Approximation Algorithms For Minimum-Cost Low-Degree Subgraphs

Jochen Könemann

A Dissertation submitted in partial fulfilment of the requirements for the Degree of Doctor of Philosophy at the Graduate School of Industrial Administration, Carnegie Mellon University

Graduate School of Industrial Administration
Carnegie Mellon University
Schenley Park
Pittsburgh, PA 15213, U.S.A.

(Dissertation for the Degree of Doctor of Philosophy in Algorithms, Combinatorics and Optimization presented at Carnegie Mellon University in 2003)

Abstract

In this thesis we address problems in minimum-cost low-degree network design. In the design of communication networks we often face the problem of building a network connecting a large number of end-hosts. Available hardware or software often imposes additional restrictions on the topology of the network.

One example for such a requirement arises in the area of computer networks: We are given a number of client hosts in a network that communicate with each other using a certain communication protocol that limits the number of simultaneously open connections for each host. In graph theoretic terms, where hosts correspond to nodes and network connections are represented by edges, this restriction naturally translates to a bound on the maximum node-degree of the constructed network.

In this thesis we first address the problem of finding a spanning tree T for a given undirected graph $G = (V, E)$ with maximum node-degree at most a given parameter $B > 1$. Among all such trees, we aim to find one that minimizes the total edge-cost for a given cost function c on the edges of G. We develop an algorithm based on Lagrangean relaxation. We show how to compute a spanning tree with maximum node-degree $O(B + \log(n))$ and total cost at most a constant factor worse than the cost of an optimum degree-B-bounded spanning tree in an n-node network.

We present a second algorithm that is also based on ideas from Lagrangean relaxation but does not rely on computing a solution to a linear program. The new method can handle non-uniform degree-bounds, i.e. we are given integers $B_v > 1$ for all $v \in V$ and the degree of each node $v \in V$ is constrained to be at most B_v in any feasible solution. The algorithm uses a repeated application of Kruskal's MST algorithm interleaved with a combinatorial update of approximate Lagrangean node-multipliers maintained by the algorithm. These updates cause subsequent repetitions of the spanning tree algorithm to run for longer and longer times, leading to overall progress and a proof of the performance guarantee.

Finally, we show how to extend the second algorithm to the case of Steiner trees where we use a primal-dual approximation algorithm due to Agrawal, Klein, and Ravi in place of Kruskal's minimum-cost spanning tree algorithm. The algorithm computes a Steiner tree of maximum degree $O(B + \log n)$ and total cost that is within a constant factor of that of a minimum-cost Steiner tree whose maximum degree is bounded by B. However, this method has quasi-polynomial running time.

This dissertation is dedicated to my family
Lydia, Wolfgang, and Katja Könemann

Contents

Chapter 1

Introduction

In algorithm design, we are concerned with devising efficient algorithms for solving a particular problem. Here, a natural measure for efficiency is the algorithm's running time: We are interested in the number of atomic operations that the algorithm performs in order to compute a solution to an input instance of a given problem. More precisely, we state the time complexity of an algorithm as a function of the size of its input. This function then allows us to estimate the running time of our algorithm as the sizes of the considered problem instances vary.

This said, an obvious goal is to find an algorithm for a posed problem whose running time can be stated as a low-order polynomial in the input size. We refer to these procedures as *polynomial-time* algorithms. In general, the lower the order of this polynomial the larger problem instances we can hope to solve. Unfortunately, there is a large number of optimization and decision problems which are unlikely to be solved in polynomial-time. There is a rich body of work devoted to the classification of algorithmic problems into *complexity classes*. We refer the reader to the book of Papadimitriou [56] for further information on complexity theory.

The relevant complexity classes for this work are `P` and `NP`. Roughly, a problem A is in `P` if there is a polynomial time algorithm that computes an exact solution for any given input instance of A. The class `NP` contains `P` but also contains a set of `NP`-*complete* problems, which are unlikely to be in `P`. Currently, we do not know of the existence of a polynomial-time algorithm for any `NP`-complete decision problem. Moreover, it is a consequence of the definition of completeness that the existence of a polynomial-time procedure for any `NP`-complete problem would imply that all problems in `NP` admit polynomial-time algorithms as well.

The reasons that determine whether a problem is in `P` or whether it is `NP`-complete often lie in the combinatorial structure that it exhibits. Designing an algorithm for a problem in `P` requires unraveling and characterizing its combinatorial properties as a first step. We then exploit the uncovered structure and develop an algorithm. In this thesis, we will be concerned with `NP`-hard problems. Even though these problems are unlikely to admit polynomial-time algorithms, many of them still exhibit a rich combinatorial structure that we can use.

What can we do once a problem has been proven to be `NP`-hard? A natural immediate idea is to settle for suboptimal, *practical* solutions to a given problem that can be computed in polynomial time. In fact, there is a vast body of literature on so-called *heuristic* algorithms for a large number of different applications (some examples are the Steiner tree problem [34], graph partitioning [19, 32, 36, 38, 39, 65], and the traveling sales person problem [9, 31, 48, 50, 60]). The main point of criticism with these heuristic algorithms is however, that there is no guarantee on the quality of the computed solution. In fact, for each given heuristic there are usually examples on which it performs poorly: The computed solution has a value which deviates from the optimum solution value by a large factor.

In the following sections we define and introduce the fundamental concepts underlying our work. We first introduce *approximation algorithms* which address the main weakness of heuristics of not being able to bound the quality of the computed solution. Subsequently, in Section 1.1.1 we define the notion of bicriteria optimization since the problems addressed in our work will all have two competing optimization criteria. In Section 1.2 we give a brief overview of network design with a special focus on finding low-degree subgraphs. We end the introduction with a road-map for this thesis.

1.1 APPROXIMATION ALGORITHMS

The field of approximation algorithms was primarily founded to alleviate the above deficiency of heuristics. Approximation algorithms are heuristic algorithms supplied with an upper bound on the worst case ratio between the value of any approximate solution and that of an optimum solution. This thesis is about approximation algorithms for a certain class of network design problems. We now introduce the notion of approximation algorithms formally. Our notation will loosely follow the conventions used in the recent book by Vazirani [63].

For an $\mathtt{NP}$-optimization problem Π, let $\mathcal{I}_\Pi$ be the set of all input instances. For each $I \in \mathcal{I}_\Pi$ we are then given a set $\mathcal{S}_I$ of feasible solutions to I. Moreover, we have an objective function $c_I : \mathcal{S}_I \to \mathbb{R}^+$ that assigns an objective function value to each feasible solution to I. An approximation algorithm $\mathcal{A}$ for problem Π takes an arbitrary instance $I \in \mathcal{I}_\Pi$ and computes a feasible solution $\mathtt{APX}_I \in \mathcal{S}_I$ in time polynomial in the size of the input instance. Without loss of generality, assume that Π is a minimization problem.

In the following we use $\mathtt{apx}_I$ as an abbreviation for $c(\mathtt{APX}_I)$, i.e. $\mathtt{apx}_I$ is the objective function value of the approximate solution $\mathtt{APX}_I$. Similarly, we let $\mathtt{OPT}_I$ be an optimum solution to instance I and we also let $\mathtt{opt}_I$ be its objective function value. We then say, that algorithm $\mathcal{A}$ is an α-approximation if

$$\max_{I \in \mathcal{I}_\Pi} \frac{\mathtt{apx}_I}{\mathtt{opt}_I} \leq \alpha.$$

The task of developing an approximation algorithm for an optimization problem Π is much like developing an exact algorithm for a problem in $\mathtt{P}$. We need to exploit the combinatorial structure underlying Π and use this to prove that the value of any solution that we compute is *close* to that of any optimum solution. The way in which this is done can be roughly summarized as follows: We first use the structure of any solution to a given instance to compute a lower-bound on the objective function value of any solution. In a second step, we then use this lower-bound and construct an approximate solution whose value is close to the lower-bound and hence also close to the optimum solution value.

For optimization problems in $\mathtt{P}$, we often know from complexity theory that there is a fixed degree polynomial $p(n)$ of the input size n such that no algorithm can exist that solves this problem exactly and whose running is $o(p(n))$. On the other hand, many problems which are in $\mathtt{NP}$ have been characterized according to their *approximability*. There are optimization problems which admit an $(1+\epsilon)$-approximation algorithm whose running is polynomial in the input size and in $1/\epsilon$ for any fixed ϵ. We refer to such an algorithm as a *fully-polynomial-time approximation scheme* (FPTAS).

Inapproximability results generally show for a given problem Π that there is a *hardness factor* $f(n)$ such that no $o(f(n))$-approximation exists assuming that $\mathtt{NP} \neq \mathtt{P}$. Problems which do not admit an FPTAS can be classified according to their hardness-factors into the following three main classes:

1. $f(n) = O(1)$
2. $f(n) = O(\log n)$
3. $f(n) = O(n^\epsilon)$ for a fixed $\epsilon > 0$.

Apart from the theoretical desire of having an approximation ratio, i.e. a fixed upper bound on the ratio between approximate and optimum solution values, there is hope that approximation algorithm design leads to the discovery of new properties of a given problem that in turn lead to better practical algorithms. It is often the case, for example, that approximation algorithms perform much better on practical instances than what their approximation ratio promises (see e.g. [37, 68]).

1.1.1 Bicriteria approximation

The problems considered in this thesis fall into the class of *bicriteria optimization problems.* In a bicriteria optimization problem Π we are given two objective functions c_I^1 and c_I^2 that assign values to a solution $S \in \mathcal{S}_I$ for instance I. We are now asked to find a solution that optimizes the value of both functions. For simplicity, let us assume from now on that we are asked to minimize the value of both objectives.

Usually the two objective functions interfere with each other in that lowering the value of one objective might entail raising the other one. One approach for this kind of problem often compute the minimum-cost solution under a linear combination of the two cost measures [51, 58]. However, in the case of very disparate objectives these techniques usually do not produce useful solutions.

Our formulation assumes that one objective, say c_I^2, is accompanied by a budget B. Our task then is to compute a feasible solution $S \in \mathcal{S}_I$ such that $c_I^1(S)$ is as small as possible and $c_I^2(S) \leq B$.

We now extend the definition of approximation ratio to bicriteria optimization problems. Let $(\Pi, c^1, c^2, \mathcal{I}, B)$ be a bicriteria optimization problem where B is a budget on the c_I^2-value of any solution. Let $\mathcal{A}$ be an approximation algorithm for this problem and denote by $\mathtt{APX}_I \in \mathcal{S}_I$ the solution computed by $\mathcal{A}$. As before, we use the abbreviations $\mathtt{apx}_I^1$ and $\mathtt{apx}_I^2$ for $c_I^1(\mathtt{APX}_I)$ and $c_I^2(\mathtt{APX}_I)$. In addition, we use $\mathtt{opt}_I^1$ to denote the minimum c_I^1 value of any feasible solution $S \in \mathcal{S}_I$ that satisfies $c_I^2(S) \leq B$. We then say that an algorithm $\mathcal{A}$ is an (α, β)-approximation for the given problem if

$$\max_{I \in \mathcal{I}} \frac{\mathtt{apx}_I^1}{\mathtt{opt}_I^1} \leq \alpha$$

and $\mathtt{apx}_I^2 \leq \beta \cdot B$ for all $I \in \mathcal{I}$.

This way of formulating bicriteria optimization problems was first formalized in [51, 58]. Among other things, the authors prove that working with the budgeted formulation does not lead to a reduction in generality. Specifically, we can use a given (α, β)-approximation for the budgeted version of a bicriteria problem as black box to design a $\max\{\alpha, \beta\}$-approximation for the problem of minimizing a linear combination of c_I^1 and c_I^2.

1.2 LOW-DEGREE NETWORK DESIGN

The problems considered in this thesis can broadly be classified as *low-degree network design* problems. In a generic instance of a low-degree network design problem the goal would be to find a subgraph of a given graph that satisfies certain connectivity properties and at the same time obeys given degree requirements at the vertices.

Low-degree network design problems arise naturally in communication network design where one wants connectivity of a given telecommunication system and the degree constraints reflect the goal of decentralizing the total system. The following section provides an illustrative example from the realm of computer networks.

1.2.1 An example: Multicast routing

Consider the following scenario: We are given a computer network connecting a server to a multitude of client hosts using an infrastructure of switching nodes. An application that runs on the server would like to send the same information from the server to each of the client hosts. A few examples for such applications are:

Distributed databases A data-item is replicated a number of times and stored at various sites throughout the network. When one copy the data-item is changed all other replicas need to be updated (See e.g. [4, 14]).

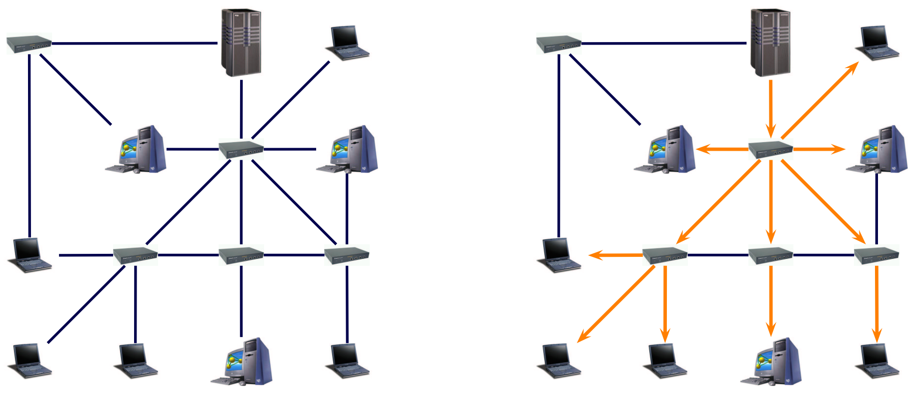

Figure 1.1: The two figures show a computer network that connects a server node at the top to a set of client hosts using intermediate switches. The right side shows a tree that connects the server to all clients. The directions of the arrows indicate how information is forwarded through the network.

Video distribution over IP The central server runs a video distribution application that needs to transmit a continuous video stream to all client applications (See e.g. [6]).

How can we transmit the information efficiently? In the obvious *unicast* solution we open a separate connection between the server and each host. This entails that if there are k clients in the network, the information that is stored at the server has to be transmitted k times.

Multicasting [13, 12] has been developed to cope with the deficiencies of the mentioned unicast approach. In a nutshell, in multicasting, we compute a spanning tree in the underlying network that connects the server node and all client hosts. The server then transmits the data along all its outgoing links in the computed spanning tree. An intermediate node receives incoming data along its *parent* link in the tree and copies it to each of its descendants. Figure 1.1 shows an example network and a possible tree along which data is forwarded. It is obvious that this method reduces the total network load when compared to the naive unicast solution.

