spanning Subgraph, and many more. We provided an approximation algorithm that uses approximation algorithms for 2-ECSS and WTAP as a black-box. Using the current best known approximation factors for these problems, we obtained approximation factors of 2.523 for general instances and 2.404 for bounded-weight instances.

We are not aware of examples that actually reach these upper bounds. An interesting open question is whether there are such examples or to show that the approximation guarantees are indeed better than the ones presented here.

Of course, the main open problem is whether one can achieve an approximation factor close to 2 (current best approximation factor for 2-ECSS). One way to tackle this problem is to “open the box“ instead of simply using the approximation algorithms for 2-ECSS and WTAP as a black-box.

Chapter 6

Conclusion and Outlook

In this thesis we investigated three bulk-robust combinatorial optimization problems on graphs, namely the problems ROBUST MATCHING AUGMENTATION, BULK-ROBUST k-DISJOINT PATHS and FLEXIBLE GRAPH CONNECTIVITY. We presented efficient exact and approximation algorithms as well as FPT and XP algorithms, complemented by hardness and inapproximability results. In our opinion, the case of single-edge failure is well understood for most questions that sprang up in the analysis of the three problems. Clearly, as indicated in Chapter 4 the main open question is whether DIRECTED 1-ROBUST k-DISJOINT PATHS admits a polynomial-time algorithm or not. Another interesting question is whether it is possible to obtain a polynomial-time approximation algorithm for FLEXIBLE GRAPH CONNECTIVITY with guarantee 2.

A possible further direction for research could be to consider edge failure of constant size. Note that if we consider edge failure of 'large' size the bulk-robust counterpart of almost any combinatorial optimization problem becomes as hard as SET COVER [ASZ15]. However, instead of edge failure one could also consider vertex failure and the corresponding bulk-robust version. Besides the work of Bindewald [Bin18] who considered the assignment problem with vertex failure, we are not aware of any other work considering vertex failure.

Another direction for further research could be to investigate bulk-robust counterparts of other combinatorial optimization problems. So far, only bulk-robust counterparts of polynomial-time solvable problems have been considered. For example, the problem STEINER TREE could be a promising problem as it generalizes the problem MINIMUM SPANNING TREE. Therefore, the bulk-robust version of STEINER TREE would generalize FLEXIBLE GRAPH CONNECTIVITY and it would be interesting to see if the methods presented in Chapter 5 could also be applied to this problem.

Bibliography

[ABM16] David Adjiashvili, Viktor Bindewald, and Dennis Michaels. Robust Assignments via Ear Decompositions and Randomized Rounding. In *43rd International Colloquium on Automata, Languages, and Programming*, volume 55, pages 71:1–71:14, Dagstuhl, Germany, 2016.

[ABM17] David Adjiashvili, Viktor Bindewald, and Dennis Michaels. Robust assignments with vulnerable nodes. *arXiv preprint arXiv:1703.06074*, 2017.

[ABV05a] Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Approximation complexity of min-max (regret) versions of shortest path, spanning tree, and knapsack. In *European Symposium on Algorithms*, pages 862–873. Springer, 2005.

[ABV05b] Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Complexity of the min–max and min–max regret assignment problems. *Operations research letters*, 33(6):634–640, 2005.

[ABV05c] Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Pseudo-polynomial algorithms for min-max and min-max regret problems. *5th ISORA*, pages 171–178, 2005.

[ABV08] Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Complexity of the min–max (regret) versions of min cut problems. *Discrete optimization*, 5(1):66–73, 2008.

[ABWZ14] David Adjiashvili, Sandro Bosio, Robert Weismantel, and Rico Zenklusen. Time-expanded packings. In *International Colloquium *on Automata, Languages, and Programming*, pages 64–76. Springer Berlin Heidelberg, 2014.

[Adj12] David Adjiashvili. *Structural robustness in combinatorial optimization.* PhD thesis, ETH Zurich, 2012.

[Adj18] David Adjiashvili. Beating approximation factor two for weighted tree augmentation with bounded costs. *ACM Transactions on Algorithms (TALG)*, 15(2):19, 2018.

[AHM20] David Adjiashvili, Felix Hommelsheim, and Moritz Mühlenthaler. Flexible graph connectivity. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 13–26. Springer, 2020.

[ASZ15] David Adjiashvili, Sebastian Stiller, and Rico Zenklusen. Bulk-robust combinatorial optimization. *Mathematical Programming*, 149(1-2):361–390, 2015.

[AZ06] Amitai Armon and Uri Zwick. Multicriteria global minimum cuts. *Algorithmica*, 46(1):15–26, 2006.

