technische universität
dortmund

Parameterized Algorithms for Stochastic Steiner Tree Problems

Denis Kurz

Algorithm Engineering Report
TR12-1-001
March 2012
ISSN 1864-4503

Faculty of Computer Science
Algorithm Engineering (Ls11)
44221 Dortmund / Germany
http://ls11-www.cs.uni-dortmund.de/

technische universität
dortmund

Diploma Thesis

Parameterized Algorithms for Stochastic Steiner Tree Problems

Denis Kurz
January 2012

Reviewer:
Prof. Dr. Petra Mutzel
Dipl.-Inf. Bernd Zey

Technische Universität Dortmund
Department of Computer Science
Algorithm Engineering (Chair 11)
http://ls11-www.cs.tu-dortmund.de

Abstract

Since 1971, numerous parameterized algorithms for the Steiner tree problem have been published, first of which was introduced by Dreyfus and Wagner. However, up to today there is not a single parameterized algorithm for the two-stage stochastic Steiner tree problem. This problem is still practically important and deserves more attention. This document first presents an algorithm that has a worst-case running time of $\mathcal{O}((Kn)^2 3^{t^*})$, where K is the number of scenarios, n is the number of vertices of the input graph and t^* is the total number of terminals. Therefore, it places the two-stage stochastic Steiner tree problem as well as its prize-collecting variant in the complexity class FPT. Afterwards, a framework and some possible implementations are presented that might be suitable to parameterize the problem with the number of non-terminals.

Zusammenfassung

Der erste parametrisierte Algorithmus für das Steinerbaumproblem wurde im Jahr 1971 von Dreyfus und Wagner vorgestellt. Seitdem haben auch zahlreiche andere an derartigen Algorithmen gearbeitet. Obwohl auch das stochastische Steinerbaumproblem bekannt und praktisch relevant ist, wurde bisher kein einziger parametrisierter Algorithmus dafür veröffentlicht. Diese Arbeit hat das Ziel, die Lücke zu schließen. Es wird ein Algorithmus für das stochastische Steinerbaumproblem sowie dessen Prize-Collecting-Variante mit einer Laufzeit von $\mathcal{O}((Kn)^2 3^{t^*})$ vorgestellt, wobei K die Anzahl der Szenarien und t^* die Summe der Terminalanzahl über alle Szenarien bezeichnet. Damit wird das Problem erstmals in der Komplexitätsklasse FPT der parametrisierbaren Entscheidungsprobleme platziert. Anschließend werden erfolgversprechende Ideen erläutert, die eine Parametrisierung mit der Anzahl der Nicht-Terminale zulassen könnte.

Contents

Chapter 1

Introduction

In the 17th century, Pierre de Fermat wrote a letter to Evangelista Torricelli in which he formulated the following problem, see e.g. [12].

Given the coordinates of three points in the plane, find the coordinates of a fourth point such that the sum of distances of the original three points to the fourth point is minimal.

The fourth point is called the Fermat point or Torricelli point today. The problem of finding it is still relevant. Imagine a supermarket chain that intends to establish a new store serving three villages: The Fermat point of the three villages might be a good position to start with.

Both Fermat and Torricelli proposed a construction of the Fermat point. The problem was later generalized to what is known today as the Euclidean Steiner tree problem:

Given a set of points in the plane, interconnect the points with line segments while minimizing the total length of the line segments. The crossing points do not have to be points of the original point set.

The given points that we have to connect are called *terminals*.

In this setting, we may choose how many crossing points we add to connect the terminals. For example, if the terminals are collinear, a single line segment between the outermost two terminals interconnects them all without the need of any additional crossing points. Non-terminal crossing points are called *Steiner points*.

There may still be reasons why we would consider the Euclidean Steiner tree problem. We might want to connect some cities with a new system of streets. Of course, we are interested in minimizing the total cost. A set of street segments that represents a Euclidean Steiner tree might be a good basis. Probably, it would have to be refined because minimizing the total cost may not be the only objective.

The Euclidean Steiner tree problem has some restrictions that keep us from modeling problems that we encounter in practice as Euclidean Steiner tree problems. It only allows for one measurement, i.e., the Euclidean edge length. Furthermore, we cannot control the possible crossing points. Both drawbacks are fixed by the Steiner tree problem on weighted graphs.

A graph is a data structure that can be used to model relations between objects. In the case of Steiner tree problems, the objects are locations and the relations are candidates for connections between two locations. Weighted graphs allow us to assign arbitrary lengths or weights to such relations.

Given a weighted graph and some terminals in that graph, find a lightest connected subgraph that spans the terminals.

Consider an Internet Service Provider (ISP) that plans to install fiber optic cables to improve the Internet performance for a set of households. A connection has to be established between a central gateway and each of the households. Crossing cables are connected with each other by switches that are only allowed in some predetermined positions. The graph for this problem would contain the gateway, the households and the switch position candidates as nodes. Two nodes are connected if a cable may run directly between them. The edge weight is the cost for the installation of the corresponding cable. A minimum Steiner tree is the cheapest way for the ISP to achieve its goal.

This kind of modelling forces us to interconnect every pair of terminals. Perhaps, the ISP does not know in advance which of the households he wants to include in his new network. Including household A might be more profitable than including household B. This leads to the concept of prize-collecting Steiner trees:

Given a weighted graph and some terminals in that graph, each associated with a profit, find a connected subgraph such that the total weight of its edges minus the total profit of its nodes is minimal.

We are now allowed to spare some households if—economically speaking—connecting them would be a bad idea.

On the other hand, some of the data might be uncertain, i.e., the ISP might not even know which of the households have to be connected because its inhabitants may not be interested in the new product at all. Even the costs of connections between different households may differ. The ISP issues surveys among the inhabitants. These surveys lead to a set of scenarios that might possibly arise. Each scenario has its own set of households that have to be connected.

Because cables are cheaper the earlier they are bought, the ISP wants to install some them before the actual scenario is revealed. The cables that are installed in this early stage should minimize the overall costs, i.e., the costs of the edges themselves plus the expected costs of edges that have to be added afterwards to connect the households of the actual scenario. This problem is called the two-stage stochastic Steiner tree problem.

Given a weighted graph and a set of scenarios, find a set of first stage edges and a set of second stage edges for each scenario such that the terminals of each scenario are connected by its second stage and the first stage edges and the overall expected costs are minimal.

Of course, the prize-collecting and the two-stage stochastic Steiner tree problem can be combined. In this case, the ISP would not only deduce costs of connections and the interested households, but also the profit such a household would yield, should it be connected.

Then, we want to find a set of first and second stage edges such that the overall revenue is maximized. The overall revenue is the expected sum of edge costs minus the expected sum of profits yielded by the connected households. This last problem is called the two-stage stochastic prize-collecting Steiner tree problem.

1.1 Previous Work

In 1971, Dreyfus and Wagner [7] introduced the first parameterized algorithm for the Steiner tree problem. The algorithm was later improved to run in time $\mathcal{O}^*(2^{|T|})$ by Björklund et al. [1]. Independently, Fuchs et al. [9] proposed another parameterized algorithm that requires time $(2+\delta)^{|T|}n^{\mathcal{O}(1/(\delta/\ln(1/\delta))^{\zeta})}$ for any $0 < \delta < 1$ and $\frac{1}{2} < \zeta < 1$.

There are no publications concerning parameterized algorithm for the stochastic Steiner tree problem so far. The problem has been studied with respect to approximation algorithms. In 2004, Gupta et al. [11] proposed a 3.55-approximation for the rooted two-stage stochastic Steiner tree problem under certain cost sharing conditions. In 2005, Gupta and Pál [10] presented a constant-factor approximation for the unrooted version. For an overview of two-stage stochastic optimization problems, see Shmoys and Swany [17].

Recently, Bomze et al. [3] improved on the previously best-known ILP formulation for the stochastic Steiner tree problem. This might allow for better approximation algorithms.

1.2 Our Contribution

This document is dedicated to parameterized algorithms for the two-stage stochastic Steiner tree problem and its prize-collecting variant. As far as we know, we are the first to write about this aspect.

In Chapter 2 we provide the definitions and notation that are used throughout the document.

Chapter 3 gives an introduction to the basic, deterministic Steiner tree problem on graphs. We describe an algorithm that solves the problem if the underlying graph is a tree. We prove that

every variant of the Steiner tree problem discussed in this document is NP-complete on arbitrary graphs.

Chapter 4 covers the basics on stochastic Steiner trees. We present an algorithm that solves the two-stage stochastic Steiner tree problem on trees using the algorithm introduced in Chapter 3.

Chapter 5 deals with algorithms that are parameterized by the number of terminals. We describe a parameterized algorithm for the Steiner tree problem that was introduced by Dreyfus and Wagner [7] in 1971. This algorithm is expanded to solve the directed and the prize-collecting Steiner tree problem as well. Finally, the stochastic variants are solved by reducing them to their directed, deterministic counterparts.

In Chapter 6 we discuss ideas for parameterized algorithms that use the number of non-terminals as parameter. We show how to solve the Steiner tree problem and try to carry the results over to the stochastic Steiner tree problem.

Chapter 7 gives an overview of the results of this document and points out some open problems in the area of parameterized algorithms for SSTP.

Chapter 2

Definitions

This chapter provides all definitions that are used throughout the document. Definitions of basic concepts about graph problems are given in Section 2.1. The following Section 2.2 introduces combinatorial optimization problems and decision problems as well as complexity classes to categorize them. At last, we define the Steiner tree problem and those variations that are relevant in this document in Section 2.3.

2.1 Graphs

For an arbitrary set M, we denote by $\mathcal{P}(M) := \{M' \mid M' \subseteq M\}$ the *power set* of M, i.e., the set of all subsets of M. The *cardinality* $|M|$ of a set M is the number of its elements. Furthermore, $\mathcal{P}^k(M) := \{M' \in \mathcal{P}(M) \mid |M'| = k\}$ refers to the set of all subsets of M whose cardinality are exactly k.

Definition 2.1.1 (Undirected Graph)**.** *An undirected graph $G = (V, E)$ is a pair consisting of an arbitrary set V of* vertices *and an irreflexive, symmetric relation $E \subseteq V \times V$ of* edges *over V. Every edge is identified with its symmetric counterpart, yielding $(v_1, v_2) = (v_2, v_1)$ for each $v_1, v_2 \in V$.*

Let $G = (V, E)$ be an undirected graph with $v_1, v_2, v_3 \in V$ and $e_1 = (v_1, v_2), e_2 = (v_2, v_3) \in E$. The edge e_1 is said to be *incident* both to v_1 and v_2, while v_1 is *adjacent* to v_2 because of e_1 and vice versa. The two edges e_1, e_2 are also adjacent, as they both are incident to the same vertex v_2. The *neighborhood* $N_G(v)$ of a vertex v is the set of vertices that are adjacent to v in G, i.e., $N_G(v) = \{w \in V \mid (v, w) \in E\}$. The *degree* $\deg_G(v)$ of a vertex v is the number of its neighbors $|N_G(v)|$

Definition 2.1.2 (Directed Graph)**.** *A directed graph $G = (V, E)$ is a pair consisting of an arbitrary set V of vertices and an irreflexive binary relation $E \subseteq V \times V$ of edges over V.*

All terms defined for undirected graphs above can also be applied for directed graphs. In directed graphs, edges are considered to have an orientation. The edge (v_1, v_2) runs from its *tail* v_1 to its *head* v_2, but not from v_2 to v_1.

A node v is a *predecessor* of w if there is an edge (v, w). In this case, w is a *successor* of v. The *indegree* $\deg^-_G(v)$ of a vertex v in a graph G is the number of predecessors of v in G. Accordingly, $\deg^+_G(v)$ denotes the number of successors of v in G and is called *outdegree*.

Definition 2.1.3 (Subgraph)**.** *A graph $G' = (V', E')$ is a subgraph of a graph $G = (V, E)$ iff $V' \subseteq V$ and $E' \subseteq E$.*

Let $G = (V, E)$ be a graph and $V' \subseteq V$ a vertex subset. $G[V']$ denotes the subgraph of G that is *induced* by V', i.e., $G[V'] = (V', E')$ with $E' = \{(v_1, v_2) \in E \mid v_1, v_2 \in V'\}$. Note that every graph is a subgraph of itself induced by its whole vertex set. We also use the expressions $G - V' := G[V - V']$ and $G - F := (V, E \setminus F)$ for $F \subseteq E$.

Definition 2.1.4 (Weighted Graph)**.** *A weighted graph $G = (V, E, c)$ is a triple that consists of a vertex set V, an irreflexive binary relation $E \subseteq V \times V$ of edges and an edge weight function $c : E \to \mathbb{R}_0^+$. Weighted graphs may be undirected, in which case the edge relation is symmetric and edges are identified with their reverse edges.*

Note that for an undirected graph, the edges (v, w) and (w, v) for any adjacent vertices v, w have to be associated with the same weight due to identification of those two edges. For convenience we also write $c(E') := \sum_{e \in E'} c(e)$ to denote the weight of a subset $E' \subseteq E$ of a graph's edge set E.

Definition 2.1.5 (Reachability)**.** *The binary reachability relation $\rightsquigarrow_G$ of a graph $G = (V, E)$ is the reflexive, transitive closure of E. The node v_2 is* reachable *from the node v_1 iff $v_1 \rightsquigarrow_G v_2$.*

The reachability relation is always symmetric for undirected graphs. It can be either symmetric or asymmetric for directed graphs. The relation $\leftrightsquigarrow_G$, defined by $v_1 \leftrightsquigarrow_G v_2 :\Leftrightarrow v_1 \rightsquigarrow_G v_2 \wedge v_2 \rightsquigarrow_G v_1$, is called *mutual reachability*. Mutual reachability is an equivalence relation and identical to reachability in undirected graphs.

Definition 2.1.6 (Connected Component)**.** *A connected component is an equivalence class of $\leftrightsquigarrow_G$ in an undirected graph G. A strongly connected component is an equivalence class of $\leftrightsquigarrow_G$ in a directed graph G. The number of distinct (strongly) connected components of a graph G is denoted by $\#(G)$.*

A graph is *(strongly) connected* if the number of its (strongly) connected components equals 1.
An undirected *cycle* is a connected undirected graph $G = (V, E)$ with

$$\min_{v \in V} \deg_G(v) = \max_{v \in V} \deg_G(v) = 2.$$

A directed cycle is a strongly connected directed graph $G = (V, E)$ with

$$\max_{v \in V} \deg_G^-(v) = \max_{v \in V} \deg_G^+(v) = 1.$$

An undirected or directed *path* is obtained by removing one edge from an undirected or directed cycle, respectively. Vertices of a path that are incident to only one edge are called *endpoints* of the path. In directed paths, the vertices without predecessors or successors may also be called *head* or *tail* of the path, respectively. Cycles and paths may be weighted. The *length* $c(P)$ of a weighted path P is the sum of weights of its edge set.

For a graph $G = (V, E)$ with $v_1, v_2 \in V$ and $v_1 \rightsquigarrow_G v_2$ there exists a subgraph P of G that is a path whose endpoints are v_1 and v_2. Furthermore, if G is directed, there exists a path such that v_1 is the tail and v_2 is the head of this path. Subgraphs of this type are called paths from v_1 to v_2 in G regardless of whether G is directed or undirected. The path P is called shortest path from v_1 to v_2 if there exists no path P' from v_1 to v_2 in G with $c(P) > c(P')$. The length of a shortest path is called the *distance* $d_G(v_1, v_2)$ from v_1 to v_2 in G.

A *tree* is a connected undirected graph that does not contain a cycle as a subgraph. An *arborescence* is a directed graph $G = (V, E)$ with designated root r such that for every vertex $w \in V$ there is exactly one directed path from r to w. Vertices of trees and arborescences may be called *nodes*. Nodes with degree 1 are called *leafs*.

Let H be a subgraph of G. H is said to *span* G if the two graphs share the same node set. If H is a tree, it is called *spanning tree* of G. A minimum spanning tree of G is a spanning tree with minimum edge weight, assuming unit weight for unweighted graphs.

Every symbol defined above where a graph is used as a subscript argument may also be used without this argument if the graph is evident.

2.2 Combinatorial Problems

The computational problems explored in this document are combinatorial optimization and decision problems. The definition of combinatorial optimization problems is based on [5] and [15].

Definition 2.2.1 (Combinatorial optimization problem)**.** *An* instance of a combinatorial optimization problem *consists of a finite set Ω, a subset $\mathcal{S} \subseteq \mathcal{P}(\Omega)$ of feasible solutions and a value function $v : \Omega \mapsto \mathbb{R}_0^+$. A* combinatorial optimization problem *P is a set of instances and either a minimization or a maximization problem. For a solution $y \in \mathcal{S}$ we denote the sum of values of the elements of y by $v(y) := \sum_{\omega \in y} v(\omega)$. The objective is to find a solution $y^* \in \mathcal{S}$ with*

$$\forall y \in \mathcal{S} : v(y) \geq v(y^*)$$

if P is a minimization problem, and

$$\forall y \in \mathcal{S} : v(y) \leq v(y^*)$$

if P is a maximization problem.

A solution that meets the conditions of y^* in the definition above is called an optimal solution. Its value $v(y^*)$ is denoted by $\mathrm{OPT}(\Omega, \mathcal{S}, v)$, or OPT if Ω, $\mathcal{S}$ and v are evident.