Observe that the number of neighbors of a node in the computed spanning tree is proportional to the computational work that the node has to accomplish. In other words a node with high node-degree will have to duplicate and forward a large amount of data. In order to distribute load evenly among the hosts in the network we could thus try to bound the maximum degree of any node in the multicast spanning tree. The connection between node-degree in multicast trees and node performance has been considered by Bauer and Varma [5] and more recently by Chu et al. [10].

While spanning trees in a network can be computed efficiently, it is NP-hard to compute a spanning tree whose maximum degree is as small as possible. The following section surveys the known work on low-degree network design in the unweighted and weighted settings.

1.2.2 Algorithms for low-degree network design: Previous work

The previous section motivates a fundamental graph-theoretic question in the realm of low-degree network design: Compute an edge-subgraph of a given graph $G = (V, E)$ that has minimum maximum degree and in addition obeys certain connectivity properties.

Previous work: Unweighted problems The simplest to formulate problem in the area of unweighted low-degree network design is probably that of computing a spanning tree of minimum degree.

While the problem statement is deceptively simple, it is NP-complete to decide whether a given graph has a spanning tree of maximum degree B for an integer $B > 1$ [25]: Asking whether a given graph G has a spanning tree of maximum degree 2 is equivalent to asking whether there is a Hamiltonian path in G.

Furthermore, Gavish [26] solved the minimum-degree spanning tree problem exactly (albeit of course not in polynomial time) using integer programming and Lagrangean relaxation techniques.

The first to develop approximation algorithms for the minimum-degree spanning tree problem were Fürer and Raghavachari [21] who gave an $O(\log |V|)$ approximation. Their algorithm generalizes to directed graphs where spanning trees are replaced by *arborescences*: An arborescence is a rooted subgraph of the original graph such that its underlying undirected graph is a tree and where each edge is directed away from the root. The work in [21] generalizes to the problem of finding an arborescence of minimum maximum out-degree with approximation ratio $O(\log |V|)$.

Agrawal, Klein, and Ravi [1] were the first to develop a polynomial-time algorithm to find an approximate Steiner tree (i.e. a tree spanning a given set $R \subseteq V$ of required vertices) of minimum maximum vertex degree. The approximation factor of this algorithm is $O(\log |R|)$.

Subsequently, Fürer and Raghavachari improve upon their previous result and show how to compute a spanning tree that has degree within an additive one of the minimum possible degree [22]. In the same paper, the authors also give a local search algorithm for the MDST problem. This approach yields a tree with maximum degree at most $b\Delta^* + \log_b |V|$ for any constant $b > 1$ where Δ^* is the minimum possible maximum degree of any spanning tree of G. The algorithm in [22] and its performance guarantees extend to the Steiner case and hence improves upon [1].

Recently, Krishnan and Raghavachari [46] improve [21] in the directed setting by giving a quasi-polynomial time algorithm that computes an out-arborescence with maximum out-degree at most $O(B + \log |V|)$.

A further generalization of the above minimum-degree Steiner tree problem is the minimum-degree f-join problem [29]. Here, in addition to an undirected graph $G = (V, E)$, we are also given a *proper* cut function f that specifies connectivity requirements between node sets. Proper functions have range $\{0, 1\}$ and satisfy

- $f(V) = 0$,
- f is symmetric, i.e. $f(S) = f(V - S)$ for all $S \subseteq V$, and
- for all pairs of disjoint sets $A, B \subseteq V$, we must have $f(A \cup B) \leq \max\{f(A), f(B)\}$.

Let $\delta(S)$ denote the set of edges which have one endpoint in S and one in $V \setminus S$. We now require any solution to the minimum-degree f-join problem to have an edge from all cuts $\delta(S)$ with $f(S) = 1$.

Ravi, Raghavachari and Klein [59] give a quasi-polynomial approximation algorithm that computes a feasible solution for a given f-join problem with degree at most $O(\texttt{opt} + \log |V|)$ where opt is the minimum possible degree of any feasible f-join. In [59], the authors also develop a quasi-polynomial time algorithm to compute 2-edge-connected spanning subgraph of maximum degree $b\Delta^* + O(\log |V|)$ for any $b > 1$ where Δ^* denotes the minimum maximum degree of any 2-edge connected spanning subgraph.

Previous work: Weighted problems Special cases of the problem that are solvable exactly in polynomial time have been exhibited by Lawler [49, 7] who showed that Edmond's matroid intersection algorithm [17] can be used to compute a spanning tree of minimum cost such that the degrees of the nodes of an independent set $I \subseteq V$ are bounded by a given parameter $B > 0$. The special case of the problem with only one vertex-degree constraint has been considered by Gabow [23], Gabow and Tarjan [24], and by Glover and Klingman [27].

Fischer [20] noted that the local-search procedure by Fürer and Raghavachari from [22] can be adapted to find a minimum-cost spanning tree with maximum degree $b\Delta^* + \log_b |V|$ for any constant $b > 1$ where Δ^* is the minimum maximum degree of any minimum-cost spanning tree.

Ravi et al. [58] are the first to address the problem of computing minimum-cost degree-bounded trees: For a given parameter $B > 1$ and a non-negative cost function c on the edges of G, the authors show how to compute a spanning tree T of maximum degree $O(B \log |V|)$ and total cost at most $O(\log |V|)\, \mathtt{opt}_B$. Here $\mathtt{opt}_B$ denotes the minimum cost of any degree-B-bounded spanning tree of G. The authors also generalize their ideas to Steiner trees, generalized Steiner forests and the node-weighted case. Klein and Ravi [42] show that the $O(\log |V|)$-approximation algorithm is essentially best possible for the node-weighted case (via reductions from the set covering problem).

Fekete et al. [18] consider the case where edge-weights in the underlying graph are induced by a tree metric: In addition to the undirected graph $G = (V, E)$ we are also given a spanning tree T of G. Each edge e in T has a given weight w_e and all edges $e' = (u, v)$ that are not in the tree have weight equal to the weight of unique u, v-path in T. Given a degree bound $B_v \geq 2$ for each node $v \in V$, the authors show how to compute a spanning tree that obeys all degree constraints exactly and whose weight is at most the weight of T times

$$2 - \min_{v:\deg_T(v)>2} \frac{B_v - 2}{\deg_T(v) - 2}$$

where $\deg_T(v)$ denotes the degree of node v in tree T.

The problem of computing a minimum-weight spanning tree of maximum node-degree B for a given parameter $B > 1$ remains $\mathtt{NP}$-hard even if the nodes of our graph correspond to points in a given Euclidean metric space and the weight of an edge (u, v) is given by the Euclidean distance between the points corresponding to u and v, respectively [25, 35]. Following from a reduction from the Hamilton Path problem it is clear that the problem is $\mathtt{NP}$-hard in the special case of $B = 2$. Papadimitriou and Vazirani [57] proved that the problem remains hard even in the special case of $B = 3$ and conjectured that the case $B = 4$ is also $\mathtt{NP}$-hard.

On the positive side, Papadimitriou and Vazirani show in [57] that any minimum-weight spanning tree in $\mathbb{R}^2$ has maximum node degree at most 5 if the given points have integer coordinates. Monma and Suri [53] extended this result to points in $\mathbb{R}^2$ with arbitrary coordinates.

Khuller, Raghavachari, and Young [41] showed how to compute a spanning tree in $\mathbb{R}^2$ of maximum degree at most 3 and of total weight at most 1.5 times the weight of a minimum-weight spanning tree of the given points. They also give an algorithm to compute a spanning tree of weight at most 1.25 the weight of a minimum-weight spanning tree whose maximum degree is bounded by 4. Their results generalize to Euclidean spaces of higher dimensions where the authors show how to compute a spanning tree of maximum degree 3 and total weight at most $5/3$ times the weight of a minimum-weight spanning tree.

Finally, computational results concerning the degree-constrained minimum spanning tree can be found in [54, 61, 64]. We also refer the reader to an excellent survey on low-degree network design due to Raghavachari in Chapter 7 of [33].

1.3 NOTATIONAL CONVENTIONS

Most of the presentation in this thesis will use standard graph theoretical notation. For a general introduction to graph theory we refer the reader to one of the standard texts in the area (see e.g. [15, 66]).

All results in this work assume an n-node and m-edge undirected graph $G = (V, E)$. We denote an edge between vertices u and v by uv or by (u, v) interchangeably.

For a subgraph H of G, we will use $V[H]$ and $E[H]$ to denote the nodes and edges in H. We will also use $\delta_H(v)$ to denote the set of edges in E that are incident to node v in graph H. We then let $\deg_H(v)$ to denote the degree of node v in H, i.e.

$$\deg_H(v) = |\delta_H(v)|.$$

Furthermore, we will use $\Delta(H)$ to denote the maximum node degree of any node in H, i.e.

$$\Delta(H) = \max_{v \in V[H]} \deg_H(v).$$

Finally, for a graph $G = (V, E)$ and an edge $e \notin E$, we let $G + e$ be the graph obtained from adding e to E, i.e. $G + e = (V, E \cup \{e\})$.

1.4 THIS WORK

Our contributions can be grouped into three parts each of which finds place in a separate chapter.

1.4.1 Part 1: Minimum-cost degree-bounded spanning trees via Lagrangean relaxation

In Chapter 3 we address the minimum-cost degree-bounded spanning tree problem where we are given an undirected graph $G = (V, E)$, non-negative costs c_e for all edges $e \in E$, and a parameter $B \geq 2$. The goal is to find a tree T of minimum cost and maximum node-degree B. We prove the following theorem:

Theorem 1 (see [43, 44]) *There is a polynomial-time approximation algorithm that, given a graph $G = (V, E)$, a nonnegative cost function $c : E \to \mathbb{R}^+$, a degree bound $B \geq 2$, and parameters $\omega > 0$ and $b > 1$, computes a spanning tree T such that*

1. $\Delta(T) \leq (1 + \omega)bB + \log_b n$, *and*
2. $c(T) < (1 + 1/\omega)\, \mathtt{opt}_B$.

The techniques used in this work are linear programming based. We first give a natural integer programming formulation for the problem where the constraint set consists of a full description of the spanning tree polyhedron [8, 16] augmented by a degree constraint for each vertex in V.

The main idea in the proof of Theorem 1 is to use Lagrangean relaxation (see [55]) and to dualize the slightly weakened degree constraints (from B to $(1 + \omega)B$). For each constraint corresponding to vertex v we introduce a Lagrangean multiplier λ_v. We then remove all degree constraints and add corresponding penalty terms to the objective function.

Subsequently, we show that we can compute the optimum set of Lagrangean multipliers $\{\lambda^*_v\}_{v \in V}$ in polynomial time. We define a cost function

$$c^{\lambda^*}_{uv} = c_{uv} + \lambda^*_u + \lambda^*_v$$

and show that there is a minimum-c^{λ^*}-cost spanning tree of G with maximum degree $O(B + \log n)$. The proof is constructive and uses a local search algorithm proposed in [22]. Finally, we show that the c-cost of the computed tree is close to the minimum c-cost of any degree-B-bounded spanning tree using duality and the weakening in the degree constraints carefully.

1.4.2 Part 2: A new hybrid primal-dual, local search algorithm for the BMST problem

The prior algorithm from [44] has two main drawbacks: First, it relies on solving a large linear programming to compute an optimum set of Lagrangean multipliers. Secondly, the approach and its analysis inherently rely on the fact that we can compute an exact solution to a minimum-cost spanning tree problem. Since the problem of computing minimum-cost Steiner trees is NP-hard, we have little hope that the algorithm from [44] extends to this more general setting.

Therefore in Chapter 4 we develop a new *direct* algorithm for the degree-bounded minimum-cost spanning tree problem in order to address the above concerns. This algorithm does not use linear programming. Instead it uses a repeated application of Kruskal's [47] minimum-cost spanning tree algorithm interleaved with a combinatorial update of approximate Lagrangean node-multipliers maintained by the algorithm. These updates cause subsequent repetitions of the primal-dual algorithm to run for longer and longer times, leading to overall progress and a proof of the performance guarantee. A second useful feature of our algorithm is that, for the first time, it can handle non-uniform degree bounds on the nodes. The main theorem is as follows:

Theorem 2 *(see [45]) There is a primal-dual approximation algorithm that, given a graph $G = (V, E)$, a nonnegative cost function $c : E \rightarrow \mathbb{R}^+$, integer bounds $B_v > 1$ for all $v \in V$ and parameters $\omega > 1$ and $b > 1$ computes a tree T such that*

1. $\deg_T(v) \leq \max\left\{\frac{\omega}{\omega-1}, \omega\right\} \cdot bB_v + 2 \cdot \log_b n$ *for all* $v \in V$, *and*
2. $c(T) < \omega \cdot \texttt{opt}$.

The running time is $O(n^6 \log n)$.

We believe that the analysis of the primal-dual algorithm in Theorem 2 is of independent interest. The update steps of the Lagrangean node-multipliers can be viewed as local search steps. The algorithm is hence a hybrid of local-search and primal-dual updates. We believe that our techniques are applicable to other constrained network design problems where we take a network design problem for which primal-dual approximation algorithms are known and augment it by additional constraints.

1.4.3 Part 3: Minimum-cost degree-bounded Steiner trees

Finally, in Chapter 5, we show how the previous algorithm extends to Steiner trees using the primal-dual algorithm for approximate minimum-cost Steiner trees introduced by Agrawal, Klein, and Ravi [3] in place of Kruskal's minimum-cost spanning tree algorithm. Our main result there is as follows:

Theorem 3 *There is a primal-dual approximation algorithm that, given a graph $G = (V, E)$, a set of terminal nodes $R \subseteq V$, a nonnegative cost function $c : E \rightarrow \mathbb{R}^+$, integers $B_v > 1$ for all $v \in V$, and an arbitrary $b > 1$ computes a Steiner tree T that spans the nodes of R such that*

1. $\deg_T(v) \leq 12b \cdot B_v + \lceil 4 \log_b n \rceil + 1$ *for all* $v \in V$, *and*
2. $c(T) \leq 3\,\texttt{opt}$

where $\texttt{opt}$ *is the minimum cost of any Steiner tree whose degree at node v is bounded by B_v for all v. Our method runs in $O(n \log(|R|) \cdot |R|^{\lceil 4 \log n \rceil})$ iterations each of which can be implemented in polynomial time.*

The algorithm combines ideas from the primal-dual algorithm for Steiner trees due to Agrawal et al. [3] with local search elements from [22].

1.5 A ROAD-MAP OF THIS THESIS

The following Chapter 2 introduces the important ideas of *graph toughness* and *witness sets*, that were introduced by Chvátal [11] and Fürer and Raghavachari [22], respectively. Based on the idea of witness sets we review a local-search algorithm for minimum-degree spanning trees due to Fürer and Raghavachari [22] and its extention to minimum-degree minimum-cost spanning trees due to Fischer [20]. We also present a useful extention of the results in [20, 22] to fractional trees due to Éva Tardos [62].