[BBI14] Frank Baumann, Christoph Buchheim, and Anna Ilyina. A lagrangean decomposition approach for robust combinatorial optimization. *Optimization Online*, 2014.

[BBM$^+$13] Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Approximation algorithms for spanner problems and directed steiner forest. *Information and Computation*, 222:93–107, 2013. 38th International Colloquium on Automata, Languages and Programming (ICALP 2011).

[BCKN15] Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.

[Bel58] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[BGA20] Jarosław Byrka, Fabrizio Grandoni, and Afrouz Jabal Ameli. Breaching the 2-approximation barrier for connectivity augmentation: a reduction to steiner tree. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 815–825, 2020.

[BGRS13] Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *Journal of the ACM (JACM)*, 60(1):1–33, 2013.

[BHT12] Andreas Björklund, Thore Husfeldt, and Nina Taslaman. Shortest cycle through specified elements. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1747–1753. SIAM, 2012.

[Bin18] Viktor Bindewald. *Bulk-robust assignment problems: hardness, approximability and algorithms*. PhD thesis, Fakultät für Mathematik, TU Dortmund University, 2018.

[BK18] Christoph Buchheim and Jannis Kurtz. Robust combinatorial optimization under convex and discrete cost uncertainty. *EURO Journal on Computational Optimization*, 6(3):211–238, 2018.

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[Buc20] Christoph Buchheim. A note on the nonexistence of oracle-polynomial algorithms for robust combinatorial optimization. *Discrete Applied Mathematics*, 285:591 – 593, 2020.

[CCC$^+$99] Moses Charikar, Chandra Chekuri, To-yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *Journal of Algorithms*, 33(1):73–91, 1999.

[CDG$^+$19] J. Cheriyan, J. Dippel, Fabrizio Grandoni, Arindam Khan, and V. Narayan. The matching augmentation problem: a $\frac{7}{4}$-approximation algorithm. *Mathematical Programming*, 04 2019.

[CEGS08] Chandra Chekuri, Guy Even, Anupam Gupta, and Danny Segev. Set connectivity problems in undirected graphs and the directed steiner network problem. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 532–541. Society for Industrial and Applied Mathematics, 2008.

[CFK$^+$15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[CLNV14] Joseph Cheriyan, Bundit Laekhanukit, Guyslain Naves, and Adrian Vetta. Approximating rooted steiner networks. *ACM Transactions on Algorithms (TALG)*, 11(2):1–22, 2014.

[Coo71] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[CT00] Joseph Cheriyan and Ramakrishna Thurimella. Approximating minimum-size k-connected spanning subgraphs via matching. *SIAM Journal on Computing*, 30(2):528–560, 2000.

[D$^+$59] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[DGRS05] Kedar Dhamdhere, Vineet Goyal, R Ravi, and Mohit Singh. How to pay, come what may: Approximation algorithms for demand-robust covering problems. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 367–376. IEEE, 2005.

[Die12] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[DKL76] E.A. Dinits, A.V. Karzanov, and M.V. Lomonosov. On the structure of a family of minimal weighted cuts in a graph. *Studies in Discrete Optimization*, pages 290–306, 1976.

[DMP$^+$15] Mitre C. Dourado, Dirk Meierling, Lucia D. Penso, Dieter Rautenbach, Fabio Protti, and Aline Ribeiro de Almeida. Robust recoverable perfect matchings. *Networks*, 66(3):210–213, 2015.

[DS14] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, pages 624–633. ACM, 2014.

[DW71] Stuart E Dreyfus and Robert A Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[ET76] Kapali P. Eswaran and Robert E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976.

[FF09] Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. In *Classic papers in combinatorics*, pages 243–248. Springer, 2009.

[FGKS18] Samuel Fiorini, Martin Groß, Jochen Könemann, and Laura Sanità. Approximating weighted tree augmentation via Chvátal-gomory cuts. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 817–831. Society for Industrial and Applied Mathematics, 2018.

[FJ56] Lester R Ford Jr. Network flow theory. Technical report, Rand Corp Santa Monica Ca, 1956.

[FJ81] Greg N Frederickson and Joseph JáJá. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270–283, 1981.

[FJ15] András Frank and Tibor Jordán. Graph connectivity augmentation. In *Handbook of Graph Theory, Combinatorial Optimization, and Algorithms*, chapter 14, pages 313–346. CRC Press, 2015.

[FJMM07] Uriel Feige, Kamal Jain, Mohammad Mahdian, and Vahab Mirrokni. Robust combinatorial optimization with exponential scenarios. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 439–453. Springer, 2007.

[FKN09] Moran Feldman, Guy Kortsarz, and Zeev Nutov. Improved approximating algorithms for directed steiner forest. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 922–931. Society for Industrial and Applied Mathematics, 2009.