Definition 2.2.2 (Decision problem)**.** *A* decision problem *is a* language *L, which is a subset of an underlying set Ω. An* instance of a decision problem *is an element ω of Ω. The objective is to decide whether or not $\omega \in L$.*

Elements of Ω may be called the *input* of the decision problem. We occasionally identify the problem with its language and thus speak of the input for L.

Combinatorial optimization problems yield canonical decision variants by supplementing an instance $(\Omega', \mathcal{S}, v)$ with a positive real number $k \in \mathbb{R}_0^+$. The underlying set Ω of our newly formed decision problem contains every such enhanced instance. Its subset L is then defined to contain every such instance whose optimal solution has a better value than k, i.e.,

$$L := \{(\Omega', \mathcal{S}, v, k) \mid \mathrm{OPT}(\Omega, \mathcal{S}, v) \leq k\}$$

for minimization problems and

$$L := \{(\Omega', \mathcal{S}, v, k) \mid \mathrm{OPT}(\Omega, \mathcal{S}, v) \geq k\}$$

for maximization problems. Given an instance $(\Omega', \mathcal{S}, v)$ of a combinatorial optimization problem and a positive real number k, the canonical decision problem can be interpreted as the question of whether or not $\mathrm{OPT}(\Omega, \mathcal{S}, v) \leq k$.

Optimization and other computational problems are categorized using a range of different properties, most notably the running time needed for algorithms to solve them. This categorization often helps to decide whether a problem is practically tractable. The following definitions are mostly adopted from [18].

Definition 2.2.3 (Complexity class P)**.** *A computational problem belongs to the complexity class P iff there exists an algorithm that solves the problem and has a worst case running time that is polynomial in the input length.*

This class P covers the problems that are usually considered practically tractable.

Definition 2.2.4 (Complexity class NP)**.** *A decision problem L belongs to the complexity class NP iff there exists a decision problem $L' \in P$ and a polynomial p such that $L = \{x \mid \exists y \in \{0,1\}^{p(|x|)} : (x, y) \in L'\}$.*

The class NP can thus be described as the class of decision problems that can be verified in polynomial time, given a *certificate* y. The certificate can be thought of as a proof that $x \in L$, encoded as a binary string. For a combinatorial problem $(\Omega, \mathcal{S}, v)$, the certificate for its canonical decision problem can be interpreted as an encoded solution I with $v(I) \leq k$. If there is no solution with better value than k, no such certificate exists.

Obviously, P $\subseteq$ NP because we may just ignore any certificates and solve the problem itself in polynomial time.

Definition 2.2.5 (Polynomial-time reducibility)**.** *Let D_1, D_2 be decision problems. D_1 is polynomial-time reducible to D_2 ($D_1 \leq_p D_2$) iff there exists a polynomial time computable function f that transforms every input x for D_1 into an input $f(x)$ for D_2 such that $x \in D_1 \Leftrightarrow f(x) \in D_2$.*

Polynomial-time reductions can be used to compare problems with each other.

Definition 2.2.6 (NP-Completeness)**.** *A decision problem $D \in NP$ is* NP-complete *($D \in NPC$) iff every problem $D' \in NP$ is polynomial-time reducible to D.*

If there is a problem $L \in \mathrm{NPC} \cap \mathrm{P}$, we would be able to reduce every problem in NP to this problem L, yielding a polynomial-time algorithm for every problem in NP and P = NP. However, today we assume that P $\neq$ NP. If this assumption is correct, NP-complete problems cannot be solved in polynomial time.

Due to the transitivity of polynomial-time reductions we only need to reduce one NP-complete problem to another problem R in NP to show that R is NP-complete, too.

As the title of this document suggests, we are interested in the parameterized complexity of stochastic Steiner tree problems. The following definitions are based on [6] and [14].

Definition 2.2.7 (Parameterized language)**.** *Let $L \subseteq \Omega$ be a language. A* parameterized language *L' is a subset of $L \times \mathbb{N}$. A corresponding decision problem requires to determine whether a pair $(\omega, k) \in \Omega \times \mathbb{N}$ belongs to L'. For an instance (ω, k), ω is called the* main part *and k the* parameter part *of the instance.*

The *length* of a pair $(\omega, k) \in \Omega \times \mathbb{N}$ is denoted by $|(\omega, k)|$. We denote by L^{Par} a modification of L that is parameterized by a parameter described by Par.

Definition 2.2.8 (Complexity class FPT)**.** *The complexity class FPT contains the decision problem of every parameterized language $L \times \mathbb{N}, L \subseteq \Omega$, that can be solved in time $\mathcal{O}(|(\omega, k)|^{\alpha} f(k))$ for each $(\omega, k) \in \Omega \times \mathbb{N}$, some $\alpha \in \mathbb{R}^+$ and a computable function f.*

For decision problems that are not known to be in P there may still be an algorithm that has polynomial worst-case running time under certain conditions. There might be a parameter $p(x)$ of inputs x such that the running time is polynomial except for a non-polynomial term that only depends on $p(x)$. Parameterized languages in P can also be found in FPT because the conditions are identical if we use $f(x) = 1, x \in \mathbb{N}_0$.

2.3 Steiner Tree Problems

This section defines the several Steiner tree problems that are treated in later chapters. All of them search for variations of a concept called Steiner tree.

Definition 2.3.1 (Steiner tree)**.** *Let $G = (V, E, c)$ be an undirected, weighted graph and $T \subseteq V$ a set of* terminal nodes*. An edge subset $F \subseteq E$ is called a* Steiner tree for T in G *iff there exists $F' \subseteq F$ and $V' \supseteq T$ such that (V', F') is a connected subgraph of G.*

The term Steiner tree is also used for the the connected subgraph (V', F'). Note that the graph (V', F') is always connected while $(V'', F), V'' = \{v \in V \mid \exists w \in V : (v, w) \in F \vee (w, v) \in F\}$ may have more than one connected component. We point out what we mean by the term Steiner tree whenever the distinction between those two variants is necessary but not immediately obvious. We also say that F and (V', F') *connect T in G*, respectively.

By definition, the vertices in $V' \setminus T$ and their incident edges are not strictly required for (V', F') to be a Steiner tree. They may, however, allow for smaller Steiner trees by introducing shortcuts. In some cases they may even be necessary to make the connection of T possible. In either case, these vertices are called *Steiner nodes*.

Each of the following Steiner tree problems is a minimization problem.

Definition 2.3.2 (Steiner tree problem)**.** *Let $G = (V, E, c)$ be an undirected, weighted graph with strictly positive edge weights and $T \subseteq V$ a set of terminals. The triple $(\Omega, \mathcal{S}, v)$ is an instance of the Steiner tree problem (STP), where $\Omega = E$, $v = c$, and $\mathcal{S}$ contains every Steiner tree for T in G.*

An algorithm solving the Steiner tree problem receives a representation of $(\Omega, \mathcal{S}, v)$ as *input*. The representation used for the algorithms in this document is (G, T).

The value of an optimal solution $\mathrm{OPT}(\Omega, \mathcal{S}, c)$ to the Steiner tree problem for the weighted, undirected graph G and the terminal set T is denoted by $SMT(G, T)$.

Definition 2.3.3 (Directed Steiner tree problem)**.** *Let $G = (V, E, c)$ be a directed, weighted graph, $T \subseteq V$ a set of terminals and $r \in T$ a designated root. The triple $(\Omega, \mathcal{S}, v)$ is an instance of the directed Steiner tree problem (DSTP), where $\Omega = E$, $v = c$ and $F \in \mathcal{S}$ iff there exists $V' \supseteq T$ such that (V', F) is an arborescence rooted at r.*

This directed variant of the Steiner tree problem asks for a directed subgraph S of G such that every terminal is reachable from r. We use the DSTP to deduce some results regarding the SSTP.

A common generalization of the Steiner tree problem is its prize-collecting variant:

Definition 2.3.4 (Prize-collecting Steiner tree problem)**.** *Let $G = (V, E, c)$ be an undirected, weighted graph and $P : E \mapsto \mathbb{R}_0^+$ a* node profit function*. The triple $(\Omega, \mathcal{S}, v)$ is an instance of the prize-collecting Steiner tree problem (PCSTP), where $\Omega = V \cup E$, $v(w) = P(w)$ for $w \in V$ and $v(e) = c(e)$ for $e \in E$, and $F \in \mathcal{S}$ iff $(V \setminus F, E \cap F)$ is a connected graph.*

In contrast to the STP, there are no terminals that we have to connect. Instead, some of the nodes of G are assigned a profit. A node's profit is paid out if the node is part of the Steiner tree. The value of a solution involves the sum of costs of the selected edges. Intuitively, the profit of every node that is covered by the selected edges is then subtracted. However, our definition of combinatorial optimization problems does not allow for subtracting, nor does it allow negative element weights.

To avoid this problem, we do not think of the included nodes as part of the solution, but the excluded ones. Thus, the value of a solution gets worse with every node with positive profit that is *not* covered by the selected edges. The value of a solution $V' \cup F$, $V' \subseteq V$, $F \subseteq E$, where V' consists of the vertices that are not included in the Steiner tree, is

$$v(V' \cup F) = \sum_{e \in F} c(e) + \sum_{w \in V'} P(w).$$

Since $\sum_{v \in V} p(v)$ is independent of a specific solution, we can subtract this from the value of each solution without changing the semantics of the problem or the set of optimal solutions. Therefore, it is safe to use the alternative measurement

$$v'(V^* \cup F) = \sum_{e \in F} c(e) - \sum_{w \in V^*} P(w)$$

to look for optimal solutions, where V^* is the set of vertices that are *included* in the Steiner tree. This measurement better suits our idea of profits that are paid out for vertices that are connected, though, technically speaking, it does not meet the conditions for measurements of combinatorial optimization problems.

The prize-collecting Steiner tree can also be considered for directed graphs:

Definition 2.3.5 (Directed prize-collecting Steiner tree problem)**.** *Let $G = (V, E, c)$ be a directed, weighted graph, $P : V \mapsto \mathbb{R}_0^+$ a node profit function and $r \in V$ a designated root. The triple $(\Omega, \mathcal{S}, v)$ is an instance of the directed prize-collecting Steiner tree problem (DPCSTP), where $\Omega = V \cup E$, $v(w) = P(w)$ for $w \in V$ and $v(e) = c(e)$ for $e \in E$, and $\mathcal{S} = \{F \subseteq \Omega \mid (V \setminus F, E \cap F)$ is an arborescence rooted at $r\}$.*

We utilize Cartesian products for our next definition, which requires some additional notation. Let E be an edge set and $n \in \mathbb{N}$. The elements of $E \times \{0, \ldots, n\}$ are pairs of an edge and an integer. For $e \in E, i \in \{0, \ldots, n\}$, we use the symbol e^i as shorthand for (e, i). For a subset $F \subseteq E$, F^i refers to the set $\{e \mid e^0 \in F \vee e^i \in F\}$.

Definition 2.3.6 (Two-stage stochastic Steiner tree problem)**.** *Let $G = (V, E, c^0)$ be an undirected, weighted graph with strictly positive edge weights, $K \in \mathbb{N}$ and $\mathcal{T} = \{(T^i, c^i, p^i) \mid 1 \leq i \leq K\}$ a set*

of scenarios*, where $T^i \subseteq V$ is a terminal set, $c^i : E \mapsto \mathbb{R}_0^+$ a weight function and $p^i \in (0,1)$ for $1 \leq i \leq K$, and $\sum_{i=1}^{K} p^i = 1$. The triple $(\Omega, \mathcal{S}, v)$ is an instance of the two-stage stochastic Steiner tree problem (SSTP), where $\Omega = E \times \{0, \dots, K\}$,*

$$v(e^i) = \begin{cases} c^0(e) & i = 0 \\ p^i c^i(e) & else \end{cases},$$

and $F \in \mathcal{S} :\Leftrightarrow F^0$ is the edge set of a connected subgraph of G and $\forall i \in \{1, \dots, K\} : F^i$ is a connected Steiner tree for T^i in G.

We assume a two-stage approach in our terminology, hence the name two-stage stochastic Steiner tree problem. Edges in F^0 are considered to be first stage edges. We have to pay their first stage price c^0. In this stage, we do not know which scenario will eventually occur in the second stage, and thus which terminal set to interconnect. For the i-th scenario, p^i denotes its probability of occurrence, T^i its terminal set, and c^i its specific edge cost function.

In the second stage, the actual scenario i is revealed and we might need to choose some additional edges $F^i \setminus F^0$ to augment the already bought edges to a Steiner tree for the terminal set T^i. The cost caused by edges bought in the second stage is their second stage price, which depends on the actual scenario.

A solution to an SSTP instance $(G, \mathcal{T})$ is called *stochastic Steiner tree* for $\mathcal{T}$ in G.

The value of a solution F is the sum of expected costs of all selected edges. For every second stage edge, we need to charge its second stage cost weighted by the probability that this edge is actually bought in the second stage. The value can thus be calculated as

$$v(F) = \sum_{e \in F^0} c^0(e) + \sum_{i=1}^{K} p^i \sum_{e \in (F^i \setminus F^0)} c^i(e).$$

The definition of $\mathcal{S}$ requires us to connect the second stage edges of each scenario to the first stage edges, because F^i has to be the edge set of a connected subgraph. If F^0 is empty, this requirement can be dropped as there are no first stage edges to connect to. In this case, the edges bought in each scenario have to be the edge set of a connected graph for itself.

The idea of stochastic Steiner trees can be combined with prize-collecting Steiner trees. It requires to extend the shorthand notation e^i for pairs $(e, i) \in E \times \{0, \dots, K\}$ to pairs $(v, i) \in V \times \{1, \dots, K\}$.

Definition 2.3.7 (Two-stage stochastic prize-collecting Steiner tree problem)**.** *Let $G = (V, E, c^0)$ be an undirected, weighted graph with strictly positive edge weights, $K \in \mathbb{N}$, $\mathcal{T} = \{(T^i, c^i, p^i) \mid 1 \leq i \leq K\}$ a set of scenarios as above and $P : V \times \{1, \dots, K\} \mapsto \mathbb{R}_0^+$ a node profit function that additionally depends on a scenario. The triple $(\Omega, \mathcal{S}, v)$ is an instance of the two-stage stochastic prize-collecting Steiner tree problem (SPCSTP), where $\Omega = (V \times \{1, \dots, K\}) \cup (E \times \{0, \dots, K\})$,*

$$v(x) = \begin{cases} c^0(e) & x = e^0, e \in E \\ p^i c^i(e) & x = e^i, e \in E \\ p^i P(v, i) & x = (v, i), v \in V \end{cases},$$

and $V' \cup E'$ belongs to $\mathcal{S}$ iff $(V \setminus V'^i, E'^i)$ is a connected graph for each scenario, $V' \subseteq (V \times \{1, \dots, K\})$, $E' \subseteq (E \times \{0, \dots, K\})$.

Again, there are no terminal nodes that have to be connected. The vertices are assigned a profit that depends on the scenario. A solution to SPCSTP consists of the selected edges E' and, for each scenario i, the vertices that are not covered by E'^i.

Note that the only variants that allow us to assign edge weights that are not strictly positive are DSTP and DPCSTP. The other variants require edge weights not to be 0 to avoid cycles that contain unnecessary edges with cost 0. Solutions for directed Steiner tree problems are arborescences and, thus, cannot contain cycles at all.

Chapter 3

The Steiner Tree Problem

This chapter is dedicated to the deterministic variant of the Steiner tree problem. The algorithms for the SSTP introduced in later chapters are derived from algorithms for the STP. Basic insight into STP is helpful to understand the observations and algorithms concerning its variants.

In Section 1.1 we give an overview of the already existing work regarding parameterized algorithms for STP. Section 3.1 provides the reader with some basic insights on the STP. The points made in this section are not necessary for the understanding of the rest of this document. Readers that are new to the field of Steiner trees are, nevertheless, encouraged to read through this section, as well as the next one. A simple algorithm is introduced in Section 3.2. It solves STP in linear time if the input graph is a tree. At last, NP-completeness proofs for STP and its variants are given in Section 3.3.

3.1 Preliminary Observations

This section provides the reader with some basic intuition on the subject of Steiner trees. It provides statements that are not strictly needed for the rest of the document. They may, however, help the reader understand some of the difficulties and fundamentals of Steiner tree problems.

Proposition 3.1.1. *Minimum Steiner trees are trees.*

Proof. Let $G = (V, E, c)$ be a graph, $T \subseteq V$ a terminal set and $S = (V', E')$ a minimum Steiner tree for T in G. Assume that S has k distinct connected components $C_1, \ldots, C_k, k > 1$ with $C_i = (V_i', E_i')$. Assume without loss of generality that C_1 contains at least one terminal. If there was another component C_i, $i \neq 1$, that also contained terminals, S would not provide paths from terminals in C_1 to terminals in C_i. In this case, S would not be a Steiner tree. Thus, C_1 contains every terminal.

As C_1 is a connected subgraph of G and contains every terminal, it is a Steiner tree for itself. Because of

$$\sum_{e \in E'} c(e) = \sum_{i=1}^{k} \sum_{e \in E_i'} c(e) > \sum_{e \in E_1'} c(e)$$

we are able to find a smaller Steiner tree C_1 for T in G, contradicting our assumption that S is not connected.