Chapter 3 has the proof of Theorem 1. Chapter 4 presents our primal-dual algorithm for minimum-cost degree-bounded trees along with a proof of Theorem 2. Finally, Chapter 5 has our results for Steiner trees and a proof of Theorem 3.

Witness Sets, Toughness and Minimum-Degree Spanning Trees

The work in this thesis is based on fundamental work on lower-bounds for the minimum maximum degree of any node in any spanning tree of a given undirected graph.

Chvátal [11] defined the *toughness* of a given graph $G = (V, E)$. For a vertex set $S \subseteq V$ let $c(S)$ be the number of connected components of $G[V \setminus S]$.

Definition 1 *For a given undirected graph $G = (V, E)$, the minimum ratio $|S|/c(S)$ over all subsets S of V with $c(S) > 1$ is called the* toughness *of G. We denote this quantity by $\tau(G)$.*

The toughness of a given graph is closely connected to the minimum maximum degree of any spanning tree of a given graph. Let $\Delta^*(G)$ denote the minimum possible maximum degree of any spanning tree of an undirected graph G. It can now be seen without much effort that the minimum maximum degree of any spanning tree of G must be at least $1/\tau(G)$.

Win [69] showed the following relationship between $\tau(G)$ and $\Delta^*(G)$:

Theorem 4 *An undirected graph $G = (V, E)$ has a tree of maximum degree at most Δ if for any $S \subseteq V$,*

$$c(S) \leq (\Delta - 2) \cdot |S| + 2.$$

This immediately implies the following corollary:

Corollary 1 *An undirected graph $G = (V, E)$ has a spanning tree of degree Δ if $\tau(G) \geq 1/(\Delta - 2)$.*

The proof of Theorem 4 is by contradiction and assumes that we are given a graph G with $c(S) \leq (\Delta - 2) \cdot |S| + 2$ for all $S \subseteq V$ and that $\Delta * (G) > \Delta$. The main part of the proof then establishes the existence of a *witness set* $W \subseteq V$ such that $G[V \setminus W]$ has at least $(k-2)|W|+3$ connected components.

The interesting fact about the proof is that it constructs a vertex set W that serves as a certificate for the non-existence of a spanning tree of maximum degree Δ. We call say that W *witnesses* the fact that there is no tree of maximum degree Δ and we call W a *witness set.*

We now review a local-search method that demonstrates how witness sets can be used to compute approximate minimum-degree spanning trees.

2.1 A LOCAL IMPROVEMENT ALGORITHM

In this section we review a local-search procedure for the minimum-degree spanning tree problem due to Fürer and Raghavachari [22]. The procedure starts with an arbitrary spanning tree T and improves

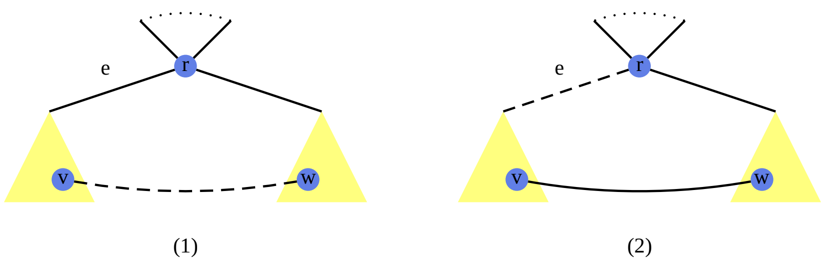

Figure 2.1: The figure shows a fragment of a spanning tree of a given undirected graph. Solid edges are part of the tree and dashed edges are not in the tree. The figure shows a swap step that swaps edge e that is incident to vertex against a non-tree edge uw. Figure (1) and (2) show the tree before and after the swap, respectively.

it by applying so called *swaps*. The degree of a node r is improved by replacing an edge $e\delta_T(r)$ with an edge $vw \notin E[T]$ such that r is on the unique cycle in $T + vw$. Figure 2.1 shows an example for a swap.

We are interested in swaps that lower the maximum degree of the triple of vertices that are involved in the swap.

Definition 2 *Given a tree T and a non-tree edge $uv \notin E[T]$, let $\mathcal{C}(uv)$ be the unique cycle in $T + \{uv\}$ and let $wz \in \mathcal{C}(e)$. We call the swap $\langle uv, wz\rangle$ an* improvement for w *if*

$$\max\{\deg_T(u), \deg_T(v)\} + 1 < \deg_T(w).$$

If an edge swap $\langle uv, wz\rangle$ is an improvement step for either w or z then the maximum degree of the nodes u, v, w and z decreases as a result of the swap; We call such a swap simply an improvement.

The algorithm in [22] performs improvement steps as long as possible. In fact, it is not hard to see that starting with an arbitrary tree, the number of possible improvements is finite. We end up with a *locally optimal tree.*

Definition 3 *A tree T is called* locally optimal *(LOT) if no vertex degree can be decreased by applying an improvement swap.*

Computing a locally optimal tree might be too ambitious a goal however. In fact, it is not known how to do this in polynomial time. However, the analysis in [22] shows that it is enough to compute a *pseudo-optimal* tree.

Definition 4 *A tree T of maximum degree $\Delta(T)$ is called* pseudo-optimal *(POT) if for all vertices v with $\Delta(T) - \lceil \log_b n \rceil \le \deg_T(v) \le \Delta(T)$, no improvement step for v is applicable. Here b is an arbitrary constant bigger than one.*

Fischer [20] considered weighted graphs where each edge $e \in E$ has a non-negative cost c_e associated with it. His adaptation of the algorithm from [22] computes a minimum-cost spanning tree of approximately minimum maximum degree. To obtain his algorithm we have to make two small changes to the procedure from [22]. First, instead of starting with an arbitrary spanning tree, we start with a minimum-cost spanning tree. Second, an improvement step must be *cost neutral.* That is, for an improvement step $\langle uv, wz\rangle$ to be applicable we must have $c_{uv} = c_{wz}$. Algorithm 1 states the procedure.

Algorithm 1 The algorithm `PLocal` computes a pseudo-optimal tree.

```
1: Given: graph $G = (V, E)$ and cost function $c : E \to \mathbb{R}^+$
2: $T \leftarrow$ MST$(G, c)$
3: while $T$ is not pseudo optimal do
4:    Identify cost neutral improvement $\langle uv, wz\rangle$
5:    $T \leftarrow T - wz + uv$
6: end while
```

2.1.1 Analysis and running time

In what follows we highlight and strengthen the major ideas of the analysis from [20, 22]. The strengthening is due to Éva Tardos [62] and will turn out to be useful in a later part of this thesis.

The fundamental underlying proof idea for the unweighted problem [22] is based on an averaging argument that we present here. Let a set $W \subseteq V$ be such that for a given graph $G = (V, E)$, the graph $G[V - W]$ has t connected components. A spanning tree of G has to connect these t components *and* the nodes from W. We need exactly $t + |W| - 1$ edges going between the nodes of W and the t connected components to accomplish this. Each of these edges must be incident to a node from W. Hence averaging yields a lower bound of $(t + |W| - 1)/|W|$ on the maximum degree Δ^* of T.

Proposition 1 (see [22]) *Let W be a set of size w whose removal splits G into t components. Then $\Delta^* \geq \lceil \frac{w+t-1}{w} \rceil$.*

We now turn to the weighted case, i.e. the *minimum-degree minimum-cost spanning tree* problem. The above mentioned strengthening of the results from [20] is based on the following definitions.

Definition 5 *Given an undirected graph $G = (V, E)$ and a non-negative cost function c on the edges, let $\mathcal{O}^c$ be defined as*

$$\mathcal{O}^c = \{T : T \text{ is an MST under cost } c\}.$$

In the following we will be talking about convex combinations of spanning trees. Hence we introduce some further simplifying notation.

Definition 6 *Let $T_c^\alpha = \sum_{T \in \mathcal{O}^c} \alpha_T T$ be a convex combination of minimum-cost spanning trees of G with respect to cost function c, i.e. $\alpha_T \geq 0$ for all T and $\sum_{T \in \mathcal{O}^c} \alpha_T = 1$. We denote the* fractional degree *of vertex v in T_c^α by*

$$\deg_c^\alpha(v) = \sum_{T \in \mathcal{O}^c} \alpha_T \deg_T(v).$$

Finally we define the minimum maximum degree of convex combinations of spanning trees.

Definition 7 *Given $G = (V, E)$ and a non-negative cost function c on the edges, let Δ_c^* denote the minimum maximum degree of any convex combination of minimum-cost spanning trees, i.e.*

$$\Delta_c^* = \min_{\text{convex comb. } \alpha} \max_{v \in V} \deg_c^\alpha(v).$$

The following easy proposition will be used in the later analysis. We provide its proof that originally appeared in [22] for completeness.

Lemma 1 *For any constant $b > 1$ and a tree T, let S_d be the set of nodes that have degree at least d in T. Then, there is a*

$$d \in \{\Delta(T) - \lceil \log_b n \rceil + 1, \ldots, \Delta(T)\}$$

such that $|S_{d-1}| \leq b|S_d|$.

Proof: Suppose for a contradiction that for all $d \in \{\Delta(T) - \lceil \log_b n \rceil + 1, \ldots, \Delta(T)\}$, we have

$$|S_{d-1}| > b \cdot |S_d|.$$

Repeated application of this inequality now yields

$$\begin{array}{rcl} |S_{\Delta(T)-\lceil \log_b n \rceil}| & > & b \cdot |S_{\Delta(T)-\lceil \log_b n \rceil + 1}| \\ & > & b^2 \cdot |S_{\Delta(T)-\lceil \log_b n \rceil + 2}| \\ & > & \ldots \\ & > & b^{\lceil \log_b n \rceil} \cdot |S_{\Delta(T)}| \\ & \geq & n \cdot |S_{\Delta(T)}|. \end{array}$$

Note that since $S_{\Delta(T)}$ has at least one vertex and our graph has n vertices this is a contradiction. ■

The main theorem is the following.

Theorem 5 (see [20, 22]) *If T is a pseudo-optimal MST, then $\Delta(T) < b\Delta_c^* + \lceil \log_b n \rceil$ for any constant $b > 1$. Moreover, a pseudo-optimal MST can be computed in polynomial time.*

Proof: Given a constant $b > 1$, choose d as in Proposition 1. That is, we have $|S_{d-1}| \leq b|S_d|$. Recall that S_d contains the nodes of degree at least d in the tree T.

Removing S_d from T leaves us with a forest F. Let $\widehat{G}$ be obtained from G by contracting each connected component of F. It is now easy to see that every minimum-cost spanning tree of G contains a minimum-cost spanning tree of $\widehat{G}$ (e.g., every edge added by Kruskal's algorithm for finding a minimum-cost spanning tree for G is feasible for a minimum-cost spanning tree of $\widehat{G}$ if it were not contracted in the formation of $\widehat{G}$).

Let $(u, v) \in E - T$ be an edge that connects two components of F such that $u, v \notin S_{d-1}$, i.e. both u and v have degree at most $d - 2$. We claim that such an edge cannot be included in any minimum spanning tree of $\widehat{G}$. To see that, let $P^T_{u,v}$ be the edges of the unique u,v-path in T and let $\widehat{P^T_{u,v}}$ be the subset of the edges of $P^T_{u,v}$ that are in $\widehat{G}$.

It follows from the pseudo-optimality of T that the cost of edge (u, v) must be higher than the cost of each edge from $\widehat{P^T_{u,v}}$. For otherwise, (u, v) can be swapped in place of another edge of the same or higher cost in $\widehat{P^T_{u,v}}$ and all such edges are incident to at least one node in S_{d-1}, leading to an improvement. This means (u, v) cannot be a part of any minimum spanning tree of $\widehat{G}$. Equivalently, a minimum-cost spanning tree of G must use edges incident to S_{d-1} to connect the components of F and the nodes of S_d.

By the definition of S_d, we know that F has at least

$$|S_d|d - 2(|S_d| - 1) = |S_d|(d - 2) + 2$$

trees. This follows from an easy counting argument after observing that every node in S_d has degree at least d in T and there are at most $|S_d| - 1$ edges going between nodes of S_d.

This means that we need at least

$$|S_d|(d - 2) + 2 + |S_d| - 1 = |S_d|(d - 1) + 1$$

edges to connect up the components of F and the nodes of S_d in *any spanning tree.* By the preceeding argument each of these edges has to be incident to at least one node of degree at least $d - 1$ in an MST. Hence the the average degree of a node of S_{d-1} in any MST is

$$\frac{|S_d|(d-1)+1}{|S_{d-1}|} > \frac{d-1}{b}.$$

Moreover, the average degree of a node in S_{d-1} in any *convex combination* of MST's is also at least the above ratio. Since Δ_c^* denotes the minimum possible maximum degree of any fractional MST, it follows using our choice of index d from Proposition 1 that

$$\Delta_c^* > \frac{d-1}{b}.$$

Using the range of d we obtain $\Delta(T) < b\Delta_c^* + \lceil \log_b n \rceil$. The results in [20, 22] show a lower-bound on the degree of any MST. The extention to fractional MST's is the mentioned strengthening [62] of the previous ideas.

For the running time we follow [22]. Note that each improvement step can be implemented in polynomial time. We need to bound the number of iterations. The proof uses a potential function argument. Define the potential of a vertex v as

$$\Phi(v) = 3^{\deg_T(v)}$$

where T is the current tree. The total potential is the sum over all vertex potentials, that is

$$\Phi(T) = \sum_{v \in V} \Phi(v).$$

Now, an improvement step in Algorithm 1 improves the degree of a vertex $v \in S_d$ with $\deg_T(v) = d$ and $d \geq \Delta(T) - \lceil \log_b n \rceil + 1$. Hence, the reduction in the potential is going to be at least

$$(3^d + 2 \cdot 3^{d-2}) - 3 \cdot 3^{d-1} = 2 \cdot 3^{d-2}.$$

Using the range of d we can lower bound the right hand side of the last equality by

$$3^{\Delta(T) - \log_b n - 1} = \Omega\left(\frac{3^{\Delta(T)}}{n}\right).$$

The potential $\Phi(T)$ of the tree T is at most $n3^{\Delta(T)}$. This implies that the overall decrease of the potential due to the improvement step is

$$\Omega\left(\frac{\Phi(T)}{n^2}\right)$$

In other words, we reduce the potential by at least a polynomial factor in each iteration. In $O(n^2)$ iterations the reduction is by a constant factor. Hence, the algorithm needs $O(n^3)$ improvement steps in total. ■

Minimum-Cost Degree-Bounded Spanning Trees With Uniform Degree Bounds

In this chapter we present the proof of Theorem 1. The degree-B-bounded minimum-cost spanning tree problem can be modeled by an integer linear program in a straight forward way.

$$\begin{aligned} \texttt{opt}_B = \min \quad & \sum_{e\in E} c_e x_e && \text{(IP}_1\text{)} \\ \text{s.t} \quad & x(\delta(v)) \le B \quad \forall v \in V && (3.1) \\ & x \in \mathrm{SP}_G \\ & x \text{ integer} \end{aligned}$$

Here, $\delta(v)$ denotes the set of all edges of E that are incident to v and SP_G is the spanning tree polyhedron, that is, the convex hull of edge-incidence vectors of spanning trees of G. We note that complete descriptions of SP_G are known in the literature (See [8, 16] and also Section 4.1).