[FR06] Jon Feldman and Matthias Ruhl. The directed steiner network problem is tractable for a constant number of terminals. *SIAM Journal on Computing*, 36(2):543–561, 2006.

[GG12] Harold N Gabow and Suzanne R Gallagher. Iterated rounding algorithms for the smallest k-edge connected spanning subgraph. *SIAM Journal on Computing*, 41(1):61–103, 2012.

[GGAS19] Waldo Gálvez, Fabrizio Grandoni, Afrouz Jabal Ameli, and Krzysztof Sornat. On the cycle augmentation problem: Hardness and approximation algorithms. In *Workshop on Approximation and Online Algorithms (WAOA)*, 2019.

[GGP$^+$94] Michel X Goemans, Andrew V Goldberg, Serge A Plotkin, David B Shmoys, Éva Tardos, and David P Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232, 1994.

[GGP$^+$15] Daniel Golovin, Vineet Goyal, Valentin Polishchuk, R Ravi, and Mikko Sysikaski. Improved approximations for two-stage min-cut and shortest path problems under uncertainty. *Mathematical Programming*, 149(1-2):167–194, 2015.

[GJ79] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[GKZ18] Fabrizio Grandoni, Christos Kalaitzis, and Rico Zenklusen. Improved approximation for tree augmentation: Saving by rewiring. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 632–645, New York, NY, USA, 2018. ACM.

[GNS11] Jiong Guo, Rolf Niedermeier, and Ondřej Suchý. Parameterized complexity of arc-weighted directed steiner problems. *SIAM Journal on Discrete Mathematics*, 25(2):583–599, 2011.

[GT88] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35:921–940, 1988.

[HK03] Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 585–594, 2003.

[HKS85] Alan J. Hoffman, Anthonius W. J. Kolen, and Michel Sakarovitch. Totally-balanced and greedy matrices. *SIAM Journal on Algebraic Discrete Methods*, 6(4):721–730, 1985.

[HMS19] Felix Hommelsheim, Moritz Mühlenthaler, and Oliver Schaudt. How to secure matchings against edge failures. In *36th International Symposium on Theoretical Aspects of Computer Science*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2019.

[Jai01] Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[JMM$^+$03] Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50(6):795–824, 2003.

[KN16] Guy Kortsarz and Zeev Nutov. A simplified 1.5-approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *ACM Transactions on Algorithms*, 12(2):23, 2016.

[KN18] Guy Kortsarz and Zeev Nutov. LP-relaxations for tree augmentation. *Discrete Applied Mathematics*, 239:94–105, 2018.

[Kol09] Vladimir Kolmogorov. Blossom v: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, 2009.

[KR08] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[KRT94] Valerie King, Satish Rao, and Rorbert Tarjan. A faster deterministic maximum flow algorithm. *Journal of Algorithms*, 17(3):447–474, 1994.

[Kru56] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[KY13] Panos Kouvelis and Gang Yu. *Robust discrete optimization and its applications*, volume 14. Springer Science & Business Media, 2013.

[LMMP12] Mathieu Lacroix, A. Ridha Mahjoub, Sébastien Martin, and Christophe Picouleau. On the np-completeness of the perfect matching free subgraph problem. *Theoretical Computer Science*, 423:25–29, 2012.

[MV80] Silvio Micali and Vijay V Vazirani. An o (sqrt(|v|) |e|) algoithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, pages 17–27. IEEE, 1980.

[Nut17] Zeev Nutov. On the tree augmentation problem. In *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87, page 61. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017.

[OPT93] James B Orlin, Serge A Plotkin, and Éva Tardos. Polynomial dual network simplex algorithms. *Mathematical programming*, 60(1-3):255–276, 1993.

[Orl13] James B. Orlin. Max flows in o(nm) time, or better. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, pages 765–774. Association for Computing Machinery, 2013.

[Oxl06] James G Oxley. *Matroid theory*, volume 3. Oxford University Press, USA, 2006.

[Pri57] Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

[PW70] Mike J Piff and Dominic JA Welsh. On the vector representation of matroids. *Journal of the London Mathematical Society*, 2(2):284–288, 1970.

[Sch03] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[SV14] András Sebő and Jens Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-tsp, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014.

[The19] The Optimization Firm. baron, 2019.

[WN99] Laurence A. Wolsey and George L. Nemhauser. *Integer and Combinatorial Optimization.* Wiley Series in Discrete Mathematics and Optimization. Wiley, 1999.

[WS11] David P Williamson and David B Shmoys. *The design of approximation algorithms.* Cambridge university press, 2011.