Now assume that S contains a cycle $C = (e_1, \ldots, e_r)$ as a subgraph for $r > 2$. For every pair t_1, t_2 of terminals we find a path from t_1 to t_2 in $G' - e_i, 1 \leq i \leq r$ as follows. Paths in G' that do not utilize e_i also exist in $G' - e_i$. Consider a path $P = (e_{j_1}, \ldots, e_{j_s})$ from t_1 to t_2 in G' that contains e_i. One connected component of the symmetric sum $C \otimes P = \{e \in E' \mid e \in C \cup P \wedge e \notin C \cap P\}$ is a tree and contains both t_1, t_2 but not e_i, yielding a unique path not containing e_i.

A minimum Steiner tree is connected and does not contain cycles and is therefore a tree. □

Proposition 3.1.2. *Let $G = (V, E, c)$ be an undirected, weighted graph and $T \subseteq V$ a terminal set. The leafs of a minimum Steiner tree S for T in G are terminals.*

Proof. Let S be a Steiner tree for T in G that contains a node $v \in V \setminus T$ with $\deg_S(v) = 1$, i.e., a Steiner node that is also a leaf of S. The only vertices of a path with degree 1 are its endpoints. Thus, a path P from one terminal t_1 to another terminal t_2 may only contain leafs of S as its endpoints. Obviously, the endpoints of P are terminals. The Steiner leaf v is not part of the path P for any selection of t_1, t_2, and is thus not needed to make S a Steiner tree. As we assume positive edge weights, we obtain a solution with smaller value by removing v and its incident edges from S, which is a contradiction to our assumption that S is a minimum Steiner tree. □

Proposition 3.1.3. *Let $G = (V, E, c)$ be an undirected, weighted graph and $T \subseteq V$ a terminal set with $T = \{v_1, \ldots, v_t\}$. For $v_i, w_j \in T, i < j$, let $P_{i,j} = (V_{i,j}, E_{i,j})$ denote one shortest path from v_i to v_j. The graph $G_1^* := (\bigcup_{i=1}^{t-1} \bigcup_{j=i+1}^{t} V_{i,j}, \bigcup_{i=1}^{t-1} \bigcup_{j=i+1}^{t} E_{i,j})$ is a Steiner tree for T in G.*

Proof. For every terminal pair $v_i, v_j \in T$, $P_{i,j}$ is a subgraph of G_1^*. Thus, every terminal is reachable from every other terminal within G_1^*. For a vertex v of G_1^* let $n(v)$ denote one of the terminals that are reachable from v. As every vertex of G_1^* lies on a shortest path between terminals, $n(v)$ exists for every such v. For vertex pairs v, w of G_1^*, a path from v to w can now be constructed by combining one path from v to $n(v)$, one path from $n(v)$ to $n(w)$ and one path from $n(w)$ to w and removing every cycle that may be formed by this union. Thus, G_1^* is connected. Its vertex set contains T. □

In most cases we are able to get much better results by exploiting the following result.

Proposition 3.1.4. *Let $G = (V, E, c)$ be an undirected, weighted graph with terminal set $T = (v_1, \ldots, v_t)$, $v \in V$ and $P_v(i) = (V_v(i), E_v(i))$ be a shortest path from v to v_i. The graph $G_2^*(v) := (\bigcup_{i=1}^{t} V_v(i), \bigcup_{i=1}^{t} E_v(i))$ is a Steiner tree for T in G.*

Proof. As above, G_2^* is connected because paths can be joined at the central node v. Every terminal v_i is included in the vertex set of G_2^* as one endpoint of $P_v(i)$. □

3.2 Algorithm for STP on Trees

In this section we present a simple algorithm that solves the STP on trees, assuming strictly positive edge costs. Given a weighted tree $G = (V, E, c)$ and a terminal set $T \subset V$, we want to find a minimum Steiner tree for T in G. We do this by computing the edge set of a minimum Steiner tree. We assume that every edge of G has strictly positive edge costs.

First of all, it is important to notice that for each Steiner tree instance on trees there is exactly one minimum Steiner tree.

Lemma 3.2.1. *If G is a tree, G_1^* is the only minimum Steiner tree connecting T in G.*

Proof. A Steiner tree S has to provide for every terminal pair v_i, v_j a path from v_i to v_j to make sure that S is connected. If G is a tree, there is exactly one path from v_i to v_j. There does not even exist a path that is longer than $d_G(v_i, v_j)$. Therefore, S must contain every single $P_{i,j}$ for $1 \leq i < j \leq k$. As a consequence, G_1^* must be a subgraph of S. Additional edges would increase the weight of S, making the solution worse, and additional vertices without additional edges would destroy the connectivity, yielding an infeasible solution. □

Note that for $v \in T$, $G_2^*(v)$ is a subgraph of G_1^* for an appropriate selection of paths. As G_1^* is optimal and $G_2^*(v)$ a subgraph and a Steiner tree, $G_2^*(v)$ is also optimal.

Now, we only need to decide for every edge whether or not it lies on a path between two terminals. This can be done by a simple depth first search starting in an arbitrary terminal. Eventually, the search reaches the leafs of the input tree and traverses the path from the selected

Algorithm 3.2.1 Compute the edge set of a minimum Steiner tree in a tree

Input: undirected, weighted tree $G = (V, E, c)$, terminal set $T \subseteq V$
Output: edge set of a minimum Steiner tree for T in G

```
1: function FindSteinerTreeEdges(G, T)
2:     for e ∈ E do
3:         selected(e) = False
4:     for v ∈ V do
5:         visited(v) = False
6:     r ← any terminal in T
7:     FindEdgesRec(G, T, r)
8:     return selected edges
9: end function
```

root terminal to the leafs in reverse. After we pass the first terminal, every following edge on the way back to the root is selected for inclusion for our desired edge set.

To compute the edge set of a minimum Steiner tree for an undirected, weighted tree G and a terminal set T in G, we call the function FindSteinerTreeEdges (Algorithm 3.2.1) with parameters G and T. This function marks every edge unselected and every node unvisited and starts a depth first search in an arbitrary terminal r—called the root henceforth—by invoking the recursive function FindEdgesRec with the parameters G, T and r. The edges that are selected in one of the recursive calls to FindEdgesRec are then returned by FindSteinerTreeEdges.

Algorithm 3.2.2 Recursive part of the algorithm used to compute the edge set of a minimum Steiner tree in a tree

Input: undirected, weighted tree $G = (V, E, c)$, terminal set $T \subseteq V$, $v \in V$
Output: edge set of a minimum Steiner tree for every terminal in the subtree of G under v

```
1: function FindEdgesRec(G, T, v)
2:     if visited(v) is False then
3:         visited(v) ← True
4:         if v ∈ T then
5:             result ← True
6:         else
7:             result ← False
8:         for (v, s) ∈ E do                          ▷ s for "successor"
9:             descHasTerminal ← FindEdgesRec(G, T, s)
10:            result ← result ∨ descHasTerminal
11:            if descHasTerimal is True then
12:                selected((v, s)) = True
13:        return result
14:    else
15:        return False
16: end function
```

A call to the function FindEdgesRec(G, T, v) (Algorithm 3.2.2) returns True iff v or at least one of its successors (nodes that have to pass v to reach the root r) is a terminal.

Theorem 3.2.1. *The above algorithm computes the edge set of a minimum Steiner tree for T in a tree G correctly in time $\mathcal{O}(|V|)$.*

Proof. Consider an edge $e = (v, s) \in E$ with $d_G(r, v) < d_G(r, s)$. The input tree G can be divided into two smaller trees $G_1 = (V_1, E_1), G_1 = (V_2, E_2)$ with $r \in V_1, s \in V_2$ by removing the edge e from G. If and only if $V_2 \cap T \neq \emptyset$, a call to FindEdgesRec$(G, T, s)$ returns True and (v, s) is selected. Thus, for every terminal $t \in T$ the edge set of the only path from r to t is part of the

algorithm's result. The result corresponds to the edge set of $G_2^*(r)$ and is thus a minimum Steiner tree for G.

The running time of the function FindSteinerTreeEdges(G, T) is $\mathcal{O}(|V| + |E|)$ (lines 2–6) plus the total running time for all invocations of FindEdgesRec, whose outer If block is entered once per vertex. Its lines 3–13 are thus executed once per vertex. Lines 3–7 require total running time $\mathcal{O}(|V|)$ when $v \in T$ can be checked in $\mathcal{O}(1)$. For each edge (v, w), lines 8–12 are executed twice: Once in FindEdgesRec(G, T, v) and once in FindEdgesRec(G, T, w), requiring total running time $\mathcal{O}(|E|)$. Because trees have $|V| - 1$ edges, we have an overall running time of $\mathcal{O}(|V|)$. □

3.3 NP-completeness of Steiner tree problems

As we pointed out in the definitions of the complexity class FPT, we are interested in parameterized algorithms only for problems that are at least NP-complete. In this section we show that Steiner tree problems are therefore candidates for such algorithms. We provide an NP-completeness proof for the decision variant of STP by placing it in NP and then reducing SAT to it. We refer to the decision variant of STP as STP in the following text for convenience.

Lemma 3.3.1. *STP ∈ NP*

Proof. Let F be the edge set of a solution to the graph $G = (V, E, c)$ of the STP instance (G, T, k). We can compute the sum $\sum_{e \in F} c(e)$ in time $\mathcal{O}(|E|)$ and compare it in constant time to k. □

A *literal* of a *boolean variable* x is either x or $\neg x$. Let $X = \{x_1, \ldots, x_n\}$ be a set of variables. A *clause* over literals of variables in X is a set of these literals. An *assignment* for X is a function $a : X \mapsto \{True, False\}$. A clause $\{y_1, \ldots, y_r\}$, where each y_i is a literal of a variable in X, evaluates to True given an assignment a for X if there is $1 \leq i \leq r$ such that $a(y_i) =$ True. A clause is *satisfiable* if there is an assignment to its variables such that the clause evaluates to True.

Definition 3.3.1 (Satisfiability problem)**.** *The* satisfiability problem *(SAT) receives a set of variables $x_1, \ldots, x_n$ and a set of clauses $C_1, \ldots, C_m$ over literals of $x_1, \ldots, x_n$ as input. The output is True iff there is an assignment to the variables $x_1, \ldots, x_n$ such that every clause evaluates to True.*

Let $X = \{x_1, \ldots, x_n\}, \mathcal{C} = \{C_1, \ldots, C_m\}$ be an input for SAT. The function f transforms an input $(X, \mathcal{C})$ for SAT into an input $(G = (V, E, c), T, k)$ for STP with

- $V = \{v_0\} \cup \{z_i^j \mid 1 \leq i \leq n \wedge j \in \{0, -, +\}\} \cup \{D_j \mid C_j \in \mathcal{C}\}$
- $E = E_1 \cup E_2 \cup E_3$
  - $E_1 = \{(v_0, z_i^j) \mid 1 \leq i \leq n \wedge j \in \{-, +\}\}$
  - $E_2 = \{(z_i^0, z_i^j) \mid 1 \leq i \leq n \wedge j \in \{-, +\}\}$
  - $E_3 = \{(z_i^-, D_j) \mid \neg x_i \in C_j\} \cup \{(z_i^+, D_j) \mid x_i \in C_j\}$
- $c(e) = 1$ for each $e \in E$
- $T = \{v_0\} \cup \{z_i^0 \mid 1 \leq i \leq n\} \cup \{D_j \mid C_j \in \mathcal{C}\}$
- $k = m + 2n$

We first provide a lower bound for minimum Steiner trees for graphs that originate from this transformation.

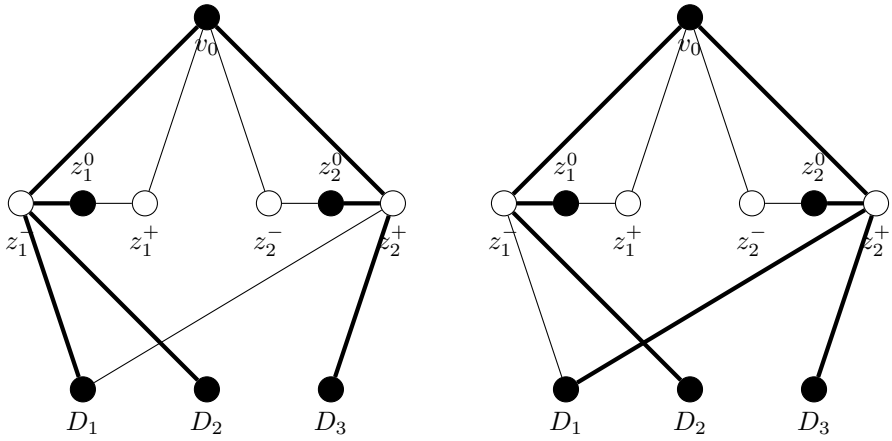
Lemma 3.3.2. *Let $(X = \{x_1, \ldots, x_n\}, \mathcal{C})$ be an instance for SAT and $(G, T, k) := f(X, \mathcal{C})$ with $G = (V, E, c)$. Then, SMT$(G, T) \geq k$.*

Proof. Paths from v_0 to terminals $z_i^0, 1 \leq i \leq n$, require using corresponding non-terminal node z_i^- or z_i^+. Thus, a Steiner tree for T in G has at least n Steiner nodes and exactly $m + n + 1$ terminal nodes. We need at least $(n + m + n + 1) - 1 = m + 2n$ edges to connect those nodes. □

The following lemma helps to recognize Steiner trees with no satisfying assignments.

Lemma 3.3.3. *Let $(X = \{x_1, \ldots, x_n\}, \mathcal{C})$ be an instance for SAT and $(G, T, k) := f(X, \mathcal{C})$ with $G = (V, E, c)$. Let $S = (V', E')$ be a Steiner tree for T in G with $z_i^-, z_i^+ \in V'$ for some $1 \leq i \leq n$. Then, $v(S) \geq k + 1$.*

Proof. We still need to choose at least $n - 1$ other Steiner nodes to connect v_0 to $z_j, j \neq i$. Then, we require $n + 1$ Steiner nodes in addition to the $n + m + 1$ terminal nodes, yielding $|V'| \geq n + 1 + n + m + 1 = m + 2n + 2$ and $|E'| \geq m + 2n + 1 = k + 1$. □



(a) Solution 1 (b) Solution 2

(c) Solution 3 (d) Solution 4

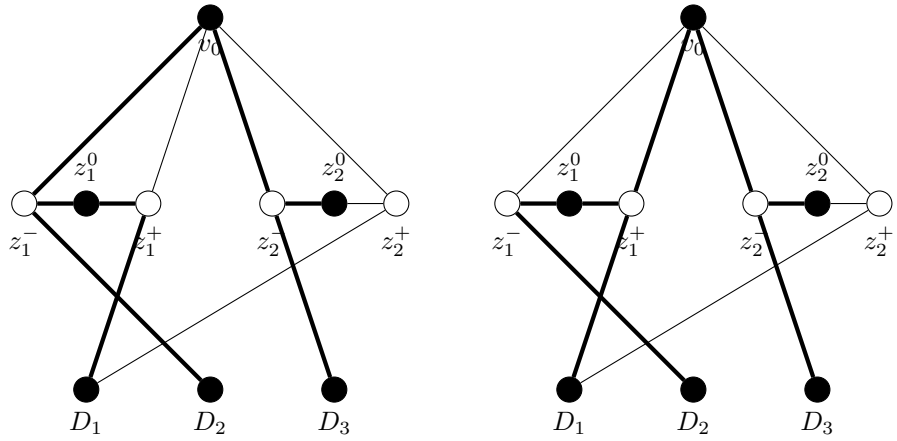
Figure 3.1: Example transformation of a satisfiable SAT instance including every optimal solution

Figures 3.1 and 3.2 depict transformations. Both are transformations from a SAT instance with variables x_1, x_2 and clauses C_1, C_2, C_3 with $C_2 = \{\neg x_1\}$. Figure 3.1 shows the graph of a satisfiable instance, where $C_1 = \{\neg x_1, x_2\}$ and $C_3 = \{x_2\}$. The SAT instance used for Figure 3.2 is not satisfiable. Its clauses are $C_1 = \{x_1, x_2\}$ and $C_3 = \{\neg x_2\}$. Some minimum Steiner trees are highlighted in both examples.