3.1 TECHNIQUE: LAGRANGEAN DUALITY

Our algorithm builds on the Lagrangean dual of (IP$_1$) resulting from *dualizing* the degree constraints. We denote its value by $\texttt{opt}_{LD(B)}$.

$$\texttt{opt}_{LD(B)} = \max_{\lambda\ge 0} \min_{T\in \mathrm{SP}_G} \{c(T) + \sum_{v\in V} \lambda_v(\deg_T(v) - B)\}. \qquad \text{(LD(B))}$$

For any fixed $\lambda \ge 0$, an optimum integral solution to (IP$_1$) is a feasible candidate for attaining the inner minimum above. Since the maximum degree of such a solution is at most B and $\lambda \ge 0$, it follows that $\texttt{opt}_{LD(B)}$ is a lower bound on $\texttt{opt}_B$.

Proposition 2 (see [55]) $\texttt{opt}_{LD(B)} \le \texttt{opt}_B$

The interesting fact is that $\texttt{opt}_{LD(B)}$ can be computed in polynomial time since the vector λ^B of optimum Lagrangean multipliers on the nodes can be obtained by computing an optimum solution to the dual of the linear programming relaxation of (IP$_1$) (see the proof of Theorem 6). We also obtain a set of optimum trees $\mathcal{O}^B$, all of which achieve the inner minimum for λ^B. In other words, every tree $T^B \in \mathcal{O}^B$ minimizes the following objective:

$$c(T^B) + \sum_{v\in V} \lambda_v^B(\deg_{T^B}(v) - B).$$

Given λ^B, the task of finding a tree T that minimizes the above objective function is called the Lagrangean subproblem of LD(B).

Using $c^{\lambda^B}(uv) = c(uv) + \lambda_u^B + \lambda_v^B$ the last expression can be restated as

$$c^{\lambda^B}(T^B) - B \sum_{v \in V} \lambda_v^B \tag{3.2}$$

Notice that for a given λ^B and B, the second term is a constant. Hence, any minimum spanning tree of G under cost c^{λ^B}, denoted by $\texttt{MST}(G, c^{\lambda^B})$, is a solution for T.

An important feature of our algorithm is to relax the degree constraints slightly from B to $(1+\omega)B$ for some fixed $\omega > 0$ and show that there is a tree $T \in \mathcal{O}^{(1+\omega)B}$ that satisfies the conditions of Theorem 1.

We now present our first algorithm for the minimum-cost degree-bounded spanning tree problem. It uses the local-search procedure for minimum-degree spanning trees from Chapter 2 crucially.

3.2 THE BMST-ALGORITHM

In this section, we describe our algorithm for the BMST problem. It uses the Lagrangean formulation LD(B) from the introduction and Algorithm 1. We use `PLocal(T)` to denote an application of the local-search procedure from Chapter 2 to the tree T.

The input to our algorithm consists of a graph G, a non-negative cost function c, a degree bound B and a non-negative parameter ω. Let $B^* = (1 + \omega)B$.

Algorithm 2 Our algorithm for the BMST problem

```
1: Given: graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}^+$, a degree bound $B \geq 2$ and a parameter
   $\omega > 0$.
2: $B^* \leftarrow (1 + \omega)B$
3: $\lambda^* \leftarrow \texttt{Solve}(\text{LD}(B^*))$
4: $\overline{T} \leftarrow \texttt{PLocal}(G, c^{\lambda^*})$
```

Since the optimum Lagrange multipliers and pseudo-optimal MST's can be computed in polynomial time [22, 55], Algorithm 2 runs in polynomial time.

Recall that c^{λ^*} denotes the original cost function c *augmented* by the Lagrangean multipliers λ^*, i.e. $c_{uv}^{\lambda^*} = c_{uv} + \lambda_u^* + \lambda_v^*$. We use $\mathcal{O}^*$ to denote the set of all minimum-cost spanning trees of G for cost function c^{λ^*}.

3.3 ANALYSIS

In this section we prove Theorem 1. First we show that the cost $c(\overline{T})$ of the tree output by Algorithm 2, $\overline{T}$, is small. Then, we prove that $\overline{T}$ has low maximum degree. Our proofs rely on techniques from Lagrangean duality.

3.3.1 The cost of $\overline{T}$

Recall that $\texttt{opt}_{LD(B)} \leq \texttt{opt}_B$ from Proposition 2. Unfortunately, $\texttt{opt}_{LD(B)} = \texttt{opt}_B$ is not true in general. There might be a *duality gap*. However, it can be shown that $\texttt{opt}_{LD(B)}$ equals the optimum objective function value of the relaxation of (IP$_1$). The proof is insightful and hence we present it here.

Theorem 6 *(see [55])* $\texttt{opt}_{LD(B)} = \min\{c(T) : T \in SP_G, \forall v \in V : \deg_T(v) \leq B\}$

Proof: We can restate (LD(B)) as the following linear program in variables η and λ. Recall that we denote its maximum objective function value by $\texttt{opt}_{LD(B)}$.

$$\begin{aligned} \max \quad & \eta \\ \text{s.t.} \quad & \eta \le c(T) - \sum_{v \in V} \lambda_v (B - \deg_T(v)) \quad \forall T \in \mathrm{SP}_G \\ & \lambda \ge 0 \end{aligned} \tag{3.3}$$

The first block of constraints states that η is at most the cost of any spanning tree T of G with respect to the Lagrangean function (3.2). The maximization objective of (3.3) forces η to attain the best possible cost. Writing down the dual of the last program yields

$$\begin{aligned} \min \quad & c\Big(\sum_{T \in \mathrm{SP}_G} \alpha_T T\Big) \\ \text{s.t.} \quad & \sum_{T \in \mathrm{SP}_G} \alpha_T = 1 \\ & \sum_{T \in \mathrm{SP}_G} \alpha_T \deg_T(v) \le B \sum_{T \in \mathrm{SP}_G} \alpha_T = B \quad \forall v \in V \\ & \alpha \ge 0 \end{aligned} \tag{3.4}$$

Note that $T^\alpha = \sum_{T \in \mathrm{SP}_G} \alpha_T T$ is a convex combination of trees in SP_G. Also, observe that

$$\sum_{T \in \mathrm{SP}_G} \alpha_T \deg_T(v)$$

is precisely the degree $\deg^\alpha(v)$ of this fractional tree at node v. These observations yield the theorem. ■

The theorem has two nice corollaries that we use. In the following, let λ^* denote the vector of optimum Lagrangean multipliers for (LD(B^*)). Recall that $\mathcal{O}^*$ is the set of minimum-cost spanning trees under c^{λ^*}.

Corollary 2 *There exists a convex combination* $T^\alpha = \sum_{T \in \mathcal{O}^*} \alpha_T T$ *such that*

1. $\forall v \in V : \deg^\alpha_{c^{\lambda^*}}(v) \le B^*$ *and*
2. $\lambda^*_v > 0$ *only if* $\deg^\alpha_{c^{\lambda^*}}(v) = B^*$.

Proof: This follows from complementary slackness applied to the optimum solutions of the dual linear programs (3.3) and (3.4) with B and λ replaced by B^* and λ^*. ■

The second corollary gives a bound on $\Delta^*_{c^{\lambda^*}}$.

Corollary 3 $\Delta^*_{c^{\lambda^*}} \le B^*$

Proof: By Corollary 2, we know that there is a convex combination T^α of trees from $\mathcal{O}^*$ such that $\deg^\alpha_{c^{\lambda^*}}(v) \le B^*$ for all v. Hence

$$\Delta^*_{c^{\lambda^*}} = \min_\alpha \max_{v \in V} \deg^\alpha_{c^{\lambda^*}}(v) \le B^*.$$

■

We now prove that $c(\overline{T})$ is small.

Lemma 2 *For all trees $T \in \mathcal{O}^*$ we have $c(T) < (1+1/\omega)\,\texttt{opt}_B$.*

Proof: Recall that we defined $B^* = (1+\omega)B$

The following inequality holds for every $T \in \mathcal{O}^*$:

$$\begin{aligned}\sum_{v\in V} \lambda_v^*(\deg_T(v) - B) &\leq c(T) + \sum_{v\in V} \lambda_v^*(\deg_T(v) - B) \\ &\leq \texttt{opt}_{LD(B)}\end{aligned} \tag{3.5}$$

In the first inequality we just added $c(T)$. Note, that the right hand side of the first line is just the Lagrangean objective function (3.2) for B. Recall that T is a minimum spanning tree with respect to c^{λ^*}. Moreover, λ^* is a feasible set of multipliers for (LD(B)). Hence, the second inequality follows.

We also have

$$\begin{aligned}c(T) &= c(T) + \sum_{v\in V} \lambda_v^*(\deg_T(v) - B) + \sum_{v\in V} \lambda_v^*(B - \deg_T(v)) \\ &\leq \texttt{opt}_{LD(B)} + \sum_{v\in V} \lambda_v^*(B - \deg_T(v))\end{aligned}$$

where the inequality follows from (3.5). Applying Proposition 2 and the fact that $\deg_T(v) \geq 1$ for all $v \in V$ leads to

$$c(T) < \texttt{opt}_B + B \sum_{v\in V} \lambda_v^*.$$

In the remainder of this proof we will derive the inequality $B\sum_{v\in V}\lambda_v^* \leq \texttt{opt}_B/\omega$. This yields the lemma. From Corollary 2, we know that there is a convex combination

$$T^\alpha = \sum_{T\in\mathcal{O}^*} \alpha_T T$$

such that $\lambda_v^* > 0$ only if $\deg^{\alpha}_{c^{\lambda^*}}(v) = B^*$.

We derive a new inequality by summing over all $T \in \mathcal{O}^*$, α_T times the inequality (3.5) for each T. We obtain

$$\sum_{T\in\mathcal{O}^*} \alpha_T \left(\sum_{v\in V} \lambda_v^*(\deg_T(v) - B) \right) \leq \texttt{opt}_{LD(B)} \sum_{T\in\mathcal{O}^*} \alpha_T \tag{3.6}$$

The right hand side is equivalent to $\texttt{opt}_{LD(B)}$ because $\sum_{T\in\mathcal{O}^*}\alpha_T = 1$. Reordering the left hand side yields

$$\sum_{v\in V} \lambda_v^* \left(\left(\sum_{T\in\mathcal{O}^*} \alpha_T \deg_T(v) \right) - B \right)$$

Instead of summing over all $v \in V$ it suffices to sum over v, where $\lambda_v^* > 0$. For such v, we have

$$\deg^{\alpha}_{c^{\lambda^*}} = \sum_{T\in\mathcal{O}^*} \alpha_T \deg_T(v) = B^*$$

by Corollary 2. Using $B^* = (1+\omega)B$ it follows that the left hand side of (3.6) is equivalent to

$$\omega B \sum_{v\in V} \lambda_v^*$$

and this finishes the proof of the lemma. ■

3.3.2 The Maximum Degree of $\overline{T}$

Lemma 3 $\Delta(\overline{T}) \leq (1+\omega)bB + \lceil \log_b n \rceil$ *for constants* $b > 1$ *and* ω*.*

Proof: $\overline{T}$ is a pseudo-optimal minimum-cost spanning tree with respect to cost function c^{λ^*}. From Theorem 5 we know that

$$\Delta(\overline{T}) \leq b\Delta^*_{c^{\lambda^*}} + \lceil \log_b n \rceil.$$

An application of Corollary 3, noting $B^* = (1+\omega)B$ yields the lemma. ■

Minimum-Cost Degree-Bounded Spanning Trees With Non-Uniform Degree Bounds

In this chapter, we present a proof of Theorem 2. Given an undirected graph $G = (V, E)$, a cost function $c : E \rightarrow \mathbb{R}^+$ and positive integers $\{B_v\}_{v \in V}$ all greater than 1, the goal is to find a spanning tree T of minimum total cost such that for all vertices $v \in V$ the degree of v in T is at most B_v.

In the light of the results presented in the previous Chapter 3 the results in this chapter are novel in two ways: First, we present improved approximation algorithms for the minimum-cost degree-bounded spanning tree problem in the presence of non-uniform degree bounds. Second, our algorithm is *direct* in the sense that we do not solve linear programs. The algorithm in Chapter 3 uses Lagrangean relaxation and thus needs to solve a linear program. Moreover, the analysis relies crucially on the fact that we compute an exact solution to this LP.

Our new algorithm integrates elements from the primal-dual method for approximation algorithms for network design problems with local search methods for minimum-degree network problems [22]. The algorithm goes through a series of spanning trees and improves the maximum deviation of any vertex degree from its respective degree bound continuously. A practical consequence of this is that we can terminate the algorithm at any point in time and still obtain a spanning tree of the input graph (whose node-degrees, of course, may not meet the worst-case guarantees we prove).

This chapter is organized as follows: First, we review the primal-dual interpretation of the well-known algorithm for minimum-cost spanning trees by Kruskal [47]. Subsequently, we show how to use this algorithm for the nBMST problem and present an analysis of performance guarantee and running time of our method.

4.1 A PRIMAL-DUAL ALGORITHM TO COMPUTE MST'S

In this section we review Kruskal's minimum-cost spanning tree algorithm. More specifically, we discuss a primal-dual interpretation of this method that follows from [8]. We start by giving a linear programming formulation of the convex hull of incidence vectors of spanning trees.

4.1.1 The spanning tree polyhedron

In the following, we formulate the minimum-cost spanning tree problem as an integer program where we associate a 0, 1-variable x_e with every edge $e \in E$. In a solution x, the value of x_e is one if e is included in the spanning tree corresponding to x and 0 otherwise. Our formulation relies on a complete formulation of the convex hull of incidence vectors of spanning trees (denoted by SP_G) given by Chopra [8].

Chopra's formulation uses the notion of a *feasible partition* of vertex set V. A feasible partition of V is a set $\pi = \{V_1, \ldots, V_k\}$ such that the V_i are pairwise disjoint subsets of V. Moreover, $V = V_1 \cup \ldots \cup V_k$

and the induced subgraphs $G[V_i]$ are connected. Let G_π denote the (multi-) graph that has one vertex for each V_i and edge (V_i, V_j) occurs with multiplicity $|\{(v_i, v_j) : v_i \in V_i, v_j \in V_j\}|$. In other words, G_π results from G by contracting each of the V_i to a single node. Define the *rank* $r(\pi)$ of π as the number of nodes of G_π and let Π be the set of all feasible partitions of V. Chopra showed that

$$\mathrm{SP}_G = \{x \in \mathbb{R}^m : \sum_{e \in E(G_\pi)} x_e \geq r(\pi) - 1 \quad \forall \pi \in \Pi\}.$$

We now let $\delta(v)$ denote the set of edges $e \in E$ that are incident to node v and, for a subset $S \subseteq V$, we define $\delta(S)$ to be the set of edges that have exactly one endpoint in S. We obtain an integer programming formulation for our problem:

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e & \text{(IP)} \\ \text{s.t} \quad & \sum_{e \in E[G_\pi]} x_e \geq r(\pi) - 1 \quad \forall \pi \in \Pi & \\ & x(\delta(v)) \leq B_v \quad \forall v \in V & (4.1) \\ & x \text{ integer} & \end{aligned}$$

The dual of the linear programming relaxation (LP) of (IP) is given by

$$\begin{aligned} \max \quad & \sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_\pi - \sum_{v \in V} \lambda_v B_v & \text{(D)} \\ \text{s.t} \quad & \sum_{\pi : e \in E[G_\pi]} y_\pi \leq c_e + \lambda_u + \lambda_v \quad \forall e = uv \in E & (4.2) \\ & y, \lambda \geq 0 & \end{aligned}$$

We also let (IP-SP) denote (IP) without constraints of type (4.1). Let the LP relaxation be denoted by (LP-SP) and let its dual be (D-SP).

4.1.2 A primal-dual interpretation of Kruskal's MST algorithm

Kruskal's algorithm can be viewed as a continuous process over *time*: We start with an empty tree at time 0 and add edges as we go along. The algorithm terminates at time t^* with a spanning tree of the input graph G. In this section we show that Kruskal's method can be interpreted as a primal-dual algorithm (see also [30]). At any time $0 \leq t \leq t^*$ we keep a pair (x_t, y_t), where x_t is a partial (possibly infeasible) primal solution for (LP-SP) and y_t is a feasible dual solution for (D-SP). Initially, we let $x_{e,0} = 0$ for all $e \in E$ and $y_{\pi,0} = 0$ for all $\pi \in \Pi$.

Let E_t be the forest corresponding to partial solution x_t, i.e. $E_t = \{e \in E : x_{e,t} = 1\}$. We then denote by π_t the partition induced by the connected components of $G[E_t]$. At time t, the algorithm then increases y_{π_t} until a constraint of type (4.2) for edge $e \in E \setminus E_t$ becomes tight. Assume that this happens at time $t' > t$. The dual update is

$$y_{\pi_t, t'} = t' - t.$$

We then include e into our solution, i.e. we set $x_{e,t'} = 1$. If more than one edge becomes tight at time t', we can process these events in any arbitrary order; Thus, note that we can pick any such tight edge first in our solution. Subsequently, we will use `MST` to refer to the above algorithm.

The proof of the following lemma is folklore. We supply it for the sake of completeness.

Lemma 4 *At time t^*, Algorithm* MST *finishes with a pair (x_{t^*}, y_{t^*}) of primal and dual feasible solutions to (IP-SP) and (D-SP), respectively, such that*

$$\sum_{e \in E} c_e x_{e,t^*} = \sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_{\pi,t^*}$$

Proof: Notice, that for all edges $e \in E_{t^*}$ we must have $c_e = \sum_{\pi : e \in E[G_\pi]} y_{\pi,t^*}$ and hence, we can express the cost of the final tree as follows:

$$c(E_{t^*}) = \sum_{e \in E_{t^*}} \sum_{\pi : e \in E[G_\pi]} y_{\pi,t^*} = \sum_{\pi \in \Pi} |E_{t^*} \cap E[G_\pi]| \cdot y_{\pi,t^*}.$$

By construction E_{t^*} is a tree and we must have that the set $E_{t^*} \cap E[G_\pi]$ has cardinality exactly $r(\pi) - 1$ for all $\pi \in \Pi$ with $y_{\pi,t^*} > 0$. We obtain that $\sum_{e \in E} c_e x_{e,t^*} = \sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_{\pi,t^*}$ and this finishes the proof of the lemma. ■

4.2 MINIMUM-COST DEGREE-BOUNDED SPANNING TREES

In this section, we propose a modification of the above algorithm for approximating degree-bounded spanning trees of low total cost (for suitably weakened degree bounds). Our algorithm goes through a sequence of spanning trees $E^0, \ldots, E^t$ and associated pairs of primal (infeasible) and dual feasible solutions x^i, (y^i, λ^i) for $0 \leq i \leq t$. The idea is to reduce the degree of nodes $v \in V$ whose degree is substantially higher than their associated bound B_v, as we proceed through this sequence, while keeping the cost of the associated primal solution (tree) bounded with respect to the corresponding dual solution.

To begin, our algorithm first computes an approximate minimum-cost spanning tree using the Algorithm MST. This yields a feasible primal solution x^0 for (LP-SP) and a feasible dual solution y^0 for (D-SP). Notice that y^0 also induces a feasible solution for (D) by letting $\lambda^0_v = 0$ for all $v \in V$ while x^0 potentially violates constraints of type (4.1).

We introduce the notion of *normalized degree* of a node V in a tree T and denote it by

$$\texttt{ndeg}_T(v) = \max\{0, \deg_T(v) - \beta_v \cdot B_v\} \tag{4.3}$$

where $\beta_v > 0$ are constants for all $v \in V$ to be specified later. Our algorithm successively computes pairs of spanning trees and associated dual solutions to (D), i.e.

$$(x^1, \{y^1, \lambda^1\}), (x^2, \{y^2, \lambda^2\}), \ldots, (x^t, \{y^t, \lambda^t\}).$$

From one such pair to the next, we try to reduce the degree of nodes of high normalized degree. Specifically, our algorithm runs as long as there is a node in the current tree with $\texttt{ndeg}(v) \geq 2\log_b(n)$ for some constant $b > 1$.

We let E^i be the spanning tree corresponding to x^i and let Δ^i be the maximum normalized degree of any node in the tree E^i. The central piece of our algorithm is a *recompute step* where we raise the λ values of a carefully chosen set S_d of nodes with high normalized degree. This introduces slack in many of the constraints of type (4.2). Subsequently, we rerun MST and ensure that it generates a feasible dual packing that takes advantage of the newly created slack around nodes of high normalized degree. At the same time MST computes a new tree and we ensure at all times that its cost is close to the objective function value of the associated dual. We are able to show that the number of recompute steps is polynomial, by arguing that we make substantial progress in the normalized degree sequence of all nodes.

As mentioned, each recompute step takes a pair of primal infeasible and dual feasible solutions $(x^i, (y^i, \lambda^i))$ and computes a new pair of primal (infeasible) and dual feasible solutions $(x^{i+1}, (y^{i+1}, \lambda^{i+1}))$.

In the following we use $\mathtt{ndeg}^i(v)$ as a short for $\mathtt{ndeg}_{E^i}(v)$. We then adapt the notation from [21, 22] and let

$$S_d^i = \{v \in V \,:\, \mathtt{ndeg}^i(v) \geq d\}$$

be the set of all nodes whose normalized degree is at least d in the i^{th} solution. Furthermore, let Δ^i be the maximum normalized degree of any node in E^i, i.e.

$$\Delta^i = \max_{v \in V} \mathtt{ndeg}^i(v).$$

Algorithm 3 The algorithm for the nBMST problem attempts to reduce the maximum normalized degree of any node in a given spanning tree.

```
1: $\lambda_v^0 \leftarrow 0, \forall v \in V; \widetilde{c}_e^0 \leftarrow c_e, \forall e \in E$
2: $(x^0, y^0) \leftarrow \mathtt{MST}(G, \widetilde{c}^0)$
3: $i \leftarrow 0$
4: while $\Delta^i > \lceil 2\log_b n \rceil$ do
5:    Choose $d^i \in \{\Delta^i - \lceil 2\log_b(n) \rceil + 1, \ldots, \Delta^i\}$ s.t. $\sum_{v \in S^i_{d^i-1}} B_v \leq b \cdot \sum_{v \in S^i_{d^i}} B_v$
6:    Choose $\epsilon^i > 0$
7:    For all $v \in V$ let
          $$\lambda_v^{i+1} \leftarrow \begin{cases} \lambda_v^i + \epsilon^i & : \; v \in S^i_{d^i-1} \\ \lambda_v^i & : \; \text{otherwise} \end{cases}$$
8:    $\widetilde{c}^{i+1}(e) \leftarrow \widetilde{c}^i(e) + \epsilon^i$ if either $e \in E^i$ and $e \cap S^i_{d^i} \neq \emptyset$ or $e \notin E^i$ and $e \cap S^i_{d^i-1} \neq \emptyset$
9:    $(x^{i+1}, y^{i+1}) \leftarrow \mathtt{MST}(G, \widetilde{c}^{i+1})$
10:   $i \leftarrow i + 1$
11: end while
```

A detailed description of the procedure is given in Algorithm 3. Recall that $\mathtt{MST}(G, \widetilde{c})$ returns a pair of primal and dual optimal solutions to (LP) and (D) for cost function $\widetilde{c}$. In step 5 of Algorithm 3, we choose a suitable set of nodes whose λ-values we increase. The arguments used to prove Lemma 1 can be extended to show the following.

Lemma 5 *There is a $d^i \in \{\Delta^i - \lceil 2\log_b n \rceil + 1, \ldots, \Delta^i\}$ such that*

$$\sum_{v \in S^i_{d^i-1}} B_v \leq b \cdot \sum_{v \in S^i_{d^i}} B_v$$

for a given constant $b > 1$.

Proof: Suppose for a contradiction that for all $d^i \in \{\Delta^i - \lceil 2\log_b n \rceil + 1, \ldots, \Delta^i\}$, we have

$$\sum_{v \in S^i_{d^i-1}} B_v > b \cdot \sum_{v \in S^i_{d^i}} B_v.$$

Note that since we may assume $B_v \leq (n-1)$ for all vertices, we must have $\sum_{v \in V} B_v \leq n(n-1)$. However, since $\sum_{v \in S^i_{\Delta^i}} B_v \geq 1$, we have in this case that

$$\sum_{v \in S^i_{\Delta^i - 2\log_b n}} B_v \geq b^{\lceil 2\log_b(n) \rceil} \geq n^2$$

a contradiction. ■

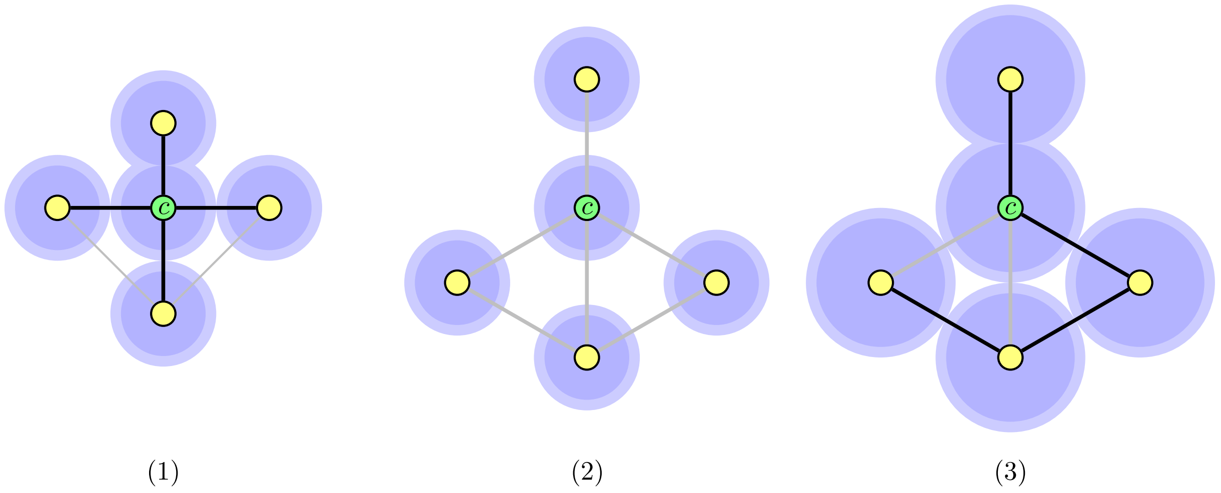

Figure 4.1: The figure shows the intuition behind our algorithm. Figure (1) shows a spanning tree with a high-degree node c. Figure (2) shows how raising λ_c creates slack for all the edges that are incident to c and finally Figure (3) shows the new tree after running `MST` again.

The *low expansion* of $S^i_{d^i}$ turns out to be crucial in the analysis of the performance guarantee of our algorithm.

Intuitively, increasing λ_v for a node $v \in V$ *lengthens* edges of the form $uv \in E$; In this way, in the current packing $\{y^i_\pi\}_{\pi\in\Pi}$, we create extra slack in constraints of type (4.2) for edges incident to v. The hope is to increase the length of edges incident to nodes of high normalized degree until other edges that are incident to nodes of lower normalized degree are used in their place in the spanning tree. Figure 4.1 exemplifies this intuition.

Step 6 of Algorithm 3 hides the details of choosing an appropriate ϵ^i by which edges in the current tree that are incident to nodes of normalized degree at least d^i are lengthened. Our choice of ϵ^i will ensure that there exists at least one point in time during the execution of `MST` in step 9 at which we now have the choice to connect two connected components using edges e^i and $\overline{e}^i$ where $e^i \in E^i$ and $\overline{e}^i \notin E^i$. We now break ties such that $\overline{e}^i$ is chosen instead of e^i and obtain a new tree E^{i+1} that differs in exactly one edge from E^i.

We show that we can choose ϵ^i and hence a pair of edges $\langle e^i, \overline{e}^i\rangle$ such that $\overline{e}^i$ is not incident on any node from $S^i_{d^i-1}$. The main idea here is to increase λ_v for nodes $v \in S^i_{d^i-1}$ by ϵ^i and increase the lengths of non-tree edges that are incident to nodes $v \in S^i_{d^i-1}$ by ϵ^i as well. In other words, the length of non-tree edges incident to nodes of normalized degree at least $d^i - 1$ increases by the same amount as the length of tree edges incident to nodes of normalized degree at least d^i. This way, we enforce that the edge we swap in touches nodes of normalized degree at most $d^i - 2$. Once we accomplish this, adapting a potential function argument from [21] we can put a polynomial upper bound on the number of such iterations (see Section 4.2.5).

Suppose, that our algorithm terminates after t iterations. Our goal is then to prove

$$\sum_{e\in E^i} c_e \le \omega \sum_{\pi\in\Pi} (r(\pi) - 1) \cdot y^i_\pi - \omega \cdot \sum_{v\in V} B_v \cdot \lambda^i_v \tag{4.4}$$

for all $1 \le i \le t$. The right-hand side of (4.4) is ω times the dual objective function value of the feasible dual solution (y^i, λ^i). Therefore, using weak duality, (4.4) implies that $\sum_{e\in E^i} c_e \le \omega \cdot \texttt{opt}$. We now describe how to choose ϵ^i so that the above conditions are satisfied.