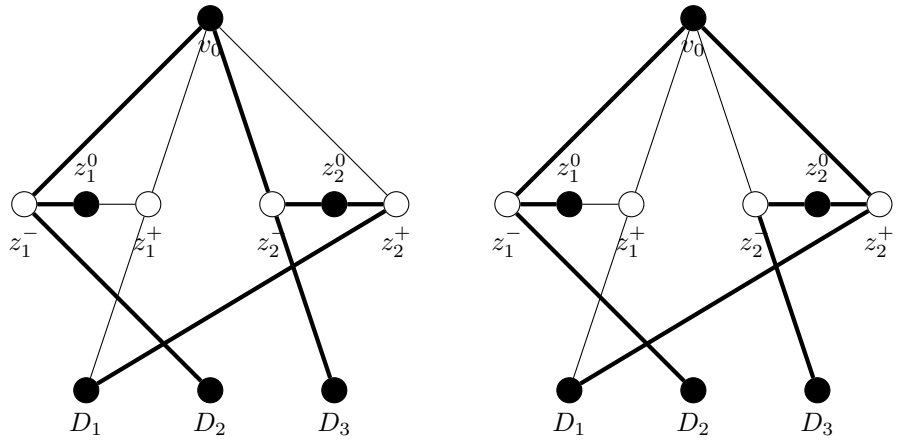
Proposition 3.3.1. *STP ∈ NPC*

Proof. As stated above, STP ∈ NP. Furthermore, SAT is reducible to STP by using f to transform SAT instances into STP instances. We need to prove the following equivalence:

$$\forall \text{SAT instance } (X, \mathcal{C}) : (X, \mathcal{C}) \in \text{SAT} \Leftrightarrow (G = (V, E, c), T, k) := f(X, \mathcal{C}) \in \text{STP}$$



(a) Solution 1 (b) Solution 2



(c) Solution 3 (d) Solution 4

Figure 3.2: Example transformation of an unsatisfiable SAT instance and some of its optimal solutions

"⇒": Assume that $(X, \mathcal{C}) \in$ SAT, i.e., there exists an assignment a that satisfies every clause in $\mathcal{C}$. Let $s : \{1, \ldots, m\} \mapsto \{1, \ldots, n\}$ be a mapping from clause to variable indices such that $(a(x_{s(j)}) = True \wedge x_{s(j)} \in C_j) \vee (a(x_{s(j)}) = False \wedge \neg x_{s(j)} \in C_j)$, i.e., s maps a clause index to the index of a variable that satisfies it. As a satisfies $\mathcal{C}$, such a mapping clearly exists.

The graph $S = (V', F)$ is a Steiner tree for T in G with $v(S) = k$, where

$$\begin{aligned} R &:= \{z_i^- \mid a(x_i) = False\} \cup \{z_i^+ \mid a(x_i) = True\}, \\ V' &:= T \cup R, \\ F_0 &:= \{(v_0, u) \mid u \in R\}, \\ F_v &:= \{(z_i^j, z_i^0) \mid z_i^j \in R\}, \\ F_- &:= \{(z_i^-, D_j) \mid a(x_{s(j)}) = False\} \text{ and} \\ F_+ &:= \{(z_i^+, D_j) \mid a(x_{s(j)}) = True\}, \\ F &:= F_0 \cup F_v \cup F_- \cup F_+. \end{aligned}$$

Every z_i^0 is connected to v_0 by the first two edge sets F_0 and F_v. The latter two edge sets F_- and F_+ make sure that the clause vertices D_j are connected to one Steiner node in R. The number of edges in F is exactly k.

"⇐": Assume that there is a Steiner tree $S = (V', F)$ for T in G with $\sum_{e \in F} c(e) \leq k$. For every $1 \leq i \leq n$ we have $\left|\{z_i^-, z_i^+\} \cap F\right| = 1$. If there were an i such that this intersection was empty, S would not be connected or the corresponding z_i^0 would not be part of S, because we can only reach z_i^0 by passing z_i^- or z_i^+. If there were an i such that $z_i^-, z_i^+ \in F$, the value of S would be at least $k + 1$ as stated in Lemma 3.3.3.

Consider the assignment

$$a(x_i) = \begin{cases} True & z_i^+ \in F \\ False & z_i^- \in F \end{cases}$$

for X. This assignment is well-defined as stated above. Moreover, it satisfies every clause in $\mathcal{C}$. To see this, we consider a clause $C_j \in \mathcal{C}$. Because of $D_j \in T$, it has to be part of S and thus connected to at least one other node in V. The only edges incident to D_j lead to the non-terminals z_i^-, z_i^+ for some i. Let z_i^l be a neighbor of D_j in S. By construction, the corresponding x_i satisfies C_j. □

Proposition 3.3.2. *DSTP ∈ NPC*

Proof. An undirected graph can be interpreted as a directed one where every edge falls into two edges, one for each direction. Instances of STP are thus also instances for DSTP, if we choose one of the terminals as root node, which is required for DSTP instances. Solutions to STP instances are not necessarily also solutions to the corresponding DSTP instances, because STP solutions are not required to be connected. Optimal solutions to STP instances, however, are connected and thus also DSTP solutions. The measurement of solutions is the same for both problems. □

Proposition 3.3.3. *PCSTP ∈ NPC*

Proof. Let $(G = (V, E, c), T, k)$ be an instance for STP. We can use the same graph G and upper bound k to construct an equivalent PCSTP instance. The profit p of every non-terminal has to be 0 because we do not want to be forced to include one of these in our solutions. If we now make sure that every terminal in T is included in every feasible solution, we can present an optimal solution to the generated PCSTP instance as a solutions for the original STP instance without changing its value. We achieve this by setting the profit of terminal nodes to a value that is high enough, e.g.,

$$p(v) = \sum_{e \in E} c(e) + 1$$

for each $v \in T$. We can always avoid penalties $p(v)$ for omitted nodes v by selecting every single edge for a solution. Therefore, the value of an optimal solution is less than or equal to the sum of all edge weights. The value of a solution that skips at least one terminal is at least the sum of all edge weights plus one. Thus, optimal solutions for this kind of PCSTP instances are also optimal solutions for the original STP instance. □

Theorem 3.3.1. *SSTP ∈ NPC*

Proof. An STP instance can be extended to an SSTP instance by replacing T with a single scenario whose terminal set is T and whose edge weights are c. Let F be an optimal solution to the extended SSTP instance. Then F^1 is an optimal solution to the STP instance. On the other hand, a solution F for the STP instance can be expanded to $F \times \{0\}$, which is an optimal solution to the SSTP instance. □

The observations can easily be applied to DPCSTP and SPCSTP.

Corollary 3.3.1. *DPCSTP ∈ NP and SPCSTP ∈ NPC.*

Chapter 4

The Two-Stage Stochastic Steiner Tree Problem

This chapter addresses basics of the two-stage stochastic Steiner tree problem. We introduce the problem without discussing parameterized algorithms, which are presented in Chapters 5 and 6.

Section 4.1 provides insight to the problem itself; its main purpose is to familiarize the reader with it. Nevertheless, some of the points made in this section are referenced in later sections. Finally, an algorithm for SSTP on tree graphs is suggested in Section 4.2 which uses the algorithm and statements explained in Sections 3.2 and 3.1.

4.1 Preliminary Observations

As in the chapter on Steiner trees, this section helps building up some better understanding of the SSTP. We make some general statements about stochastic Steiner trees that lead to shorter explanations in the following sections.

The Steiner trees that are computed after the actual scenario is revealed are required to be connected by definition of the stochastic Steiner tree problem. Let F be a solution to the SSTP. There are two cases that can possibly arise: Either F^0 is empty or it is not. In the latter case, the edges selected in the second stage in each scenario i, i.e., $F^i \setminus F^0$, have to maintain connection to the edges in F^0.

In the former case, the second stage edges of different scenarios may be completely disjoint and spread over the entire graph without even touching each other in one vertex, as long as F^i is the edge set of a connected subgraph of the input graph for itself. If this is the case, we can find an optimal solution by computing minimum Steiner trees for each scenario independently, using an appropriate algorithm introduced in the chapter about Steiner trees.

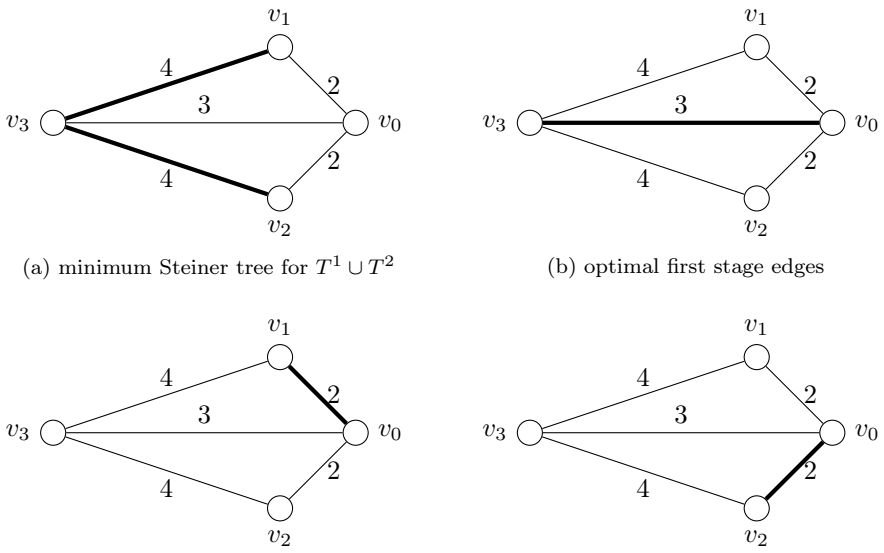
It is, therefore, sufficient to outline algorithms for the computation of optimal solutions under the condition that we have to buy at least one edge in the first stage. We can utilize these algorithms in a framework algorithm that first computes optimal solutions of the first type mentioned above using algorithms introduced in this chapter, then computes optimal solutions of the second type using algorithms of the previous chapter, and then finally selects the overall optimal solution.

First of all, we show that we actually deal with trees when discussing stochastic Steiner trees, just as the name suggests.

Proposition 4.1.1. *Let $(G, \mathcal{T})$ be an instance of SSTP and F an optimal solution to this instance. F^i is a tree for each $0 \leq i \leq K$.*

Proof. Consider an F^i, $1 \leq i \leq K$ that is not a tree. By definition of stochastic Steiner trees, F^i is connected. Therefore, it contains a cycle. Otherwise it would be a tree.

Two cases can arise in this situation. First, the cycle consists of edges in F^0 exclusively. Second, the cycle contains at least one second stage edge, i.e., an edge of $F^i \setminus F^0$.



(a) minimum Steiner tree for $T^1 \cup T^2$

(b) optimal first stage edges

(c) optimal second stage edges, scenario 1

(d) optimal second stage edges, scenario 2

Figure 4.1: Example SSTP instance with completely disjoint STP and SSTP solutions

In the first case, it is safe to remove one of the first stage edges. Removing an edge e from F^0 does not affect the connectivity of F^j, $1 \leq j \leq K$. $F \setminus \{(e, 0)\}$ is a feasible solution whose value is better than that of F itself. This contradicts our assumption that F is optimal.

In the second case, the removal of a first stage edge might cause F^j, $j \neq i$, not to be connected any more. The value can, nevertheless, be reduced by removing one second stage edge e from F^i that is part of a cycle. This edge does not affect the edges of the other scenarios. Its removal leaves the connectivity of F^i unchanged. Again, $F \setminus \{(e, i)\}$ is a better solution than F itself.

Hence, each F^i, $1 \leq i \leq K$, is a tree. F^0 is a connected subgraph of F^1 and thus a tree. □

Some proofs of later statements make a distinction depending on the stage an edge is bought in. We only consider the case that an edge is bought in the first stage *or* in the second stage, but not in both. The following proposition states that this is no restriction.

Proposition 4.1.2. *Let $(G, \mathcal{T})$ be an SSTP instance with $G = (V, E, c^0), \mathcal{T} = \{(T^i, c^i, p^i) \mid 1 \leq i \leq K\}$, and $F \subseteq E \times \{1, \ldots, K\}$ an optimal solution to this instance. There is no edge $e \in E$ with $e^0 \in F \wedge e^i \in F$ for some $i \in \{1, \ldots, K\}$.*

Proof. Assume the contrary. The removal of e^i from F does not have any effect on the connectivity of F^j for $j \neq i$. In fact, it does not even have an effect on F^i. As we did not remove e^0, e still remains in F^0, which is a subset of F^i by definition. Since $c^i(e) > 0$, the removal of e^i leads us to another feasible solution whose value is less than the value of F, which in turn cannot have been an optimal solution. □

One might think that SSTP can be solved by computing Steiner trees for the individual scenarios with their respective cost function and then combining these Steiner trees in some clever way to obtain an optimal selection of first stage edges. The following proposition suggests that this approach is not very promising.

Proposition 4.1.3. *There exist undirected, weighted graphs $G = (V, E, c)$ and scenario sets $\mathcal{T} = \{(T^i, c^i, p^i) \mid 1 \leq i \leq K\}$ such that the optimal solution to the SSTP on this input and the optimal solutions to the STP on G and T^i with their respective cost function c_i, $1 \leq i \leq K$, are disjoint.*

Proof. We prove this statement by giving an example graph that meets the conditions. The underlying graph $G = (V, E, c^0)$ for this example resembles a diamond shape as illustrated in Figure 4.1.

Its nodes v_0, v_1, v_2, v_3 are connected by the edges $(v_0, v_1), (v_0, v_2), (v_0, v_3), (v_1, v_3), (v_2, v_3)$ with $c((v_0, v_1)) = c((v_0, v_2)) = 2$, $c((v_0, v_3)) = 4$, and $c((v_1, v_3)) = c((v_2, v_3)) = 3$.

We have two scenarios in this example. The first one requires $T^1 = \{v_1, v_3\}$ to be connected, the second one $T^2 = \{v_2, v_3\}$. We assume that the cost of an edge increases in the second stage independently of the actual scenario and edge. This can be expressed by a global inflation factor. We use the factor 2, i.e., $c^1(e) = c^2(e) = 2 \cdot c^0(e)$ for each $e \in E$. The two scenarios are equiprobable, i.e., $p^1 = p^2 = \frac{1}{2}$.

The optimal solution to the STP instance $((V, E, c^1), T^1)$ consists of the single edge (v_1, v_3). Accordingly, the solution to $((V, E, c^2), T^2)$ only utilizes (v_2, v_3). The value of these solutions is 3, each. We need to buy (v_0, v_3) in the first stage and $(v_0, v_1), (v_0, v_2)$ in the second stage to achieve optimal results for SSTP. It is easy to see that no two of these solutions share edges. □

4.2 Algorithm for SSTP on Trees

In this section we describe one method to compute the edge set of an optimal stochastic Steiner tree for SSTP instances with trees as underlying graphs. For this purpose, we make use of the function FindSteinerTreeEdges that was introduced in Section 3.2.

In contrast to the Steiner tree problem on trees, SSTP instances on trees do not necessarily have a unique optimal solution. The variations between different solutions, however, are small. Differences can only occur in first stage edges F^0, but not in F^i for a scenario i, because the solutions to each scenario have to incorporate the unique optimal Steiner tree for the corresponding terminal set and be minimal under this condition.

As we learned in the section about the STP on trees, there is exactly one minimum Steiner tree for a given tree G and terminal set T, which, in fact, is independent of the edge cost function of G. For an optimal solution F to an SSTP instance $(G, \mathcal{T}), \mathcal{T} = \{(T^i, c^i, p^i) \mid 1 \leq i \leq K\}$, the minimum Steiner tree for T^i in G has to be included in F^i for each scenario i.

The idea of the following algorithm is to compute the minimum Steiner tree for each scenario and decide afterwards which edge should be bought in the first stage based on the expected cost we pay for each edge in the second stage.

We have to make sure that the second stage edges are connected to the first stage edges, if we buy first stage edges at all. To achieve this, we *guess* a node that is incident to a first stage edge by repeating the computation for every node in V.

Algorithm 4.2.1 Compute the edge set of a minimum stochastic Steiner tree in a tree

Input: undirected, weighted graph $G = (V, E, c^0)$, set of K scenarios $\mathcal{T} = \{(T^i, c^i, p^i) \mid 1 \leq i \leq K\}$
Output: edge set of a minimum stochastic Steiner tree for $\mathcal{T}$ in G

```
1: function FindStochSteinerTreeEdges(G, 𝒯)
2:     S ← FindDisconnectedSSTEdges(G, 𝒯)
3:     for r ∈ V do
4:         S' ← FindRootedSSTEdges(G, 𝒯, r)
5:         if c(S') < c(S) then
6:             S ← S'
7:     return A
8: end function
```

The function FindStochSteinerTreeEdges (Algorithm 4.2.1) first computes the optimal solution under the condition that we do not buy any edges in the first stage by calling FindDisconnectedSSTEdges. It then iterates over V: For each $r \in V$ it computes the optimal solution under the condition that r is incident to one edge of an optimal first stage by calling FindRootedSSTEdges. If we do buy edges in the first stage, calling FindRootedSSTEdges results in optimal solutions for at least two nodes. At the end of the algorithm, the best solution is returned.

The function FindDisconnectedSSTEdges (Algorithm 4.2.2) simply computes the STP solution to T^i on G for each scenario. As this function assumes an empty first stage, all edges have to be bought in the second stage. In this case, the solutions for the different scenarios do not have

Algorithm 4.2.2 Compute the edge set of a minimum stochastic Steiner tree without first stage edges in a tree

Input: weighted graph $G = (V, E, c^0)$, set of scenarios $\mathcal{T} = \{(T^i, c^i, p^i) \mid 1 \leq i \leq K\}$
Output: edge set of a minimum stochastic Steiner tree without first stage edges

```
1: function FindDisconnectedSSTEdges(G, 𝒯)
2:     for i ∈ {1, ..., K} do
3:         E_i ← FindSteinerTreeEdges(G, T^i)
4:     return {(e, i) | e ∈ E_i}
5: end function
```

an effect on each other and can thus be computed independently. We do not need to consider the scenario specific edge weight function because STP solutions on trees are unique anyway.