4.2.1 Choosing ϵ^i

In this section we elaborate on the choice of ϵ^i in step 6 of Algorithm 3. Note that we maintain

$$c_{uv} \le \widetilde{c}_{uv} \le c_{uv} + \lambda_u + \lambda_v$$

for all edges uv at all times during the run of Algorithm 3. In step 8 of Algorithm 3, we increase $\widetilde{c}_{uv}$ by ϵ^i for all tree edges uv that are incident to nodes of degree at least d^i and for all non-tree edges that are incident to nodes of degree at least $d^i - 1$. Note that our algorithm increases the λ^i-value of at least one endpoint of all the edges whose $\widetilde{c}$-cost increases.

We want to choose ϵ^i such that the subsequent update of $\widetilde{c}^i$ and the following run of `MST` yields a new tree E^{i+1} that differs from E^i by a single edge swap: $E^{i+1} = E^i \setminus \{e^i\} \cup \{\overline{e}^i\}$. Here, the edge $e^i \in E^i$ is a tree edge that is incident to a node from $S^i_{d^i}$. On the other hand $\overline{e}^i \in E \setminus E^i$ is a non-tree edge that is not incident to any node from $S^i_{d^i-1}$. We want that $\widetilde{c}^i(e^i) \le \widetilde{c}^i(\overline{e}^i)$ and $\widetilde{c}^i(e^i) + \epsilon^i = \widetilde{c}^i(\overline{e}^i)$. In other words, the update of λ_v for $v \in S^i_{d^i-1}$ creates one more beneficial swap.

We let $\mathcal{K}^i$ be the set of connected components of the forest $E^i \setminus S^i_{d^i}$, i.e. the forest that results from removing nodes of normalized degree at least d^i from E^i. We say that an edge $e = uv \in E$ is a *cross-edge* if

1. e is a non-tree edge, i.e. $e \in E \setminus E^i$, and
2. $u \in K_1, v \in K_2$ for $K_1, K_2 \in \mathcal{K}^i$ and $K_1 \neq K_2$.

We denote the set of cross-edges in iteration i by $\mathcal{C}^i$.

It is now clear that $E^i + e$ contains a unique cycle C^i_e for each cross-edge $e \in \mathcal{C}^i$. Furthermore, there must be at least one vertex v on C^i_e that has normalized degree at least d^i.

For each cross-edge $e \in \mathcal{C}^i$, we now let

$$\epsilon^i_e = \min_{e' \in C^i_e, e' \cap S^i_{d^i} \neq \emptyset} \left(\widetilde{c}^i(e) - \widetilde{c}^i(e') \right).$$

We then let $\epsilon^i = \min_{e \in \mathcal{C}^i} \epsilon^i_e$.

In the following, we let $\langle e^i, \overline{e}^i \rangle$ be the *witness pair* for ϵ^i. In other words, for all edges $e' \in C^i_{\overline{e}^i}$ such that e' is incident to a node of normalized degree at least d^i, we have $\widetilde{c}^i_{e'} + \epsilon^i \le \widetilde{c}^i_{\overline{e}^i}$ and equality holds for $e' = e^i$.

Note that ϵ^i can be 0. In this case we have

$$\widetilde{c}^i(e^i) = \widetilde{c}^i(\overline{e}^i)$$

and the swap can be viewed as a *local-improvement* step along the lines of the local-search algorithm for the minimum-degree spanning tree problem from Chapter 2. We do not modify the dual solution but decrease the normalized degree of a node of high normalized degree.

We now show that (4.4) is maintained throughout the execution of our algorithm.

4.2.2 Analysis: Performance guarantee

Assume that Algorithm 3 terminates after iteration t^*. In this section we prove that (4.4) must hold for all $0 \le i \le t^*$. Observe that (4.4) holds for $i = 0$ by Lemma 4. We concentrate on the case $i \ge 1$.

Growing λ^i_v by ϵ^i at nodes $v \in S^i_{d^i-1}$ decreases the right-hand side of (4.4) by $\omega\epsilon^i \cdot \sum_{v \in S^i_{d^i-1}} B_v$. Still the cost of the spanning tree E^{i+1} is potentially higher than the cost of the old tree E^i. We must show that the first term on the right hand-side of (4.4), i.e. $\omega \cdot \sum_{\pi \in \Pi}(r(\pi) - 1)y^i_\pi$ grows sufficiently to compensate for the decrease in the second term and the increased spanning tree cost.

To do this, we maintain the following invariant inductively for all $0 \le i \le t^*$:

$$\omega \cdot \sum_{v \in V} B_v \lambda_v^i \le (\omega - 1) \cdot \sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_\pi^i. \tag{Inv}$$

Since, $\lambda_v^0 = 0$ for all $v \in V$ by definition, (Inv) holds for $i = 0$.

The following claim that proves the inductive step of (Inv) is the essential insight that ultimately yields (4.4).

Claim 1 *Choosing $\beta_v \ge b\alpha$ for all $v \in V$ in the definition of normalized degree yields*

$$\sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_\pi^{i+1} \ge \sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_\pi^i + \epsilon^i \alpha \cdot \sum_{v \in S^i_{d^i - 1}} B_v$$

for all $0 \le i \le t^*$.

Proof: Let $E^i = \{e_1^i, \dots, e_{n-1}^i\}$ and let t_j^i be the time at which `MST` included edge e_j^i. W.l.o.g., assume that $t_1^i \le \cdots \le t_{n-1}^i$. From the description of `MST` we can rewrite

$$\begin{aligned} \sum_{\pi \in \Pi} (r(\pi) - 1) \cdot y_\pi^i &= \sum_{j=1}^{n-1} (t_j^i - t_{j-1}^i) \cdot (n - j) \qquad (4.5) \\ &= \sum_{j=1}^{n-1} t_j^i \left((n - j + 1) - (n - j)\right) \\ &= \sum_{j=1}^{n-1} t_j^i \end{aligned}$$

where we define $t_0^i = 0$.

In fact, we can use (4.5) to quantify the change in dual in iteration i:

$$\sum_{\pi \in \Pi} (r(\pi) - 1) \cdot \left(y_\pi^{i+1} - y_\pi^i\right) = \sum_{j=1}^{n-1} \left(t_j^{i+1} - t_j^i\right) \tag{4.6}$$

In iteration i, we lengthen all edges $e \in E^i$ that are incident to nodes of normalized degree at least d^i by ϵ^i. Hence, all of these edges become tight ϵ^i time units later. Together with (4.6) we obtain

$$\sum_{\pi \in \Pi} (r(\pi) - 1) \cdot \left(y_\pi^{i+1} - y_\pi^i\right) \ge \epsilon^i \cdot \left|E\left(S^i_{d^i}\right) \cap E^i\right| \tag{4.7}$$

where $E\left(S^i_{d^i}\right)$ denotes the set of edges in E that are incident to nodes from $S^i_{d^i}$. (Note that we include in $E(S)$ edges with both endpoints in S).

Recall the definition of normalized degree in (5.8). Notice that it follows from the termination condition in step 4 of Algorithm 3 that $d^i > 0$. Hence, we must have that the real degree of any node $v \in S^i_{d^i}$ is at least

$$\beta_v \cdot B_v + d^i \ge \beta_v \cdot B_v + 1.$$

Finally, notice that it follows from the fact that E^i is a tree that there are at most $|S^i_{d^i}| - 1$ edges in $E\left(S^i_{d^i}\right)$ that are incident to two nodes from $S^i_{d^i}$. We can use these observation to lower-bound the right-hand side of (4.7):

$$\sum_{\pi \in \Pi} (r(\pi) - 1) \cdot \left(y_\pi^{i+1} - y_\pi^i\right) \ge \epsilon^i \cdot \left(\left(\sum_{v \in S^i_{d^i}} \beta_v \cdot B_v + 1\right) - (|S^i_{d^i}| - 1)\right).$$

Now, we choose $\beta_v \geq \alpha b$ for all $v \in V$ and obtain

$$\sum_{\pi\in\Pi}(r(\pi)-1)\cdot\left(y_\pi^{i+1}-y_\pi^i\right) \geq \epsilon^i \alpha b \cdot \sum_{v\in S^i_{d^i}} B_v.$$

Lemma 5 now yields the claim. ■

We are now ready to prove (4.4).

4.2.3 Proof of (4.4)

As MST finishes we obtain from Lemma 4 that

$$\widetilde{c}^{i+1}(E^{i+1}) = \sum_{e\in E^{i+1}} \widetilde{c}_e^{i+1} = \sum_{\pi\in\Pi}(r(\pi)-1)\cdot y_\pi^{i+1}. \tag{4.8}$$

Observe that the real cost of the spanning tree E^{i+1} can be much smaller than $\widetilde{c}^{i+1}(E^{i+1})$. In fact, notice that we have

$$c(E^{i+1}) = c(\overline{e}^i) + c(E^i \setminus \{e_i\}) \leq \widetilde{c}^{i+1}(\overline{e}^i) + \widetilde{c}^i(E^i \setminus \{e^i\}) \tag{4.9}$$

where the first step follows from the fact that $E^{i+1} = E^i \setminus \{e^i\} \cup \{\overline{e}^i\}$ by the definition of ϵ^i and the way we break ties in MST. The second inequality uses the fact that we always have $c_e \leq \widetilde{c}_e^i$ for all $1 \leq i \leq t$ and for all $e \in E$.

Also, observe that

$$\widetilde{c}^{i+1}(E^i \setminus \{e^i\}) = \widetilde{c}^i(E^i \setminus \{e^i\}) + \left(\left|E\left(S^i_{d^i}\right) \cap E^i\right| - 1\right)\cdot \epsilon^i \tag{4.10}$$

since exactly one edge from E^i that is incident to a node of normalized degree at least d^i is swapped out.

We can lower-bound $\left|E\left(S^i_{d^i}\right) \cap E^i\right|$ using the arguments from the proof of Claim 1:

$$\begin{aligned}\left|E\left(S^i_{d^i}\right) \cap E^i\right| &\geq \left(\sum_{v\in S^i_{d^i}} \beta_v \cdot B_v + 1\right) - (|S^i_{d^i}| - 1)\\ &\geq \alpha \cdot \sum_{v\in S^i_{d^i-1}} B_v + 1\end{aligned}$$

where the second inequality again follows from Lemma 5 and from our choice of β_v for all $v \in V$. The last inequality together with (4.10) implies

$$\widetilde{c}^{i+1}(E^i \setminus \{e^i\}) \geq \widetilde{c}^i(E^i \setminus \{e^i\}) + \alpha\epsilon^i \cdot \sum_{v\in S^i_{d^i-1}} B_v. \tag{4.11}$$

We now obtain

$$\begin{aligned}c(E^{i+1}) &\leq \widetilde{c}^i(E^i \setminus \{e^i\}) + \widetilde{c}^{i+1}(\overline{e}^i)\\ &\leq \widetilde{c}^{i+1}(E^i) - \alpha\epsilon^i \cdot \sum_{v\in S^i_{d^i-1}} B_v\\ &= \sum_{\pi\in\Pi}(r(\pi)-1)\cdot y_\pi^{i+1} - \alpha\epsilon^i \cdot \sum_{v\in S^i_{d^i-1}} B_v\end{aligned}$$

where the first inequality follows from (4.9), the second inequality follows from (4.11), and the last equality follows from (4.8).

Adding (Inv) to the last inequality we get

$$\begin{aligned} c(E^{i+1}) &\leq \sum_{\pi\in\Pi}(r(\pi)-1)\cdot y_\pi^{i+1} + (\omega-1)\cdot\sum_{\pi\in\Pi}(r(\pi)-1)\cdot y_\pi^i - \omega\epsilon^i\cdot\sum_{v\in S^i_{d^i-1}} B_v - \alpha\cdot\sum_{v\in V} B_v\lambda_v^i \\ &\leq \omega\cdot\sum_{\pi\in\Pi}(r(\pi)-1)\cdot y_\pi^{i+1} - \omega\epsilon^i\cdot\sum_{v\in S^i_{d^i-1}} B_v - \alpha\cdot\sum_{v\in V} B_v\lambda_v^i \end{aligned}$$

where the last inequality follows from the fact that

$$\sum_{\pi\in\Pi}(r(\pi)-1)y_\pi^i \leq \sum_{\pi\in\Pi}(r(\pi)-1)y_\pi^{i+1}.$$

Finally notice that $\lambda_v^{i+1} = \lambda_v^i + \epsilon^i$ if $v \in S^i_{d^i-1}$ and $\lambda_v^{i+1} = \lambda_v^i$ otherwise. Additionally, we choose $\alpha \geq \omega$ and get

$$c(E^{i+1}) \leq \omega\cdot\sum_{\pi\in\Pi}(r(\pi)-1)\cdot y_\pi^{i+1} - \omega\cdot\sum_{v\in V} B_v\lambda_v^{i+1}.$$

as required. This proves the inductive step of (4.4).

4.2.4 Proof of (Inv)

It remains to show that (Inv) is maintained as well. Observe that the left hand side of (Inv) increases by

$$\omega\epsilon^i\sum_{v\in S^i_{d^i-1}} B_v.$$

We obtain from Claim 1 that

$$\sum_{\pi\in\Pi}(r(\pi)-1)\cdot(y_\pi^{i+1}-y_\pi^i) \geq \epsilon^i\alpha\cdot\sum_{v\in S^i_{d^i-1}} B_v$$

Therefore, the right hand side of (Inv) increases by $(\omega-1)\cdot\alpha\epsilon^i\cdot\sum_{v\in S^i_{d^i-1}} B_v$ which is sufficient if $(\omega-1)\alpha \geq \omega$ or equivalently, if $\alpha \geq \omega/(\omega-1)$.

The proof of the performance guarantee claimed in Theorem 2 follows by choosing $\alpha \geq \max\left\{\omega, \frac{\omega}{\omega-1}\right\}$.

4.2.5 Analysis: Running time

In this section, we show that Algorithm 3 terminates in polynomial time. We accomplish this by showing that there will be only a polynomial number of iterations of the main loop in Algorithm 3.

Claim 2 *Algorithm 3 terminates after $O(n^4)$ iterations.*

Proof: Following [21], we define the potential of spanning tree E^i as

$$\Phi_i = \sum_{v\in V} 3^{\texttt{ndeg}_i(v)}$$

where $\texttt{ndeg}_i(v)$ denotes again the normalized degree of node v in the tree E^i.