Algorithm 4.2.3 Compute an optimal solution to the SSTP instance $(G, \mathcal{T})$ whose first stage covers a given node r

Input: weighted graph $G = (V, E, c^0)$, set of scenarios $\mathcal{T} = \{(T^i, c^i, p^i) \mid 1 \leq i \leq K\}$, root $r \in V$
Output: stochastic Steiner tree for $\mathcal{T}$ in G whose first stage covers r

```
1: function FindRootedSSTEdges(G, 𝒯, r)
2:     for e ∈ E do
3:         f(e) ← 0
4:     for i ∈ {1, ..., K} do
5:         E_i ← FindSteinerTreeEdges(G, T^i ∪ {r})
6:         for e ∈ E_i do
7:             f(e) ← f(e) + p^i c^i(e)
8:     E_0 ← SelectFirstStageEdges(G, f, r, r)
9:     return E^0 ∪ {(e, i) | e ∈ (E_i \ E_0)}
10: end function
```

The function FindRootedSSTEdges (Algorithm 4.2.3) assumes that a node r is incident to at least one edge that is bought in the first stage, and thus to at least one edge in F^i for each scenario i and an optimal solution F. The idea of FindRootedSSTEdges is to compute the Steiner tree for $T^i \cup \{r\}$, $1 \leq i \leq K$, assuming that these edges are bought in the second stage of the corresponding scenario, and deciding afterwards which edge should be bought in the first stage.

To achieve this, we first add r to the terminal set and compute the optimal STP solution to $(G, T^i \cup \{r\})$ as above, again assuming that those solutions correspond to the edges bought in the second stage of each scenario. The function f keeps track of the expected costs caused by each edge. We increase $f(e)$ by $p^i c^i(e)$ if the edge e is bought in scenario i. This information is—in conjunction with the first stage costs of each edge—all we need to be able to decide which edge should be transferred into the first stage.

If the edges bought in the first stage were not required to be the edge set of a connected subgraph of G, we might simply use the edges e that meet $c^0(e) < f(e)$ as first stage edges. Instead, we call SelectFirstStageEdges to ensure that the selected edges are connected to each other.

The function SelectFirstStageEdges (Algorithm 4.2.4) starts a depth-first search in r to determine which subtrees have to be included in the solution. It returns the set of edges that provide the lowest total weight under all subtrees of G rooted at r. The measurement used here to tell whether the transfer of an edge from the second to the first stage is $c^0 - f$, i.e., the difference of the first stage cost of an edge subtracted by the expected cost that we have to pay for this edge in the second stage if it is not bought in the first stage. For an edge set E', this difference can be computed as

$$(c^0 - f)(E') = \sum_{e \in E'} (c^0(e) - f(e)).$$

Algorithm 4.2.4 Compute an optimal selection of first stage edges for the subtree under v

Input: weighted graph $G = (V, E, c^0)$, expected second stage costs f, current node v and its predecessor pre
Output: set of first stage edges

```
1: function SelectFirstStageEdges(G, f, v, pre)
2:     F ← ∅
3:     for (v, s) ∈ E do
4:         if s ≠ pre then
5:             F' ← SelectFirstStageEdges(G, f, s, v)
6:             if (c^0 − f)(F' ∪ {(v, s)}) > 0 then
7:                 F ← F ∪ (F' ∪ {(v, s)})
8:     return F
9: end function
```

We calculate some kind of profit to decide whether or not we should include edges of the subtree beneath (r, s). First, we sum up $c^0 - f$ of edges that belong to the optimal solution to the subtree under s. This edge set can be obtained with a recursive call to SelectFirstStageEdges. Second, we add $(c^0 - f)((r - s))$. If the resulting profit is positive, all these edges are included. Otherwise, the subtree beneath (r, s) is ignored completely.

We pass r—or in general, the predecessor of s—to SelectFirstStageEdges to avoid exploration of already visited nodes.

Consider a call to SelectFirstStageEdges with parameters G, c, v, pre. If v is a leaf of G, the only edge in E incident to v is (v, p). This edge is ignored by the above algorithm. The for-loop is skipped and the algorithm returns the empty set, which clearly is the optimal (and only) selection of edges for the empty subtree under v.

If v is an inner node of G, an optimal selection of first stage edges for G in the subtree that is rooted at v can be further decomposed. For a successor s of v, we denote by R_s the connected component containing v after removing p and every successor of v except for s. The optimal selection of edges of R_s rooted at s is empty if every non-empty solution has negative value, which would correspond to subtrees for which the transfer from the second to the first stage would be unprofitable. Otherwise, the solution has to contain the edge (v, s) and the edges of an optimal solution to the subtree beneath s, which is returned by SelectFirstStageEdges(G, f, s, v).

The solutions of different subtrees R_s do not affect each other. We can, therefore, solve these subproblems independently and compose their results to the solution of SelectFirstStageEdges(G, f, v, p).

Theorem 4.2.1. *The above algorithm solves the two-stage stochastic Steiner tree problem in time* $\mathcal{O}(|V|^3 + K \cdot |V|)$.

Proof. Our preliminary considerations illustrate that the algorithm works correctly. We only have to provide an upper bound for the algorithm's running time.

The function FindStochSteinerTreeEdges calls FindRootedSSTEdges once for each $r \in V$, and FindDisconnectedSSTEdges once. The latter computes K independent Steiner trees on the tree G, resulting in a running time of $\mathcal{O}(K(|V| + |E|))$. The former sets $f(e) = 0$ for each $e \in E$ in time $\mathcal{O}(|E|)$, and computes for each $1 \leq i \leq K$ the STP solution to $(G, T^i \cup \{r\})$ and an updated version of f in time $\mathcal{O}(|V| + |E| + |E_i|) = \mathcal{O}(|V| + |E|)$, resulting in an overall running time $\mathcal{O}(K(|V| + |E|))$ for lines 2–7. The following lines 8–9 are dominated by the call to SelectFirstStageEdges.

This function basically works like a depth-first search. Additionally, it computes $(c^0 - f)(F' \cup \{(v, s)\})$ for every edge (v, s) that has not been explored before, where F' is part of the subtree of G beneath s. We can precompute the function $c^0 - f$ in time $\mathcal{O}(|E|)$ before starting the depth-first search, allowing us to compute $(c^0 - f)(M)$ in time $|M|$. Altogether, the computation of SelectFirstStageEdges requires time $\mathcal{O}(|E| + (|V| + |E| \cdot |E|)) = \mathcal{O}(|V| + |E|^2)$.

This results in a running time of $\mathcal{O}((|V| + |E|) + |V|\,(|V| + |E|^2)) = \mathcal{O}(|V|\,(|V| + |E|^2))$ for FindStochSteinerTreeEdges. Because of $|E| = |V| - 1$ in trees, this can be simplified to $\mathcal{O}(|V|^3)$. □

Chapter 5

Parameter: Number of Terminals

This chapter is dedicated to algorithms with running times that are super-polynomial only in the number of terminals. The algorithms described here are suitable for solving every variant of the Steiner tree problem defined above. As a result, the problems $\text{STP}^{|T|}, \text{DSTP}^{|T|}$ etc. are placed in FPT.

In Section 5.1 we introduce the classical algorithm of Dreyfus and Wagner. It solves the STP in time $\mathcal{O}(n^2 3^{|T|})$. The central theorem of this section—the Optimal Decomposition Property—is adapted to DSTP and PCSTP in Section 5.2. In Section 5.3 we finally present a way to solve the two-stage stochastic Steiner tree problem in time $\mathcal{O}((Kn)^2 3^{|T^*|})$ with $T^* = \bigcup_{i=1}^{K} T^i$.

5.1 Algorithm of Dreyfus and Wagner

Up to today, the best-known parameterized algorithm for the STP is the one Dreyfus and Wagner introduced in [7]. It uses the dynamic programming approach by basically exploiting their central theorem, the so-called Optimal Decomposition Property. It states the following:

Theorem 5.1.1. *Let $G = (V, E, c)$ be an undirected, weighted graph, $T \subseteq V$ a set of terminals in G with $|T| \geq 3$ and $p \in T$. Let S be a minimum Steiner tree for T in G. There exist a* joining node *$q \in V$ and two terminal subsets $T_1, T_2, T_3 \subset T \cup \{q\}$ with $T_1 \cap T_2 = \{q\}$, $T_1 \cup T_2 = (T \setminus \{p\}) \cup \{q\}$ and $T_3 = \{p, q\}$ such that S can be split into three subgraphs S_1, S_2, S_3, where S_i is a minimum Steiner tree for T_i for each $1 \leq i \leq 3$.*

Proof. The following proof is borrowed from [7].

We first show that special subtrees of a minimum Steiner tree are minimum Steiner trees itself, but for a smaller set of terminals. Afterwards, we show that they can be decomposed into one shortest path and two minimum Steiner trees as the theorem suggests.

Given a minimum Steiner tree S and a node v of S, such subtrees S_i as above consist of all nodes that are reachable in S from v by only leaving v via a subset $M \subseteq N_S(v)$ of its neighbors. They can be constructed by removing $N_S(v) \setminus M$ from S and then only keeping the connected component of the resulting tree that contains v, i.e.,

$$V'(v, M) := \{w \in V \mid v \rightsquigarrow_{S-M} w\} \text{ and}$$
$$S(v, M) = (V', E') := S[V'].$$

Subtrees of this kind are candidates for S_1, S_2 as those two can be obtained like this, i.e., given S_1, S_2 as above, there are $M_1, M_2 \subseteq V$ such that $S_i = S(q, M_i), i = 1, 2$.

Let $T' := V' \cap T$. We need to show that $S(v, M)$ is a minimum Steiner tree for T' in G; so let us assume the opposite. It is obvious that $S(v, M)$ is a Steiner tree, so it is sufficient to prove that it is also minimal. If there exists a Steiner tree for T' in G with a smaller total weight than $S(v, M)$ we would be able to construct a Steiner tree for T in G by replacing $S(v, M)$ with the smaller Steiner tree for T' in S. However, S is an optimal solution. The construction of a smaller solution contradicts our assumption that $S(v, M)$ is not a minimum Steiner tree for T'.

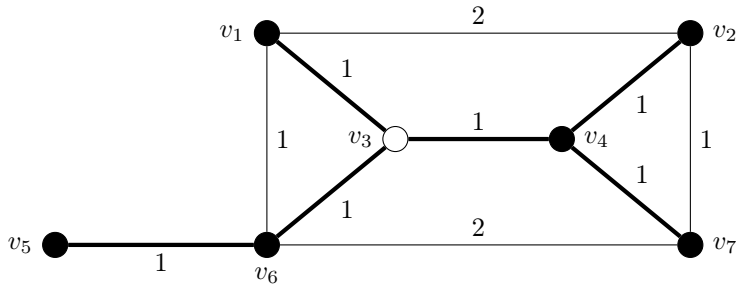


Figure 5.1: Example STP instance to illustrate the different cases covered by the Optimal Decomposition Property

We now prove that the tree decomposition suggested above is always possible by case differentiation. We need to provide a vertex $q \in V$ and a partition of $(T \setminus \{p\}) \cup \{q\}$ into two nonempty sets T_1, T_2 as defined above for any choice of $p \in T$. The different cases are illustrated in Figure 5.1. In this figure, terminal nodes are filled black and the edges of an optimal Steiner tree are highlighted.

The first case assumes that $\deg_S(p) > 1$. In this case, we have to choose $q = p$ and S_3 is empty. For $M \subseteq N_S(p)$ and $(V', E') := S(v, M)$ as above, we may use $T_1 := V' \cap T$ and $T_2 := (T \setminus T_1) \cup \{p\}$ to obtain a valid decomposition. Note that $T_1 \cup T_2 = T = (T \setminus \{p\}) \cup \{q\}$ for $q = p$.

The first case can be found in the example by setting $p = q = v_6, T_1 = \{v_5, v_6\}$ and $T_2 = \{v_1, v_2, v_4, v_6, v_7\}$, or $p = q = v_4, T_1 = \{v_1, v_4, v_5, v_6\}$ and $T_2 = \{v_2, v_4, v_7\}$. In fact, there are two other possible assignments to T_1, T_2 if $p = q = v_4$.

The second case, $\deg_S(p) = 1$, must be divided into two sub-cases. In the first sub-case, none of the terminals in $T \setminus \{p\}$ can be reached from p without passing an intermediate node v with $\deg_S(v) > 2$. Let w be the neighbor of v that has to be passed to reach p, $M \subseteq (N_S(v) \setminus \{w\})$ and $(V', E') = S(v, M)$. We set $q = v$, $T_1 = (T \cap V')$ and $T_2 = ((T \setminus (T_1 \cup \{p\})) \cup \{q\})$. Again, $T_1 \cup T_2 = (T_1 \cup (T \setminus T_1)) \setminus \{p\} = T \setminus \{p\}$ because of $p \notin (T \cap V')$. An example for this sub-case is $p = v_1, q = v_3, T_1 = \{v_3, v_5, v_6\}$ and $T_2 = \{v_2, v_3, v_4, v_7\}$. Note that $p = v_2$ or $p = v_7$ are also considered examples of this sub-case with $q = v_4$ and $T_1 = \{v_1, v_4, v_5, v_6\}$ and $T_2 = \{v_4, v_7\}$ $T_2 = \{v_2\}$, respectively.

In the second sub-case, we have $\deg_S(p) = 1$ and there is a $v \in T, v \neq p$, such that the path from p to v in S does not contain any other terminal. Let w be the neighbor of v that has to be passed to reach p, $M \subseteq (N_S(v) \setminus \{w\})$ and $(V', E') = S(v, M)$. It is now save to choose the same setting for q, T_1 and T_2 as we did in the last sub-case. An example for this is $p = v_5$, $q = v_6$, $T_1 = \{v_1, v_2, v_4, v_6, v_7\}$ and $T_2 = \{v_6\}$. □

A simple way to compute the value of a minimum Steiner tree using this result is shown in Algorithm 5.1.1.

Algorithm 5.1.1 Algorithm for STP that exploits the Optimal Decomposition Property

Input: undirected, weighted graph $G = (V, E, c)$, terminal set $T = \{t_1, \ldots, t_k\} \subseteq V$
Output: edge set of a minimum Steiner tree for T in G

```
1: function MinSteinerTreeODP(G, T)
2:     return DecompRec(G, t_1, {t_2, ..., t_k})
3: end function
```

To compute the value of an optimal Steiner tree, we call the function MinSteinerTreeODP (**O**ptimal **D**ecomposition **P**roperty) with parameters $G = (V, E, c)$ and $T \subseteq V$. This function just calls the recursive part of the algorithm by selecting an arbitrary terminal t_1 and passing it to DecompRec$(G, t_1, T \setminus \{t_1\})$.

The function DecompRec first checks in lines 2–5 whether the solution is trivial, in which case it just returns an appropriate result. It then tries to find the best joining node $q \in V$ and the best decomposition of T' into T_1, T_2 by simple enumeration. As we do not want to compute the value

Algorithm 5.1.2 Recursive part of the STP algorithm that exploits the Optimal Decomposition Property

Input: undirected, weighted graph $G = (V, E, c)$, vertex $p \in V$, terminal set $T' \subseteq V$
Output: edge set of a minimum Steiner tree for $T' \cup \{p\}$ in G

```
1:  function DecompRec(G, p, T')
2:      if T' = {p} then
3:          return 0
4:      if T' ∪ {p} = {p, w} then
5:          return d_G(p, w)
6:      min ← ∞
7:      v_1 ← arbitrary terminal in T'
8:      for q ∈ V do                                              ▷ |V|
9:          d ← d_G(p, q)
10:         if d < min then
11:             for {v_1} ⊆ T_1 ⊂ T' do                           ▷ O(2^|T'|)
12:                 T_2 ← T' \ T_1
13:                 m ← DecompRec(G, q, T_1) + DecompRec(G, q, T_2) + d
14:                 if m < min then
15:                     min ← m
16:     return min
17: end function
```

of a solution to a given decomposition twice because of commutation of T_1 and T_2, we require T_1 to always contain one selected terminal v_1.

The outer For loop spanning lines 9–15 requires $|V|$ iterations. Now consider line 11. There are $2^{|T|}$ subsets of T, one of which is T itself. Furthermore, there are $2^{|T\setminus\{v_1\}|}$ subset of $T \setminus \{v_1\}$. Thus, ignoring T itself and every subset of T that does not contain v_1, there are $2^{|T|} - 2^{|T\setminus\{v_1\}|} - 1 = 2^{|T|} - 2^{|T|-1} - 1 = 2^{|T|-1} - 1$ configurations for T_1.

The function then calls itself for every combination of q and T_1, T_2 twice, computing recursively the best solution to an instance with a smaller terminal set in the same graph. The main problem of this simple approach is the repeated computation of smaller solutions. The smaller the terminal set, the more often its optimal solution is computed.

Dreyfus and Wagner introduced the following algorithm that avoids some shortcomings of the above one by utilizing the dynamic programming approach. The variables $M(v, T')$ for $v \in V, T' \subseteq T$, are considered globally available.

The array M stores the value of the minimum Steiner tree for every terminal subset and one node of V. The symbol $M(v, T')$ thus holds the value of the minimum Steiner tree connecting $T' \cup \{v\}$ on condition that v is the node mentioned as p in the Optimal Decomposition Property.

The function DreyfusWagner first initializes the array M to contain the value of a minimum Steiner tree for the terminals $v \in V$ and $t \in T$, which corresponds to the distance from v to t in G. It then chooses one terminal node p that plays the role of the node that is also called p in the Optimal Decomposition Property. According to this theorem, p is only contained in the solution as part of the path from p to our yet unknown joining node q, so we can ignore it for now.