Notice that an iteration of Algorithm 3 swaps out a single edge e^i that is incident to at least one node of normalized degree at least d^i. On the other hand we swap in one edge $\overline{e}^i$ that is incident to two nodes of normalized degree at most $d^i - 2$. The reduction in the potential hence is at least

$$(3^{d^i} + 2 \cdot 3^{d^i - 2}) - 3 \cdot 3^{d^i - 1} \geq 2 \cdot 3^{d^i - 2}$$

Using the range of d^i, we can lower-bound the right hand side of the last inequality by

$$2 \cdot 3^{\Delta^i - 2 \log_b(n) - 2} = \Omega\left(\frac{3^{\Delta^i}}{n^2}\right).$$

The initial potential Φ_0 is at most $n \cdot 3^{\Delta^0}$ and the decrease in the potential Φ_i in iteration i is at least $\Omega\left(\frac{\Phi_i}{n^3}\right)$.

In other words, in $O(n^3)$ iterations, we reduce Φ by a constant factor. Hence, the algorithm runs for $O(n^4)$ iterations total. Observing that each iteration can be implemented in time $O(n^2 \log n)$ [29], we see that the whole algorithm runs in time $O(n^6 \log n)$. ■

Minimum-Cost Degree-Bounded Steiner Trees

In this chapter we address the minimum-degree Steiner tree problem (`B-ST`) where we are given an undirected graph $G = (V, E)$, a non-negative cost c_e for each edge $e \in E$ and a set of terminal nodes $R \subseteq V$. Additionally, the problem input also specifies positive integers $\{B_v\}_{v\in V}$. The goal is to find a minimum-cost Steiner tree T covering R such each node v in T has degree at most B_v, i.e.

$$\deg_T(v) \leq B_v$$

for all $v \in V$.

We present an algorithm for the problem and give a proof of Theorem 3. The basic framework resembles the primal-dual algorithm from Chapter 4.

In the following, we review a linear programming formulation for the problem and subsequently discuss the Steiner tree algorithm from [3].

5.1 A LINEAR PROGRAMMING FORMULATION

The following natural integer programming formulation models the problem. Let $\mathcal{U} = \{U \subseteq V : U \cap R \neq \emptyset, R \setminus U \neq \emptyset\}$.

$$\begin{aligned} \min \quad & \sum_{e\in E} c_e x_e & \text{(IP)} \\ \text{s.t} \quad & x(\delta(U)) \geq 1 \quad \forall U \in \mathcal{U} & (5.1) \\ & x(\delta(v)) \leq B_v \quad \forall v \in V & (5.2) \\ & x \text{ integer} & \end{aligned}$$

The dual of the linear programming relaxation (LP) of (IP) is given by

$$\begin{aligned} \max \quad & \sum_{U\in\mathcal{U}} y_U - \sum_{v\in V} \lambda_v \cdot B_v & \text{(D)} \\ \text{s.t} \quad & \sum_{U: e\in\delta(U)} y_U \leq c_e + \lambda_u + \lambda_v \quad \forall e = uv \in E & (5.3) \\ & y, \lambda \geq 0 & \end{aligned}$$

We also let (IP-ST) denote (IP) without constraints of type (5.2). This is the usual integer programming formulation for the Steiner tree problem. Let the LP relaxation be denoted by (LP-ST) and let its dual be (D-ST).

5.2 AN ALGORITHM FOR THE STEINER TREE PROBLEM

Our algorithm is based upon previous work on the generalized Steiner tree problem by Agrawal et al. [3]. In the following, we present their algorithm to compute an approximate Steiner tree. We refer to this algorithm by `AKR`.

At each step during the algorithm we maintain a partial Steiner tree. More specifically, we have a set of edges $E' \subseteq E$. We use $\mathcal{C}(E')$ to denote the set of connected components of $G[E']$ and let

$$\mathcal{V}(E') = \{S \in \mathcal{C}(E') \, : \, S \in \mathcal{U}\}$$

denote the set of *minimally violated components* for edge set E'. Sets $S \in \mathcal{V}(E')$ are violated in the sense the inequality (5.1) in (LP-ST) is violated for set S and partial solution E'. S is minimal in the sense that there is no proper subset $S' \subset S$ such that (5.1) is violated for set S' and partial solution E'. We sometimes refer to sets $S \in \mathcal{V}(E')$ as *moats*.

The algorithm also maintains a feasible dual solution y for (D-ST) and let $y_S = 0$ initially for all $S \in \mathcal{U}$.

The construction of the final Steiner tree is now viewed as a continuous process. With each *time* $t \in [0, t^*]$, we associate a partial primal solution x^t for (LP-ST) and a feasible dual solution y^t for (D-ST). At time t, let E^t be the set of edges that we have included so far and let $\mathcal{V}^t = \mathcal{V}(E^t)$ be the set of minimally violated components. We let y^t be the dual solution a time t.

At any given point $t \in [0, t^*]$ such that x^t is not the incidence vector of a feasible Steiner tree, we increase y_S for all $S \in \mathcal{V}^t$ simultaneously until a constraint of type (5.3) for edge $e \in E \setminus E^t$ becomes tight. Assume that this happens at time $t' \geq t$. The dual update is

$$y_S^{t'} = \begin{cases} y_S^t + (t' - t) & : \quad S \in \mathcal{V}^t \\ y_S^t & : \quad \text{otherwise.} \end{cases}$$

If the tightness of e is caused by two colliding active moats $S_1, S_2 \in \mathcal{V}^t$, we *merge* them and create a new moat $S = S_1 \cup S_2$. In this case, there must exist a minimum-c-cost path P connecting terminal nodes $s_1 \in S_1$ and $s_2 \in S_2$ such that every edge $e' \in P$ is tight and $e \in P$. Moreover, all internal nodes v of P (we write $v \in \mathtt{int}(P)$) are Steiner nodes. We then say, that P is an s_1, t_1-*Steiner path* and we let $\mathcal{P}_{S_1,T_1}$ be the set of all Steiner paths that connect terminals in S_1 and T_1, respectively.

We include the edges of P into our solution:

$$x_{e'}^{t'} = \begin{cases} 1 & : \quad e' \in P \\ x_{e'}^t & : \quad \text{otherwise.} \end{cases}$$

Notice, that we do not update the primal solution in the case that the tightness of edge e is caused by one moat alone. This must mean that e connects S to a Steiner node. In this way, our view of the primal-dual algorithm is closer to that of Agrawal, Klein, and Ravi [3] rather than the treatment in [29] where all tight edges are added and unnecessary ones are deleted in a *reverse delete* step at the end.

In the following, let $\mathcal{P}$ be the set of Steiner nodes that Algorithm `AKR` uses to connect all terminals in R and denote by $\mathcal{P}_S$ the subset of paths from $\mathcal{P}$ that use only nodes from $S \subseteq V$. We obtain the following lemma:

Claim 3 *For any $t \in [0, t^*]$ and any active component $S \in \mathcal{V}^t$ we must have*

$$\sum_{P \in \mathcal{P}_S} c(P) \leq 2 \sum_{S' \subseteq S} y_S^t - 2t$$

Proof: The claim is trivially true for $t = 0$.

Now, let $0 \leq t_1 \leq t_2 \leq \cdots \leq t_q \leq t^*$ be those points in time when active moats collide and the partial primal solution changes. We first show that the claim is true for $t \neq t_i$ for all all $1 \leq i \leq q$.

Let $t' = t_i = \max_{j:t_j \leq t} t_j$. By induction, we have that

$$\sum_{P \in \mathcal{P}_S} c(P) \leq 2 \sum_{S' \subseteq S} y_S^{t'} - 2t' \tag{5.4}$$

Notice, that no Steiner paths are added in the time interval $(t', t]$ and that $y_S^t - y_S^{t'} = (t - t')$. It then follows easily that (5.4) holds even for t' replaced by t.

Now, let $t = t_i$ for some $1 \leq i \leq q$. Let S_1 and S_2 be the two moats colliding at time t_i and let P be the S_1, S_2 Steiner path that is used to merge S_1 and S_2. It is not hard to see that

$$c(P) = \sum_{e \in P} c_e \leq 2t. \tag{5.5}$$

Hence,

$$\begin{aligned}
\sum_{P \in \mathcal{P}_{S_1 \cup S_2}} c(P) &\leq \sum_{P \in \mathcal{P}_{S_1}} c(P) + \sum_{P \in \mathcal{P}_{S_2}} c(P) + 2t \\
&\leq \sum_{S' \subseteq S_1} y_{S'}^t + \sum_{S' \subseteq S_2} y_{S'}^t - 4t + 2t \\
&= \sum_{S' \subseteq S} y_{S'}^t - 2t
\end{aligned}$$

where the second inequality follows from our induction hypothesis. ■

This immediately implies the following corollary.

Corollary 4 *The preceeding algorithm computes a $(2 - 2/k)$-approximate Steiner tree.*

Proof: Assume that the last moat collision in `AKR` happens at time t^*. We then must have

$$\sum_{S \in \mathcal{U}} y_S^{t^*} \leq k \cdot t^* \tag{5.6}$$

and hence we must have

$$\begin{aligned}
\sum_e c_e x_e^{t^*} &\leq \sum_{P \in \mathcal{P}} c(P) \\
&\leq 2 \sum_{S \in \mathcal{U}} y_S^{t^*} - 2t^* \\
&\leq (2 - 2/k) \cdot \sum_{S \in \mathcal{U}} y_S^{t^*} \\
&\leq (2 - 2/k) \cdot \texttt{opt}\,.
\end{aligned}$$

The third inequality uses (5.6) and the last inequality uses weak duality. ■

In the following section, we derive a few useful properties of algorithm `AKR`.

5.2.1 Some useful properties of AKR

The first observation characterizes the set of Steiner paths that is used to connect terminal nodes from R.

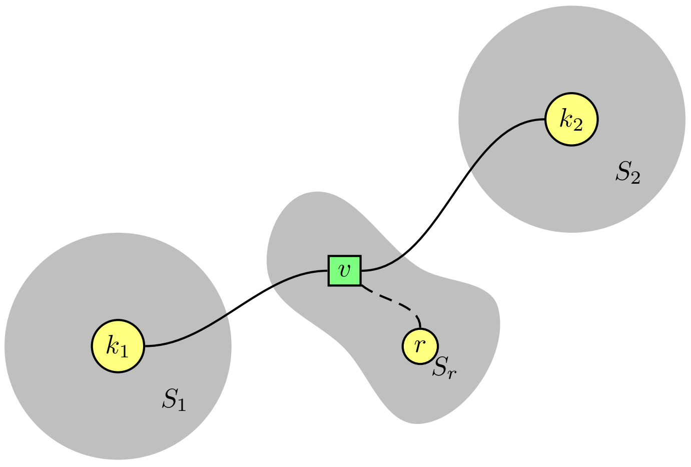

Figure 5.1: The figure shows a moat configuration during the execution of AKR. A Steiner path (solid lines) connecting two terminal nodes $k_1 \in S_1$ and $k_2 \in S_2$ is loaded by three moats S_1, S_2 and S_r.

Claim 4 *Let $t \in [0, t^*]$ be a point in time at which* AKR *merges sets $S_1 \in \mathcal{V}^t$ and $S_2 \in \mathcal{V}^t$ using path P. Let $\mathcal{P}_j$ be the set of all Steiner paths that originate at a terminal node in S_j and end in a terminal node in $R \setminus S_j$. We then must have that*

$$c(P) \leq \min_{P' \in \mathcal{P}_1 \cup \mathcal{P}_2} c(P').$$

Moreover, we must have $t = c(P)/2$.

Proof: The proof is by contradiction. Let

$$P_1, \ldots, P_k$$

be the paths of $\mathcal{P}$ in the order they are added. Assume that $P = P_i$ is the first path in this ordering for which the claim does not hold. W.l.o.g. assume that for some $k \in R \setminus S_1$ there is an S_1, k-Steiner path P' with

$$c(P') < c(P).$$

Still P is used to merge S_1 and S_2 before P' is used to merge S_1 and r.

The only way in which P can become tight earlier than P' is that at some point in time $t' < t$ there are at least three active moats S_1, S_2 and S_r intersecting P (refer to Figure 5.1). Suppose that v is a Steiner node in $S_r \cap P$ and r is the terminal in S_r which is at minimum-c-distance from v. Let

$$P = \langle P_{k_1,v}, P_{v,k_2} \rangle$$

and let $P_{v,r} \in E^t$ be a minimum-c-cost v, r-path in G.

It must be the case that either S_1 and r or S_2 and r merge before S_1 and S_2 do. W.l.o.g., assume that S_1 and r merge first. By assumption we know that this happens at time $c(\langle P_{k_1,v}, P_{v,r} \rangle)/2$. This also implies that $c(P_{v,r}) < c(P_{k_1,v})$ and hence,

$$c(\langle P_{r,v}, P_{v,k_2}) < c(\langle P_{k_1,v}, P_{v,k_2})$$

which contradicts the choice of P as a minimum-c-cost path connecting terminals in the two moats that are involved in the merge.

Finally, suppose that P becomes tight at time t_P during the execution of `AKR`. The above argument shows that at all times $0 \leq t \leq t_P$ there are exactly two active moats that intersect P. It immediately follows that P becomes tight at time $c(P)/2$. ■

Before proceeding with the description of an algorithm for minimum-cost degree-bounded Steiner trees, we present an alternate view of Algorithm `AKR` which simplifies the following developments in this paper.

5.2.2 An alternate view of `AKR`

Executing `AKR` on an undirected graph $G = (V, E)$ with terminal set $R \subseteq V$ and costs $\{c_e\}_{e \in E}$ is – in a certain sense – equivalent to executing `AKR` on the complete graph H with vertex set R where the cost of edge $e = uv \in R \times R$ is equal to the minimum cost of any u, v-path in G.

Recall that $\mathcal{P}_{S,T}$ denotes the set of s, t-Steiner paths for all $s \in R \cap S$ and $t \in R \cap T$. For an s, t-Steiner path P, we define

$$c^\lambda(P) = c(P) + \lambda_s + \lambda_t + 2 \cdot \sum_{v \in \mathtt{int}(P)} \lambda_v$$

and finally, let

$$\mathtt{dist}_{c^\lambda}(S, T) = \min_{P \in \mathcal{P}_{S,T}} c^\lambda(P)$$

be the minimum c^λ-cost for any S, T-Steiner path. We now work with the following dual:

$$\begin{aligned} \max \quad & \sum_{U \subset R} y_U - B_0 \sum_{v \in V} \lambda_v && \text{(D2)} \\ \text{s.t} \quad & \sum_{U \subset R, s \in U, t \notin U} y_U \leq c^\lambda(P) \quad \forall s, t \in R, P \in \mathcal{P}_{s,t} \in E && (5.7) \\ & y, \lambda \geq 0 \end{aligned}$$

Let H be a complete graph on vertex set R. We let the length of edge $(s, t) \in E[H]$ be $\mathtt{dist}_{c^\lambda}(s, t)$. Running `AKR` on input graph H with length function $\mathtt{dist}_{c^\lambda}$ yields a tree in H that corresponds to a Steiner tree spanning the nodes of R in G in a natural way. We also obtain a feasible dual solution for (D2). The following lemma shows that (D) and (D2) are equivalent.