The function then computes for every $T' \in \mathcal{P}(T \setminus \{p\})$ and $q \in V$ the value of a minimum Steiner tree for $T' \cup \{p\}$ in G assuming that p and q are the nodes mentioned in the above theorem, i.e., we only take into account those solutions that are composed of a shortest path from p to q, a minimum Steiner tree connecting one subset T_1 of T' with q, and one minimum Steiner tree connecting another subset T_2 of T', where $T_1 \cup T_2 = T'$ and $T_1 \cap T_2 = \emptyset$. This task is fulfilled by the procedure UpdateValueOfMinSteinerTree.

After computing the values of optimal solutions for the subproblems mentioned above, the function DreyfusWagner combines some of these to form a minimum Steiner tree for T. For every possible joining node q it computes the optimal decomposition of $T \setminus \{p\}$ by calling OptDecomp$(q, T \setminus \{p\})$, and adds to the value of the two sub-solutions yielded by this decomposition

Algorithm 5.1.3 Algorithm of Dreyfus and Wagner

Input: undirected, weighted graph $G = (V, E, c)$, terminal set $T \subseteq V$
Output: edge set of a minimum Steiner tree for T in G

```
1: function DreyfusWagner(G, T)
2:     for v ∈ V do                                          ▷ Initialization: O(n^2)
3:         for t ∈ T do
4:             M(v, {t}) ← d_G(v, t)
5:     p ← arbitrary terminal in T
6:     for i ∈ {3, ..., |T| − 2} do
7:         for T' ∈ P^i(T \ {p}) do                          ▷ (|T\{p}| choose i)
8:             for v ∈ V do                                  ▷ |V|
9:                 M(v, T') ← ∞
10:            for q ∈ V do                                  ▷ |V|
11:                UpdateValueOfMinSteinerTree(G, q, T')
12:    v ← ∞
13:    for q ∈ V do                                          ▷ |V|
14:        u ← OptDecomp(q, T \ {p})
15:        if u + d_G(p, q) < v then
16:            v ← u + d_G(p, q)
17:    return v
18: end function
```

Algorithm 5.1.4 Dreyfus-Wagner: Update the value of $M(p, T)$ for each p using the joining node q

Input: undirected, weighted graph $G = (V, E, c)$, joining node $q \in V$, terminal set $T' \subseteq V$

```
1: procedure UpdateValueOfMinSteinerTree(G, q, T')
2:     u ← OptDecomp(q, T')
3:     for p ∈ V do                                          ▷ |V|
4:         if u + d_G(p, q) < M(p, T') then
5:             M(p, T) ← u + d_G(p, q)
```

the length of a shortest path from p to q. The smallest value found in this process is the value of a minimum Steiner tree for T in G, and thus the algorithm's output.

The statements not involving procedure or function calls require constant time, assuming that the distance graph for G is given. If the distance graph is unknown, it can be computed in a preprocessing step in $\mathcal{O}(n^3)$. For details on the all-pairs shortest path see [8] or [15]. To estimate the time complexity it is therefore sufficient to count the iterations of every For statement.

Note that some of the For loops are nested and that the number of iterations is annotated in comments next to each For statement except for the outermost. The function DreyfusWagner calls UpdateValueOfMinSteinerTree from inside a loop, which itself calls OptDecomp from inside a loop, yielding the most complex nesting of this algorithm. Its innermost block in OptDecomp is executed

$$\begin{aligned}&\sum_{i=3}^{|T|-2} \binom{|T \setminus \{p\}|}{i} |V|^2 \left(2^{i-1} - 1\right)\\ &= \mathcal{O}\left(|V|^2 \sum_{i=3}^{|T|-2} 2^i \binom{|T|-1}{i}\right)\end{aligned}$$

times.

Algorithm 5.1.5 Dreyfus-Wagner: Compute the minimum costs for two Steiner trees connecting q with one subset of T' each

Input: joining node q, terminal set T'
Output: minimum weight $\mathrm{SMT}(G, T_1)+\mathrm{SMT}(G, T_2)$ of two Steiner trees connecting q with T_1, T_2, respectively; $T_1 \cup T_2 = T'$

```
1: function OptDecomp(q, T')
2:     u ← ∞
3:     v ← arbitrary terminal in T'
4:     for {v} ⊆ T_1 ⊂ T do                          ▷ 2^{|T'|-1} - 1 = O(2^{|T'|})
5:         T_2 ← T' \ T_1
6:         u' ← M(q, T_1) + M(q, T_2)
7:         if u' < u then
8:             u ← u'
9:     return u
10: end function
```

The binomial theorem states that

$$(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^i y^{n-i}$$

for a semi-ring M and $x, y \in M$.

For $n = |T| - 1, x = 2$ and $y = 1$ we obtain

$$\begin{aligned} 3^{|T|-1} &= (2+1)^{|T|-1} \\ &= \sum_{i=0}^{|T|-1} \binom{|T|-1}{i} 2^i 1^{|T|-1-i} \\ &= \sum_{i=0}^{|T|-1} \binom{|T|-1}{i} 2^i. \end{aligned}$$

We can thus simplify the running time term to

$$\begin{aligned} &\mathcal{O}\left(|V|^2 \left(3^{|T|-1} - \left(1 \cdot 1 + 2\left(|T|-1\right) + 4\binom{|T|-1}{2} + 2^{|T|-1} \cdot 1\right)\right)\right) \\ &= \mathcal{O}\left(|V|^2 3^{|T|}\right). \end{aligned}$$

The canonical decision problem of STP can be solved by computing the value of a minimum Steiner tree and comparing this value against some upper bound k.

Corollary 5.1.1. *$STP^{|T|} \in FPT$.*

5.2 Adjusting Dreyfus-Wagner to DSTP and PCSTP

The original algorithm of Dreyfus and Wagner solves the undirected, deterministic Steiner tree problem STP. The main ingredient of this algorithm is the Optimal Decomposition Property for Steiner trees in undirected graphs. This central theorem can be expanded to directed graphs as follows:

Corollary 5.2.1. *Let $G = (V, E, c)$ be a directed, weighted graph, $T \subseteq V$ a set of terminals in G with $|T| \geq 3$ and $r \in T$ a designated root. Let S be a minimum Steiner tree for (G, T, r). There exist a joining node $q \in V$ and two terminal subsets $T_1, T_2 \subset T \cup \{q\}$ with $T_1 \cap T_2 = \{q\}$, $T_1 \cup T_2 = (T \setminus \{r\}) \cup \{q\}$ and $T_3 = \{p, q\}$ such that S can be split into three subgraphs S_1, S_2, S_3, where S_i is a minimum directed Steiner tree for T_i rooted at q for each $1 \leq i \leq 3$.*

This statement can be proven in exactly the same way as the original Optimal Decomposition Property above.

Every terminal of a directed Steiner tree has to be reachable from its root r. Every terminal except for r can be found in either S_1 or S_2 (or both, if $q \in T$), which are itself directed Steiner trees rooted at q. A directed path from r to another terminal node s can be found by concatenating a path from r to q—e.g., the subgraph S_3, which is part of S by construction—and a path from q to s in the corresponding subgraph S_1 or S_2, which has to provide such a path by definition of a directed Steiner tree because s is a terminal of this sub-solution.

While the Optimal Decomposition Property is almost the same for STP and DSTP, it requires some more adjustments to fit it to PCSTP.

Proposition 5.2.1. *Let $G = (V, E, c)$ be an undirected, weighted graph, $p : V \mapsto \mathbb{R}_0^+$ a node profit function, $T = \{v \in V \mid p(v) > 0\}$ and $r \in T$. Let S be a minimum prize-collecting Steiner tree for p in G. There exist a joining node $q \in V$ and subsets T_1, T_2, T_3 of $(T \setminus S) \cup \{q\}$ with $T_1 \cup T_2 = \{q\}$, $T_1 \cup T_2 = ((T \setminus S) \setminus \{r\}) \cup \{q\}$ and $T_3 = \{r, q\}$ such that $S = (V \cap \bigcap_{i=1}^{3} S_i) \cup (E \cup \bigcap_{i=1}^{3} S_i)$, where S_i is a minimum prize-collecting Steiner tree for p_i in G for each $1 \leq i \leq 3$, with*

$$p_i(v) = \begin{cases} W & v = q \\ p(v) & v \in T_i \\ 0 & \textit{otherwise} \end{cases}$$

for $v \in V$ and $W = \sum_{e \in E} c(e)$.

Proof. Setting $p_i(q)$ to W forces us to cover q by minimum prize-collecting Steiner trees and thus excluding it from the sub-solutions, because solutions for PCSTP instances contain those nodes that the tree does not cover by definition. For the same reason, $T \setminus S$ contains exactly the nodes of T that are covered by the edges in S and thus by the actual Steiner tree.

Vertices that are excluded from the optimal prize-collecting Steiner tree for p are those that are excluded by all three sub-solutions for p_1, p_2, p_3. The solution contains an edge iff there is one sub-solution that contains it. This leads to exactly the composition described above.

The rest of the statement can be shown analogously to the proof of the original Optimal Decomposition Property. □

The algorithm of Dreyfus and Wagner solving the undirected STP can be used to solve DSTP in the above form. Even the distance graph can be computed using the same algorithm as above, namely [8].

Corollary 5.2.2. *$DSTP^{|T|} \in FPT$.*

Again, the algorithms introduced above need some adjustments before we can use them to solve PCSTP. We assume here that profits of nodes that are covered by the selected edges of a solution are subtracted from the value of a solution, although the definition of PCSTP would require us to add the profit of every node that is not included. The resulting problem is equivalent to PCSTP. For details, see Definition 2.3.4.

First, it is not sufficient to remove one of the terminals r and solve the problem for every proper subset of $T \setminus \{r\}$. Since there is no vertex that we are forced to include in a feasible solution, we have to compute the solution for every single subset of T and return the best of them.

For another thing, after the execution of the algorithm the value of an optimal solution is not necessarily stored in $M(v, T)$ for some $v \in V$ because it may be more profitable to ignore some of the vertices with positive profit. Therefore, we have to store the value of the most profitable sub-solution we have seen so far. This value can be initialized with ∞ and has to be updated every time line 5 of UpdateValueOfMinSteinerTree is executed. The new value is $M(r, T) - p(r) - \sum_{v \in T} p(v)$ if $M(r, T)$ is updated.

Corollary 5.2.3. *$PCSTP^{|T|} \in FPT$.*

The thoughts on DSTP and PCSTP can be combined to obtain an algorithm for DPCSTP. The Optimal Decomposition Property for DPCSTP resembles Proposition 5.2.1. It differs only in the input graph and the solution, which both have to be connected for DPCSTP.

Corollary 5.2.4. *$DPCSTP^{|T|} \in FPT$.*

5.3 Reduction to the Directed Steiner Tree Problem

In this section we describe a method to solve the two-stage stochastic Steiner tree problem by transforming its instances to instances of the directed Steiner tree problem that we are already able to solve using the algorithm of Dreyfus and Wagner.

Let $(G, \mathcal{T})$ be an instance of the SSTP, where $G = (V, E, c^0)$ and $\mathcal{T} = \{(T^i, c^i, p^i) \mid 1 \leq i \leq K\}$. Our goal is to generate an instance (G', T), $G' = (V', E', c')$, for DSTP such that an optimal solution to the transformed DSTP instance leads back to an optimal solution to the original SSTP instance.

The directed graph G' that results from our transformation contains one copy of G for each scenario, and one for the first stage. The graph that corresponds to scenario i is denoted by G'^i. The node in G'^i that corresponds to $v \in V$ is denoted by v^i. In the same manner, G'^0 and v^0 refer to the copy of G and $v \in V$ in the first stage copy, respectively. We also need an independent root node r that does not correspond to any node in V. The node set of the transformed graph is

$$V' = \{v^i \mid v \in V \wedge 0 \leq i \leq K\} \cup \{r\}.$$

The edge set E' of G' includes the edges (v^i, w^i) for $0 \leq i \leq K$ and $(v, w) \in E$ that originate from the $(K+1)$-fold replication of G. Note that these edges are interpreted as directed edges—one for each direction—in G', while they were interpreted as one undirected edge in G. The weight of an edge (v^0, w^0) in the first stage copy is set to the first stage cost c^0 of (v, w). The weight of an edge (v^i, w^i), $i > 0$, is the scenario's second stage cost of the corresponding edge (v, w) multiplied with the scenario's probability p^i.

The graph G' is disconnected in its current state. The different copies of G have to be interconnected with edges (v^0, v^i) for each $v \in E$, $1 \leq i \leq K$, i.e., the node that corresponds to v in the first stage copy is connected to the node corresponding to v in each scenario copy. All of these edges receive an edge weight of 0. Note that these edges only lead from the first stage copy to the scenario copies, but not in the opposite direction. There is no path from v^i to w^0 in G' for $v, w \in V$, $i > 0$.

At last, the newly created root node r is connected to each node in the first stage copy of G. These edges are unidirectional, too; also adding them in the opposite direction still would not change the set of optimal solutions, nor would it change their values. The weight of these edges has to be high enough to make sure that only one of them is chosen in an optimal solution. The sum of first stage costs of the SSTP instance,

$$W = \sum_{e \in E} c^0(e),$$

is sufficient for this purpose.

The edge set E' of G' can be written as

$$\begin{aligned} E'_1 &= \{(r, v^0) \mid v \in V\}, \\ E'_2 &= \{(v^0, v^i) \mid v \in V, 1 \leq i \leq K\}, \\ E'_3 &= \{(v^i, w^i) \mid (v, w) \in E, 1 \leq i \leq K\}, \\ E' &= E'_1 \cup E'_2 \cup E'_3, \end{aligned}$$

and are weighted by

$$c(e) = \begin{cases} W & e = (r, v^0) \\ c^0(e) & e = (v^0, w^0) \\ 0 & e = (v^0, v^i) \\ p^i c^i(e) & \text{otherwise} \end{cases}.$$

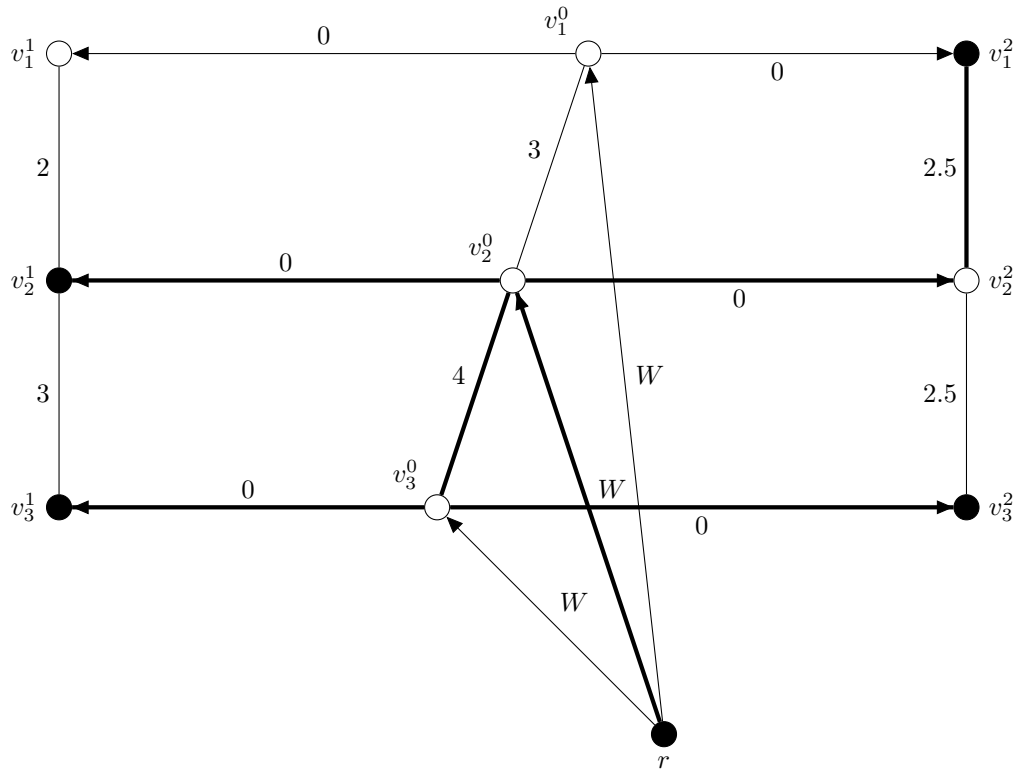


Figure 5.2: Example transformation of an SSTP instance into an equivalent DSTP instance. Bidirected edges are drawn as undirected ones for simplicity.

The terminal set of the DSTP instance consists of the root r and the terminal nodes of each scenario in the corresponding copy of G, i.e., $T = \{v_i^j \mid v_i \in T^j\} \cup \{r\}$.

An example transformation is illustrated in Figure 5.2. Consider the SSTP instance $(G, \mathcal{T})$, where G is an undirected, weighted path with nodes $\{v_1, v_2, v_3\}$ and edge weights $c^0((v_1, v_2)) = 3$ and $c^0((v_2, v_3)) = 4$.

The first of two equiprobable scenarios has edge weights $c^1((v_1, v_2)) = 4$ and $c^2((v_2, v_3)) = 6$ and terminal set $T^1 = \{v_2, v_3\}$. The terminals of the second scenario are $T^2 = \{v_1, v_3\}$, while its edge weights are $c((v_1, v_2)) = c((v_2, v_3)) = 5$.

In the course of the transformation, the graph is replicated three times: Once for each scenario and once for the first stage. The first stage copy in the middle of the figure, involving nodes v_1^0, v_2^0, v_3^0, is connected node-wise to the copy symbolizing the first scenario on the left side and the copy symbolizing the second scenario on the right side with edge weights 0. While the first stage costs are carried over to the transformed graph, the second stage costs are weighted with $\frac{1}{2}$, resulting in $c((v_1^1, v_2^1)) = c((v_2^1, v_1^1)) = 2$, $c((v_2^1, v_3^1)) = c((v_3^1, v_2^1)) = 3$ and $c((v_1^2, v_2^2)) = c((v_2^2, v_1^2)) = c((v_2^2, v_3^2)) = c((v_3^2, v_2^2)) = \frac{5}{2}$.

At last, the additional node r is connected to v_1^0, v_2^0, v_3^0 with edge cost $W = 7$ and acts as the root node for the DSTP instance.

An optimal solution for the DSTP instance is highlighted in the figure. It is important to note that the edges (v_3^0, v_2^0) and (v_1^2, v_2^2) are not part of the solutions. These edges are highlighted because their directed counterparts (v_2^0, v_3^0) and (v_2^2, v_1^2), respectively, belong to the solution.

The optimal solution to the SSTP instance consists of the first stage edge $(v_2, v_3, 0)$ and the first stage edge $(v_1, v_2, 2)$, yielding costs $4 + \frac{1}{2} \cdot 5 = \frac{13}{2}$. The value of the highlighted solution above is $W + 4 + \frac{5}{2} = W + \frac{13}{2}$, which is the value of an optimal solution to the original SSTP instance plus W.

The following proposition makes a statement about SSTP solutions whose first stages are not empty. The other solutions can be computed separately as stated in Section 4.1.

Proposition 5.3.1. *Let $(G, \mathcal{T})$ be an instance of SSTP with $G = (V, E, c^0)$ and $\mathcal{T} = \{(T^i, c^i, p^i) \mid 1 \leq i \leq K\}$, and $W = \sum_{e \in E} c^0(e)$. Let (G', T) be the instance of DSTP that results from the transformation of $(G, \mathcal{T})$ described above. A solution F to $(G, \mathcal{T})$ with $F^0 \neq \emptyset$ can be transformed into a solution F' to (G', T) with $v(F) = v(F') + W$ and vice versa.*

Proof. Let F be a solution to the SSTP instance, $V^0 \subseteq V$ the set of nodes that are incident to at least one edge in F^0, and $r' \in V^0$. For $1 \leq i \leq K$, V^i denotes the set of nodes that are incident to at least one edge in $F^i \setminus F^0$. Note that V^i might be empty for some $1 \leq i \leq K$. We discussed in Proposition 4.1.1 that F^i is the edge set of a tree for each $1 \leq i \leq K$. We denote by B^i the subtree of G with edge set F^i, assuming that r' is the root of these trees. Clearly, each tree contains r' because of $F^0 \subseteq F^i$ for each scenario i.

As the copies of G are interpreted as directed graphs in G', we define a mapping l that assigns a directed edge in E' to every element of F. Consider an edge (v, r') and $i \in \{1, \ldots, K\}$ with $(v, w, i) \in F$. Assume that the path from v to r in B^i passes the node r'. In this case, we denote by $l(v, w, i)$ the edge (w, v). If the path from v to r does not pass w, it can be enhanced to a path from w to r by appending the edge (w, v). In this case, $l(v, w, i)$ denotes the edge (v, w).

Consider the edge set

$$F' = \{(r, r'^0)\} \cup \{(v^0, v_i) \mid v \in V^0 \cap (V^i \cup T^i)\} \cup \{l(v, w, i) \mid (v, w) \in F^i\}$$

and a node $v^i \in T$. By construction, v is a terminal node of the i-th scenario of $\mathcal{T}$, i.e., $v \in T^i$. If $v = r'$, the edges $(r, r'^0), (r'^0, v^i) \in F'$ form a path from r to v^i. Otherwise, we need to find a path from r to v^i by utilizing first and second stage edges.

Because B^i is a Steiner tree for this terminal set, there exists a path from r' to v in B^i. Let $(r' = v_0, v_1, \ldots, v_\lambda = v)$ be the nodes of this path in the order they are passed by the path from r' to v. If there is a node v_j on this path such that (v_{j-1}, v_j) is bought in the first and (v_j, v_{j+1}) is bought in the second stage, we obtain a path from r to v^i by concatenating the paths from r to r', from r' to v_j^0, from v_j^0 to v_j^i and from v_j^i to v^i. All these paths exist by construction of F'. The function l makes sure that the edges of these paths are oriented the right way.

If no such v_j exists, v^i is either covered by first stage edges or the transition to the second stage edges occurs in r'. In the first case, a path from r to v^i can be found in a similar way as above by starting in r and passing r'^0 and v^0 to reach v^i directly. In the latter case, the path starts by using the edges $(r, r'^0), (r'^0, r'^i)$ and then following the path that corresponds to the edges of the path from r' to v in the second stage copy we just reached.

Thus, there exists a path from r to each terminal in T that only uses edges in F' without using redundant edges, and F' is therefore a solution for the DSTP instance. Its value is exactly W plus the value of F.

Now let F' be a solution to the DSTP instance (G', T). The set $F = \{(v, w, i) \mid (v, w) \in F'^i\}$ is a solution to the SSTP instance with $v(F') = v(F) + W$, which can be shown analogously. □

For an optimal SSTP solution F we find a DSTP solution F' with $v(F) = v(F') - W$. On the other hand, an DSTP solution F' can be used to construct an optimal SSTP solution F with $v(F') = v(F) + W$. Thus, the value of an optimal solution to the SSTP instance is exactly W plus the value of an optimal solution of a corresponding DSTP instance and optimal solutions for SSTP and DSTP can be transformed into each others.

We can, therefore, solve SSTP by transforming its instances into DSTP instances and solving them using the extended algorithm of Dreyfus and Wagner. The graph G' has $(K+1) \cdot n + 1$ nodes and $t^* := \sum_{i=1}^{K} \left|T^i\right|$ terminals. Solving SSTP with the described method requires running time $\mathcal{O}(((K+1) \cdot n + 1)^2 3^{t^*}) = \mathcal{O}((Kn)^2 3^{t^*})$, qualifying it as parameterized algorithm for SSTP.

In a very similar way, SPCSTP instances can be transformed into DPCSTP instances. We start with an SPCSTP instance $(G, \mathcal{T}, P)$. First, we apply the graph transformation as above to obtain a directed, weighted graph $G' = (V', E', c')$ without terminals. The resulting graph is again rooted at r, which will also act as the root of the DPCSTP instance. Second, we need to apply node profits for each $v \in V'$. The root and every node of the first stage copy of G are assigned a profit 0. Node v^i in the second stage copy G'^i is assigned profit $p^i P(v, i)$.

Solutions to the original SPCSTP instance and the resulting DPCSTP instance can be transformed into each others in the same way as we did with solutions to SSTP and DSTP.

This places the parameterized languages SSTP$^{|T^*|}$ and SPCSTP$^{|T^*|}$ in FPT, where T^* denotes the total number of terminals $\sum_{i=1}^{K} \left|T^i\right|$ for SSTP, or the total number of vertices with positive profit $\sum_{i=1}^{K} |\{v \in V \mid P(v,i) > 0\}|$ for SPCSTP. We summarize the results in the following theorem.

Theorem 5.3.1. *SSTP$^{|T^*|} \in$ FPT and SPCSTP$^{|T^*|} \in$ FPT.*

Chapter 6

Parameter: Number of Non-Terminals

In this chapter we investigate an algorithm for different Steiner tree problems parameterized by the number of non-terminals. A simple idea for STP is provided in Section 6.1, which is also applicable for its non-stochastic variants. We then try to carry this simple idea over to SSTP. We first describe a framework that is suitable for this task in Section 6.2. Some simple implementations of the framework are given in Section 6.3. This section also defines some notation that is also used in the final part. Section 6.4 provides more promising approaches that are inspired by Kruskal and Prim.

6.1 Algorithm for STP

The actual SSTP algorithm that we want to describe in this chapter is derived from a simple algorithm for STP. The algorithm exploits the fact that minimum Steiner trees in G are minimum spanning trees for a subgraph of G induced by a node subset, which is established by the following lemma.

Lemma 6.1.1. *Let $G = (V, E, c)$ be an undirected, weighted graph and $T \subseteq V$ a terminal set of G. Let $S = (V', E')$ be a minimum Steiner tree for T in G. S is a minimum spanning tree for $G[V']$.*

Proof. According to Proposition 3.1.1, S is a tree and thus a spanning tree for $G[V']$. Now assume that S is not a minimum spanning tree, i.e., there is a tree S' that spans V' and is cheaper than S. S' is also a Steiner tree for T in G. This contradicts our assumption that S is minimal. □

The set V' can be partitioned into the set of terminals T and the set of Steiner nodes. The terminal set is part of the input; only the set of non-terminals that are used as Steiner nodes is unknown. The set of Steiner nodes is always a subset of $N := V \setminus T$. Algorithm 6.1.1 is suitable for solving STP.

The function EnumerateNonTerminalSets simply enumerates the power set of N. It assumes that the current set of non-terminals V' is the set of Steiner nodes of a minimum Steiner tree. This is not possible if G limited to V' is not connected, so that it is safe to skip such subsets.

If $G[V']$ is yet connected, the function computes a minimum spanning tree for it. The value of this minimum spanning tree is compared to the value of the best edge set encountered so far. If it is lesser than the former minimum, the old edge set is replaced by the current one.

Theorem 6.1.1. *The above algorithm solves STP in running time $\mathcal{O}((|E| + |V| \log |V|)2^{|N|})$, $N = V \setminus T$.*

Proof. A minimum spanning tree for $G[V' \cup T]$, $V' \subseteq N$, is a tree that contains every terminal. It is thus a Steiner tree. We showed above that the computation of a minimum spanning tree for

Algorithm 6.1.1 Algorithm for STP parameterized by the number of non-terminals

Input: undirected, weighted graph $G = (V, E, c)$ and terminal set $T \subseteq V$
Output: edge set of a minimum Steiner tree for T in G

```
1: function EnumerateNonTerminalSets(G, T)
2:     result ← E
3:     for subset V' of N do
4:         if G[V'] is connected then
5:             F ← minimum spanning tree for G[V' ∪ T]
6:             if v(F) < v(result) then
7:                 result ← F
8:     return result
9: end function
```

the correct set of non-terminals yields a minimum Steiner tree. An optimal selection of Steiner nodes is tested because we enumerate the power set of $\mathcal{N}$, which contains every selection of Steiner nodes. In lines 6–7, a worse Steiner tree is always substituted by such a minimum Steiner tree. Therefore, the algorithm works correctly.

There are $2^{|N|}$ subsets of N. In a worst case scenario, a minimum spanning tree has to be computed for each of them. The computation of a minimum spanning tree for a graph with n nodes and m edges can be done in $\mathcal{O}(m + n \log n)$. $G[V' \cup T]$ has at most $|V|$ nodes and at most $|E|$ edges. This yields an overall running time of $\mathcal{O}((|E| + |V| \log |V|)2^{|N|})$. □

This algorithm places STP parameterized by the number of non-terminals in FPT. The approach is also applicable for DSTP, yielding $\text{STP}^{|N|}$ and $\text{DSTP}^{|N|} \in \text{FPT}$.

Note that it does not work for PCSTP because we do not know which vertices with positive profit are included in the optimal solution. Further enumeration of the nodes with positive profit would result in an algorithm whose running time cannot be parameterized solely by the number of non-terminals or, in this case, the number of vertices with zero profit.

6.2 Framework for SSTP Algorithms

The simple idea of enumerating the power set of $V \setminus T$ to solve the STP is now carried over to the SSTP. For this purpose, we first describe a simple framework that is shared among all algorithms proposed in the next sections.

Let $(G, \mathcal{T})$ be an SSTP instance with $G = (V, E, c^0)$ and $\mathcal{T} = \{(T^i, c^i, p^i) \mid 1 \leq i \leq K\}$. The set of non-terminals $(V \setminus T^1) \times \cdots \times (V \setminus T^K)$ is denoted by $\mathcal{N}$.

Algorithm 6.2.1 Framework for SSTP algorithms parameterized by the number of non-terminals

Input: undirected, weighted graph $G = (V, E, c^0)$, set of scenarios $\mathcal{T} = \{(T^i, c^i, p^i) \mid 1 \leq i \leq K\}$
Output: minimum stochastic Steiner tree for $\mathcal{T}$ in G

```
1: function EnumerateScenarioNonTerminals(G, T)
2:     result ← E × {0}
3:     for N ⊆ N do
4:         F ← FindEdgesBySteinerNodes(G, T, N)
5:         if v(F) < v(result) then
6:             result ← F
7:     return result
8: end function
```

The function FindEdgesBySteinerNodes is called for every selection of non-terminals for each scenario. It computes an optimal solution to the SSTP instance $(G, \mathcal{T})$ under the condition that the passed set of non-terminals is exactly the set of Steiner nodes.

The running time of EnumerateScenarioNonTerminals can be expressed as the product of the number of subsets of $(V \setminus T_1) \times \cdots \times (V \setminus T^K)$ and the running time of FindEdgesBySteinerNodes. The former can be formulated in terms of non-terminal counts, which is the desired parameter in this chapter. If there is an implementation of FindEdgesBySteinerNodes whose worst-case running time is polynomial in the length of the main part of the input, this algorithm places SSTP parameterized by the number of non-terminals in FPT.

Unfortunately, we did not find a concrete implementation for FindEdgesBySteinerNodes that we were able to prove correct with acceptable running time. In the following sections, however, we propose some implementations that might still be promising and lead to efficient algorithms.

We assume in the description of the different approaches that we already know an optimal selection of Steiner nodes for each scenario. For selections that do not qualify for optimal solutions, the result of suggested implementations of FindEdgesBySteinerNodes has non-optimal value or is not a valid solution at all. In the former case, the result is discarded immediately or at a later time as we only keep optimal solutions. In the latter case, the result can just be ignored. We can test whether the result is a valid solution in time $\mathcal{O}(K(|V| + |E|))$ with a depth first search per scenario.

6.3 Pure minimum spanning tree approach

We present some ideas that might be suitable to determine the edge set of a minimum stochastic Steiner tree when an optimal selection of Steiner trees for each scenario is given.

Let F be an optimal stochastic Steiner tree for the SSTP instance $(G, \mathcal{T})$ with $G = (V, E, c^0)$ and $\mathcal{T} = \{(T^i, c^i, p^i) \mid 1 \leq i \leq K\}$. Let F be a minimum stochastic Steiner tree for $\mathcal{T}$ in G and $\mathcal{N} = (N^1, \ldots, N^K)$ the selection of Steiner nodes for each scenario with respect to F, i.e., (V^i, F^i), $V^i = N^i \cup T^i$, is a Steiner tree for T^i in G. V^* denotes the set of nodes that are covered by every Steiner tree, i.e.,

$$V^* = \bigcap_{i=1}^{K} V^i.$$

Now consider the first stage edges F^0 of F. The edges are part of every Steiner tree (V^i, F^i), $1 \leq i \leq K$, because of $F^0 \subseteq F^i$. Thus, the nodes covered by F^0 are also part of every Steiner tree. Let V^0 denote the set of nodes that are covered by first stage edges of a minimum stochastic Steiner tree. Then clearly $V^0 \subseteq V^*$.

We showed in Lemma 6.1.1 that minimum Steiner trees in G are minimum spanning trees for an appropriate subgraph of G. The result can be carried over to the first stage edges of a minimum stochastic Steiner tree in G. Assume F^0 is not a minimum spanning tree for $G[V^0]$ with respect to its weight function c^0. Replacing F^0 with a minimum spanning tree of $G[V^0]$ would yield a better SSTP solution than F. Since F is optimal, the assumption that F^0 is not a minimum spanning tree must have been wrong.

Assume that not only an optimal $\mathcal{N}$ is given, but also an optimal V^0. We are now able to compute a minimum stochastic Steiner tree in three steps. First, we compute the edges F^0 of a minimum spanning tree for $G[V^0]$ with respect to c^0. Second, we set $c^i(e) = 0$ for each $e \in F^0$. Last, we compute the edges of a minimum spanning tree for $G[V^i]$ with respect to c^i, remove every edge that also belongs to F^0 and call the result F^i for each $1 \leq i \leq K$. The resulting F is a minimum stochastic Steiner tree for $\mathcal{T}$ in G.

A naive algorithm can try to guess V^0. Because of $V^0 \subseteq V^*$, enumerating the power set of V^* is the easiest way to find a suitable V^0. Unfortunately, V^* can be as big as V itself. This yields a running time of $\Omega(2^{|V|})$ for each call to FindEdgesBySteinerNodes. The naive algorithm is not suitable to place SSTP$^{|\mathcal{N}|}$ in FPT.

Another naive approach computes a minimum spanning forest for $G[V^*]$, i.e., a set of minimum spanning trees for each connected component of $G[V^*]$. The approach very optimistically assumes that one of the trees of the computed forest is the desired first stage tree. For each tree F^0 of the spanning forest, the second stage edges for scenario i are computed as minimum spanning trees for

$G[V^i]$ with respect to $c^i|_{E\setminus F^0}$. For an edge weight function $c : E \mapsto \mathbb{R}_0^+$ and $E' \subseteq E$, we denote by $c|_{E'}$ the function that results from f when re-mapping each element of $E \setminus E'$ to 0, i.e.,

$$c|_{E'}(e) = \begin{cases} f(e) & e \in E' \\ 0 & \text{otherwise} \end{cases}$$

for $e \in E$.

The weakness of this approach is that it considers the first stage costs of the edges in $G[V^*]$. In fact, we are not interested in the cheapest first stage edges, but rather in the edges that are most rewarding when buying them in the first instead of the second stage. A cheap first stage edge may be of no interest because there are only few scenarios that would profit from this edge.

The profit that may result from an edge that is added to the first stage cannot be computed without knowledge about the second stage edges. This drawback indicates that the decision about first stage edges has to be delayed until the second stage edges have been computed.

All of the following approaches assume that we start by computing the minimum spanning trees for the second stage of each scenario, i.e., for $G[V^i]$, $1 \leq i \leq K$. We only have to deal with the selection of appropriate first stage edges. For this purpose, we describe an operation on SSTP solutions that adds edges to, or removes edges from, the first stage edge set F^0.

Another approach is to select those edges for the first stage that are shared among several second stage edge sets F^i. If the first stage cost of an edge e is less then the aggregate cost of all scenarios that contain e, i.e.,

$$c^0(e) < \sum_{e^i \in F} p^i c^i(e),$$

the value of F can be reduced by removing the second stage edges e^i, $1 \leq i \leq K$ from F and instead adding the first stage edge e^0 to it. Repeated application of this operation might result in an F^0 that is not connected any more.

Perhaps, this problem can be circumvented by adjusting the edge weights such that minimum spanning trees are unique for each scenario. Dictating inflationary second stage costs, i.e.,

$$c^i(e) = a^i c^0(e)$$

for $1 \leq i \leq K$, $e \in E$ and some $a^i \in R^+$ may further reduce the problem. That is because inflationary second stage costs may lead to minimum spanning trees of $G[V^i]$ that are equal—or at least look similar—on $G[V^0]$.

6.4 Learning from Kruskal and Prim

The last two approaches are inspired by algorithms that are commonly used to compute minimum spanning trees, namely Kruskal's algorithm [13] and Prim's algorithm [16].

Let F be a solution to the SSTP instance $(G, \mathcal{T})$ such that every F^i is a tree. The function $a(\cdot)$, which is an abbreviation for "add first stage edge", receives G, the set of nodes V^i that each F^i has to cover, F itself and an edge e as arguments. It first adds the first stage edge e^0 to F. This produces one redundant edge per scenario. For each scenario i, the edge e has either been bought before, or the edge set F^i contains exactly one cycle after adding e because F^i is a tree. In the former case, we remove the edge e^i from F. In the latter case, we remove the most expensive second stage edge that lies on the cycle. The SSTP solution we obtain is the result of $a(G, (V^i)_{1\leq i\leq K}, F, e)$. Note that $a(G, \mathcal{T}, F, e)$ contains one first stage edge more than F itself.

Algorithm 6.4.1 can be used to compute $a(G, \mathcal{T}, F, e)$. Its running time is $\mathcal{O}(K(|V| + |E|)) = \mathcal{O}(K \cdot |E|)$ if F is represented as a two-dimensional, boolean array that allows adding and removing elements in $\mathcal{O}(1)$.

In a similar fashion, the function $d(\cdot)$ removes a first stage edge e from F. If F^i was free of cycles before the removal of e from the first stage, the resulting Steiner tree for scenario i is not connected any longer. Therefore, we need to add a second stage edge that re-connects the two connected components. Since we are interested in minimal solutions, we add the cheapest second

Algorithm 6.4.1 Add an edge as first stage edge to a given SSTP solution and remove redundant edges

Input: undirected, weighted graph $G = (V, E, c^0)$, set of scenarios $\mathcal{T} = \{(T^i, c^i, p^i) \mid 1 \leq i \leq K\}$, stochastic Steiner tree F for $\mathcal{T}$ in G, $e \in E$
Output: stochastic Steiner tree that results from adding e^0 to F and removing redundant edges

```
1: function AddFirstStageEdge(G, T, F, e)
2:     F ← F ∪ {e^0}
3:     for i = 1, ..., K do
4:         if e^i ∈ F then
5:             F ← F \ {e^i}
6:         else
7:             C ← cycle in F^i
8:             e' ← most expensive edge of C with respect to c^i
9:             F ← F ∪ {e'^i}
10:    return F
11: end function
```

stage edge for this purpose. Doing so for every scenario gives us the result of d. It has one first stage edge less than the solution F we started with.

We abbreviate $a(G, (V^i)_{1 \leq i \leq K}, F, e)$ and $d(G, (V^i)_{1 \leq i \leq K}, F, e)$ to $a(F, e)$ and $d(F, e)$, respectively, if G and V^i, $1 \leq i \leq K$ are evident.

Kruskal's algorithm finds minimum spanning trees by sorting the edges of a graph in ascending order and then successively adding the cheapest edge to the solution that does not introduce cycles. We try to re-use the idea in Algorithm 6.4.2 to find an optimal selection of first stage edges after minimum spanning trees for the second stages have been computed.

We start by sorting the edges of $G[V^*]$. The key used to sort them is the profit $c^*(e)$ yielded by moving an edge e into the first stage, i.e.,

$$c^*(e) = v(a(F, e)) - v(F).$$

While there are edges with positive value with respect to c^*, the most profitable edge e of them is added to the first stage by replacing F with $a(F, e)$.

Algorithm 6.4.2 Deduce first stage edges by traversing the edges ordered by ascending profit $c^*(\cdot)$

Input: undirected, weighted graph $G = (V, E, c^0)$, $V^i \subseteq V$, stochastic Steiner tree F such that (V^i, F^i) is a connected graph for $1 \leq i \leq K$
Output: refinement of F

```
1: function DeduceFirstStageKruskal(G, (V^i)_{1≤i≤K}, F)
2:     for edge e of G[V*] do
3:         F' ← AddFirstStageEdge(F, e)
4:         c*(e) ← v(F') − v(F)
5:     for edge e of G[V*], ordered by c*(·) do
6:         F' ← AddFirstStageEdge(F, e)
7:         if v(F') < v(F) then
8:             F ← F'
9:     return F
10: end function
```

An example implementation is given in Algorithm 6.4.2. The comparison in line 7 evaluates to true for the edge that is considered first if there is at least one edge e with positive profit $c^*(e)$.

Unfortunately, the profit an edge produces depends on the edges that have been moved to the first stage before. Therefore, the algorithm listed above requires some modification. We need to

re-calculate the value of $c^*(\cdot)$ for each edge after F has been modified. This renders the sorting step redundant, as the order potentially changes after each execution of line 8. Instead of ordering, we simply look for the currently most profitable edge in an unordered edge set.

The loop is repeated until there is no edge with positive profit left. This method lets us skip the comparison in line 7.

Line 8 is executed at most $|V|$ times. Thus, we need to re-calculate $c^*(\cdot)$ at most $|V|$ times. The value of a solution F can be calculated in time $\mathcal{O}(K \cdot |E|)$. This yields an overall running time of $\mathcal{O}(|V|\,(|V| + K \cdot (|V| + |E|) + K \cdot |E|)) = \mathcal{O}(K \cdot |V| \cdot |E|)$.

We were not able to prove the connectivity of the first stage of a solution found with this method. Nor are we able to give reasons why the algorithm does not get caught in a local minimum, even if it is connected.

The second problem can perhaps be alleviated by trying different edge orderings. Instead of traversing them in ascending order, traversing them in descending order may also lead to good results if we allow the function $a(\cdot)$ to remove previously added first stage edges. Another alternative is to start with $G[V^*]$ and then selectively removing first stage edges by replacing F with $d(F, e)$, where e is the currently most unprofitable edge.

The first problem can be avoided completely by allowing the first stage to consist of more than one connected component. A stochastic Steiner tree F of this kind would only have to provide paths between every pair of terminals $t_1, t_2 \in T^i$ that only use edges of F^i.

Another way to circumvent the first problem is to make sure that the first stage edges are connected all the time. While the algorithm above in its original form may remind the reader of Kruskal's algorithm to find a minimum spanning tree, the next one, i.e., Algorithm 6.4.3, is inspired by Prim's algorithm.

Algorithm 6.4.3 Deduce first stage edges by simulating Prim's algorithm; edge profit is re-calculated after each alteration of the solution

Input: undirected, weighted graph $G = (V, E, c^0)$, $V^i \subseteq V$, stochastic Steiner tree F such that (V^i, F^i) is a connected graph for $1 \leq i \leq K$, node r of $G[V^*]$
Output: refinement of F

```
1: function DeduceFirstStagePrim(G, (V^i)_{1≤i≤K}, F, r)
2:     e ← edge (r, w) such that c*((r, w)) is minimal
3:     while c*(e) > 0 do
4:         F ← AddFirstStageEdge(F, e)
5:         e ← edge e' such that F^0 ∪ {e'} is a tree and c*(e') is minimal
6:     return F
7: end function
```

Again, we start with the computation of the minimum spanning trees for $G[V^i]$ with respect to c^i for each scenario i before the call to DeduceFirstStagePrim. We choose an arbitrary node $r \in V^*$ that serves as a virtual root for the first stage tree F^0 that we are about to compute. If there is no profitable edge with respect to c^* that is incident to r, the solution is returned. Otherwise, we add the edge e that is most profitable with respect to c^* to F^0 using $a(F, e)$.

Then, as long as there are profitable edges adjacent to an edge in F^0, we add the most profitable of them that does not induce a cycle to the first stage. As soon as there are no such edges left, the computation is aborted and the current F is returned. Like above, $c^*(\cdot)$ has to be re-computed after every modification of F^0.

Algorithm 6.4.3 has to be called for each $r \in V^*$ as long as we do not know which vertices are covered by first stage edges. After each execution of line 4, $c^*(\cdot)$ has to be re-calculated. The loop spanning lines 3–5 is exited if there is no edge left in $G[V^*]$ that can be added to F^0 without introducing cycles or new connected components.

Lines 4 and 5 are executed at most $\mathcal{O}(|V|)$ times. Otherwise, F^0 would not be a tree any more. Line 4 requires time $\mathcal{O}(K \cdot |E|)$. Line 5 can be executed in time $\mathcal{O}(K \cdot |E|^2)$. Thus, the overall running time is $\mathcal{O}(K \cdot |V| \cdot |E|^2)$.

This approach might get stuck in solutions that are locally optimal, because connecting several profitable first stage edges that are not adjacent might require selecting unprofitable edges first. One possible way to avoid this problem is the adjustment of the exit condition in line 3. Adding edges that are not profitable by themselves causes re-calculation of $c^*(\cdot)$. This may result in profitable edges that have not been profitable before.

We could dictate inflationary second stage costs. This could possibly lead to optimal SSTP solutions F whose first stages F^0 only consist of edges that are profitable by themselves. They would not have to be connected by unprofitable edges to escape local optima.

If all this does not help, we might allow the function AddFirstStageEdge to remove first stage edges that have been added previously. The definition of this operation given above forces us to remove second stage edges that lie on cycles.

Although we are not able to prove the correctness of one of the approaches above, we believe that it is possible. This leads us to the following conjecture.

Conjecture 6.4.1. $SSTP^{|\mathcal{N}|} \in FPT$

Chapter 7

Conclusion and Outlook

7.1 Parameter: Number of Terminals

In Chapter 5 we described how to exploit the ideas of Dreyfus and Wagner to solve the variants of STP. The original form of this early STP algorithm requires running time $\mathcal{O}(n^2 3^{|T|})$, if n is the number of vertices and T the terminal set. We showed how these results can be used to solve the DSTP and PCSTP. We are also able to solve SSTP and SPCSTP by reducing them to DSTP and DPCSTP, respectively.

In 2007, Björklund et al. [1] showed that this algorithm can be enhanced to compute a solution in time $\mathcal{O}(n^2 2^{|T|})$. This is achieved by using a method called fast subset convolution. Further investigation might show that these results are also applicable for PCSTP and DSTP. An application for DSTP would also result in a faster algorithm for SSTP. We conjecture that this is the case.

If this fails, the work of Fuchs et al. [9] might be able to solve the discussed problems in less time. The paper introduced an algorithm solving STP and DSTP in time $(2+\delta)^{|T|} n^{\mathcal{O}(1/(\delta/\ln(1/\delta))^{\zeta})}$ for any $0 < \delta < 1$ and $\frac{1}{2} < \zeta < 1$. It can be used for SSTP by reducing it to DSTP first and may also be applicable to PCSTP.

7.2 Parameter: Number of Non-Terminals

In Chapter 6 we discussed an algorithm for STP with running time $\mathcal{O}((n+m)2^{n-|T|})$, if n is the number of vertices, m the number of edges and T the terminal set. This algorithm is probably eligible to solve SSTP with minor modifications. We also proposed a framework for this purpose, as well as some implementations. Because of time restrictions, we were not able to prove the correctness.

7.3 More parameters

So far, the number of terminals and the number of non-terminals have been used as parameters to place SSTP in FPT. There are other parameterizations that might be of interest. In practice, parameterizing SSTP with the number of first stage edges might be of interest. Other possible parameters include the sum of weights of first stage edges, the overall size of an optimal solution expressed in edge number or weight, and the number of scenarios. If we define the terminal core as the cut of all terminal sets, the size of the core or the number of terminals per scenario that are not part of the core might also be an interesting candidate.

The original STP can be solved in linear time if the tree width is *very* small, i.e., at most 2. However, Boekler [2] suggested a proof that the decision variant of SSTP is NP-hard on graphs with treewidths of at most 3. Chimani, Mutzel and Zey [4] proposed an algorithm that solves STP in time $\mathcal{O}(B^2_{\mathrm{tw}+2} \cdot \mathrm{tw} \cdot n)$, where n is the number of vertices and tw the treewidth of the input

graph. This algorithm places STP^{tw} in FPT. SSTP algorithms with the treewidth of the input graph are also imaginable.

List of Figures

List of Algorithms

Index

Bibliography

[1] Björklund, A., Husfeldt, T., Kaski, P., and Koivisto, M. Fourier Meets Möbius: Fast Subset Convolution. In *STOC* (2007), ACM, pp. 67–74.

[2] Boekler, F. Das Stochastische Steinerbaumproblem auf Serien-Parallelen Graphen. Diploma thesis, Department of Computer Science, Technische Universität Dortmund, 2012.

[3] Bomze, I. M., Chimani, M., Jünger, M., Ljubić, I., Mutzel, P., and Zey, B. Solving two-stage stochastic Steiner tree problems by two-stage branch-and-cut. In *ISAAC (1)* (2010), pp. 427–439.

[4] Chimani, M., Mutzel, P., and Zey, B. Improved Steiner Tree Algorithms for Bounded Treewidth. In *IWOCA* (2011), vol. 7056 of *Lecture Notes in Computer Science*, Springer, pp. 374–386.

[5] Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

[6] Downey, R. G., and Fellows, M. R. *Parameterized Complexity*. Springer, 1999.

[7] Dreyfus, S. E., and Wagner, R. A. The steiner problem in graphs. *Networks 1*, 3 (1971), 195–207.

[8] Floyd, R. W. Algorithm 97: Shortest path. *Communications of the ACM 5*, 6 (1962), 345.

[9] Fuchs, B., Kern, W., Mölle, D., Richter, S., Rossmanith, P., and Wang, X. Dynamic Programming for Minimum Steiner Trees. *Theory Comput. Syst. 41*, 3 (2007), 493–500.

[10] Gupta, A., and Pál, M. Stochastic steiner trees without a root. In *ICALP* (2005), vol. 3580 of *Lecture Notes in Computer Science*, Springer, pp. 1051–1063.

[11] Gupta, A., Pál, M., Ravi, R., and Sinha, A. Boosted sampling: approximation algorithms for stochastic optimization. In *STOC* (2004), ACM, pp. 417–426.

[12] Hwang, F., Richards, D., and Winter, P. *The Steiner tree problem*, vol. 53 of *Annals of discrete mathematics*. North-Holland, 1992.

[13] Kruskal, J. B. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society 7*, 1 (1956), 48–50.

[14] Niedermeier, R. Ubiquitous parameterization - invitation to fixed-parameter algorithms. In *MFCS* (2004), vol. 3153 of *Lecture Notes in Computer Science*, Springer, pp. 84–103.

[15] Papadimitriou, C. H., and Steiglitz, K. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[16] Prim, R. C. Shortest connection networks and some generalizations. *Bell System Technology Journal 36* (1957), 1389–1401.

[17] Swamy, C., and Shmoys, D. B. Approximation algorithms for 2-stage stochastic optimization problems. In *FSTTCS* (2006), vol. 4337 of *Lecture Notes in Computer Science*, Springer, pp. 5–19.

[18] Wegener, I. *Complexity Theory: Exploring the Limits of Efficient Algorithms.* Springer, 2005.