Claim 5 *(D) and (D2) are equivalent.*

Proof: Let $\{y_U\}_{\mathcal{U}}$ be a dual solution to (D). Define

$$\overline{y}_S = \sum_{U \in \mathcal{U}, U \cap R = S} y_U.$$

It is now easy to check that $\overline{y}$ is feasible for (D2) and that $\sum_{U \subset R} \overline{y}_U = \sum_{U \in \mathcal{U}} y_U$.

We now show how to convert a feasible solution $\{\overline{y}_U\}_{U \subset R}$ to (D2) into a feasible solution $\{y_S\}_{S \in \mathcal{U}}$ for (D). In order to do this, we keep a *budget* b_{uv} for each edge $uv \in E$ and let

$$b_{uv} = c_{uv} + \lambda_u + \lambda_v$$

for all $uv \in E$ initially. For a subset $S \subset R$ and a budget vector $\{b_e\}_{e \in E}$, we let

$$B(S, b) = \{u \in V \setminus R : \exists S, u\text{-path } P \text{ s.t. } b_e = 0 \text{ for all } e \in P\} \cup S.$$

In other words, $B(S, b)$ contains S and all Steiner vertices u that are connected to S by at least one path of 0-budget edges.

Algorithm 4 An algorithm to convert a solution $\{\overline{y}_U\}_{U \subset R}$ for (D2) into a solution $\{y_S\}_{S \in \mathcal{U}}$ for (D).

```
1: Given: feasible solution $\{\overline{y}_U\}_{U \subset R}$ for (D2).
2: $b_e \leftarrow c_e$ for all $e \in E$
3: $y'_U \leftarrow \overline{y}_U$ for all $U \subset R$
4: $y_S \leftarrow 0$ for all $S \in \mathcal{U}$
5: while $\exists U \subset R$ with $y'_U > 0$ do
6:     $\epsilon \leftarrow \min\{\min_{e \in \delta(B(U,b))} b_e, y'_U\}$
7:     $y_{B(U,\delta)} \leftarrow y_{B(U,\delta)} + \epsilon$
8:     $y'_U \leftarrow y'_U - \epsilon$
9:     $b_e \leftarrow b_e - \epsilon$ for all $e \in \delta(B(U,b))$
10: end while
```

The algorithm starts with $y'_U = \overline{y}_U$ for all $U \subset R$ and then incrementally constructs $\{y_S\}_{S \in \mathcal{U}}$. At any point in time, we look at a subset U of the terminals such that $y'_{U'} = 0$ for all $U' \subset U$. Given such a subset, we then increase $y_{B(U,b)}$ by the maximum $\epsilon > 0$ such that the resulting set of dual variables is still feasible for (D). The maximum such ϵ equals the minimum budget among the edges in $\delta(B(U,b))$. If $\epsilon > y'_U$ then let $\epsilon = y'_u$.

Afterwards, we decrease the budget b_e of all edges $e \in \delta(B(U,b))$ as well as variable y_U by ϵ. We continue this process until $y'_U = 0$ for all $U \subset R$. Algorithm 4 has the pseudo code for the conversion procedure.

It can be shown by induction that at any point during the algorithm and for any s,t-Steiner path P we must have

$$\sum_{S \in \mathcal{U}, |S \cap \{s,t\}|=1} y_S + \sum_{U \subset R, |U \cap \{s,t\}|=1} y'_U = \sum_{U \subset R, |U \cap \{s,t\}|=1} \overline{y}_U.$$

We leave the details to the reader.

This implies the termination of Algorithm 4. An immediate consequence is the feasibility of $\{y_S\}_{S \in \mathcal{U}}$ for (D) and

$$\sum_{S \in \mathcal{U}} y_S = \sum_{U \subset R} \overline{y}_U.$$

This finishes the proof of the claim. ■

5.3 AN ALGORITHM FOR THE B-ST PROBLEM

In this section, we propose a modification of AKR in order to compute a feasible degree-bounded Steiner tree of low total cost. We start by giving a rough overview over our algorithm.

5.3.1 Algorithm: Overview

Recall that we defined the normalized degree $\texttt{ndeg}_T(v)$ of a node v in a Steiner tree T as

$$\texttt{ndeg}_T(v) = \max\{0, \deg_T(v) - \beta_v\} \tag{5.8}$$

where $\{\beta_v\}_{v \in V}$ are parameters to be defined later.

Algorithm B-ST goes through a sequence of Steiner trees $E^0, \ldots, E^t$ and associated pairs of primal (infeasible) and dual feasible solutions x^i, (y^i, λ^i) for $0 \leq i \leq t$. The goal is to reduce the maximum normalized degree of at least one node in V in the transition from one Steiner tree to the next.

In the i^{th} iteration our algorithm passes through two main steps:

Compute Steiner tree. Compute an approximate Steiner tree spanning the nodes of R for our graph $G = (V, E)$ using a modified version of AKR. Roughly speaking, this algorithm implicitly assumes a cost function $\widetilde{c}$ that satisfies

$$c(P) \leq \widetilde{c}(P) \leq c^{\lambda^i}(P) \tag{5.9}$$

for all Steiner paths P.

When the algorithm finishes, we obtain a primal solution x^i together with a corresponding dual solution y^i. In the following we use $\mathcal{P}^i$ to denote the set of paths used by AKR to connect the terminals in iteration i.

Notice that using the cost function $\widetilde{c}$ that satisfies (5.9) ensures that (y^i, λ^i) is a feasible solution for (D2). The primal solution x^i may induce high normalized degree at some of the vertices of V and hence may not be feasible for (IP).

Update node multipliers. The main goal here is to update the node multipliers λ^i such that another run of AKR yields a tree in which the normalized degree of at least one node decreases. Specifically, we continue running our algorithm as long as the maximum normalized node-degree induced by x^i is at least $2 \log_b n$ where $b > 1$ is a positive constant to be specified later.

As in the last chapter, let Δ^i be the maximum normalized degree of any node in the tree induced by x^i. The algorithm then picks a threshold $d^i \geq \Delta^i - \lceil 4 \log_b n \rceil + 2$. Subsequently we raise the λ values of all nodes that have normalized degree at least $d^i - 2$ in the tree induced by x^i by some $\epsilon^i > 0$. We also implicitly increase the $\widetilde{c}$ cost of two sets of Steiner paths:

1. those paths $P \in \mathcal{P}^i$ that contain nodes of degree at least d^i and
2. those paths $P \notin \mathcal{P}^i$ that contain nodes of degree at least $d^i - 2$.

We denote to the set of all such paths by $\mathcal{L}^i$.

A subsequent rerun of AKR replaces at least one Steiner path whose $\widetilde{c}$-cost increased with a Steiner path whose length stayed the same. In other words, a path that touches a node of normalized degree at least d^i is replaced by some other path that has only nodes of normalized degree less than $d^i - 2$.

Throughout the algorithm we will maintain that the cost of the current tree induced by x^i is within a constant factor of the dual objective function value induced by (y^i, λ^i). By weak duality, this ensures that the cost of our tree is within a constant times the cost of any Steiner tree that satisfies the individual degree bounds. However, we are only able to argue that the number of iterations is quasi-polynomial.

In the following, we will give a detailed description of the algorithm. In particular, we elaborate on the choice of ϵ^i and d^i in the node-multiplier update and on the modification to AKR that we have alluded to in the previous intuitive description.

5.3.2 Algorithm: A detailed top-level description

We first present the pseudo-code of our B-ST-algorithm. In the description of the algorithm we use the abbreviation $\mathtt{ndeg}^i(v)$ in place of $\mathtt{ndeg}_{E^i}(v)$ for the normalized degree of vertex v in the Steiner tree E^i. We also let Δ^i denote the maximum normalized degree of any vertex in E^i, i.e.

$$\Delta^i = \max_{v \in R} \mathtt{ndeg}^i(v).$$

Furthermore, we again adopt the notation of [21, 22] and let

$$S^i_d = \{v \in V \,:\, \mathtt{ndeg}^i(v) \geq d\}$$

be the set of all nodes whose normalized degrees are at least d in the i^{th} solution.

The following lemma is a straight forward consequence of Lemma 1.

Lemma 6 *There is a* $d^i \in \{\Delta^i - \lceil 4\log_b n\rceil + 2, \ldots, \Delta^i\}$ *such that*

$$\sum_{v \in S^i_{d^i-2}} B_v \le b \cdot \sum_{v \in S^i_{d^i}} B_v$$

for a given constant $b > 1$.

Proof: Suppose for a contradiction that for all $d^i \in \{\Delta^i - \lceil 4\log_b n\rceil + 2, \ldots, \Delta^i\}$, we have

$$\sum_{v \in S^i_{d^i-2}} B_v > b \cdot \sum_{v \in S^i_{d^i}} B_v.$$

Note that since we may assume $B_v \le (n-1)$ for all vertices, we must have $\sum_{v\in V} B_v \le n(n-1)$. However, since $\sum_{v \in S^i_{\Delta^i}} B_v \ge 1$, we have in this case that

$$\sum_{v \in S^i_{\Delta^i - \lceil 4\log_b n\rceil}} B_v \ge b^{\lceil 2\log_b(n)\rceil} \ge n^2$$

a contradiction. ∎

Lemma *low expansion* of $S^i_{d^i-2}$ turns out to be crucial in the analysis of the performance guarantee of our algorithm.

Finally, we let `mod-AKR` denote a call to the modified version of the Steiner tree algorithm `AKR`. Algorithm 5 has the pseudo code for our method.

Algorithm 5 An algorithm to compute an approximate minimum-cost degree-bounded Steiner tree.

```
 1: Given: primal feasible solution $x^0, \mathcal{P}^0$ to (LP-ST) and dual feasible solution $y^0$ to (D-ST)
 2: $\lambda^0_v \leftarrow 0, \forall v \in V$
 3: $i \leftarrow 0$
 4: while $\Delta^i > 4\lceil\log_b n\rceil$ do
 5:     Choose $d^i \ge \Delta^i - \lceil 4\log_b n\rceil + 2$ s.t. $\sum_{v\in S^i_{d^i-2}} B_v \le b \cdot \sum_{v\in S^i_{d^i}} B_v$
 6:     Choose $\epsilon^i > 0$ and identify swap pair $\langle P^i, \overline{P}^i\rangle$.
 7:     $\lambda^{i+1}_v \leftarrow \lambda^i_v + \epsilon^i$ if $v \in S^i_{d^i-2}$ and $\lambda^{i+1}_v \leftarrow \lambda^i_v$ otherwise
 8:     $y^{i+1} \leftarrow$ mod-AKR$(\mathcal{P}^i, \epsilon^i, y^i, \langle P^i, \overline{P}^i\rangle)$
 9:     $\mathcal{P}^{i+1} \leftarrow \mathcal{P}^i \setminus \{P^i\} \cup \{\overline{P}^i\}$
10:     $i \leftarrow i + 1$
11: end while
```

Step 6 of Algorithm 5 hides the details of choosing an appropriate ϵ^i. We lengthen all Steiner paths in $\mathcal{L}^i$. Our choice of ϵ^i will ensure that there exists at least one point in time during the execution of a slightly modified version of `AKR` in step 8 at which we now have the choice to connect two moats using paths P^i and $\overline{P}^i$, respectively. We show that there is a way to pick ϵ^i such that

$$P^i \cap S^i_{d^i} \neq \emptyset \text{ and } \overline{P}^i \cap S^i_{d^i-2} = \emptyset.$$

We now break ties such that $\overline{P}^i$ is chosen instead of P^i and hence, we end up with a new Steiner tree E^{i+1}.

In `mod-AKR`, we prohibit including alternate paths that contain nodes from $S^i_{d^i-2}$ and argue that the dual load that such a non-tree path P' sees does not go up by more than ϵ^i. Hence, we preserve dual feasibility.

We first present the details of Algorithm `mod-AKR` and discuss how to find ϵ^i afterwards.

5.3.3 Algorithm: `mod-AKR`

Throughout this section and the description of `mod-AKR` we work with the modified dual (D2) as discussed in Section 5.2.2.

We first introduce some simplifying notation. For a r_1, r_2-Steiner path P with $r_1, r_2 \in R$, we let $R_P \subseteq 2^R$ denote all sets $S \subset R$ such that

$$\{r_1, r_2\} \cap S \neq \emptyset \text{ and } \{r_1, r_2\} \not\subseteq S.$$

For a dual solution y, λ we then define the cut-metric

$$l_y(P) = \sum_{S \in R_P} y_s.$$

From here it is clear that (y, λ) is a feasible dual solution iff

$$l_y(P) \leq c^{\lambda}(P)$$

for all Steiner paths P. We use $l^i(P)$ as an abbreviation for $l_{y^i}(P)$.

At all times during the execution of Algorithm 5 we want to maintain dual feasibility, i.e. we maintain

$$l^i(P) \leq c^{\lambda^i}(P) \tag{5.10}$$

for all Steiner paths P and for all i. Moreover, we want to maintain that for all i, the cost of any path $P \in \mathcal{P}^i$ is bounded by the dual load that P sees. In other words, we want to enforce that

$$c(P) \leq l^i(P) \tag{5.11}$$

for all $P \in \mathcal{P}^i$ and for all i. It is easy to see that both (5.10) and (5.11) hold for $i = 0$ from the properties of `AKR`.

First, let $\mathcal{P}^i = \mathcal{P}^i_1 \cup \mathcal{P}^i_2$ be a partition of the set of Steiner paths used to connect the terminal nodes in the i^{th} iteration. Here, a path $P \in \mathcal{P}^i$ is added to $\mathcal{P}^i_1$ iff $P \cap S^i_{d^i} \neq \emptyset$ and we let $\mathcal{P}^i_2 = \mathcal{P}^i \setminus \mathcal{P}^i_1$.

`mod-AKR` now first constructs an auxiliary graph G^i with vertex set R. We add an edge (s,t) to G^i for each s,t-path $P \in \mathcal{P}^i \setminus \{P^i\}$. The length l^{i+1}_{st} assigned to an edge (s,t) is

$$l^{i+1}_{st} = l^i(P) + \epsilon^i$$

if (s,t) corresponds to a Steiner path from $\mathcal{P}^i_1$ and $l^{i+1}_{st} = l^i(P)$ otherwise.

Assume that $\overline{P}^i$ is an s', t'-path. We then also add an edge connecting s' and t' to G^i and let its length be the maximum of $l^i(\overline{P}^i)$ and $c(\overline{P}^i)$. Observe, that since $\mathcal{P}^i \setminus \{P^i\} \cup \{\overline{P}^i\}$ is tree, G^i is a tree as well.

Subsequently, `mod-AKR` runs `AKR` on the graph G^i and returns the computed dual solution. We will show that this solution together with λ^{i+1} is feasible for (D2). A formal definition of `mod-AKR` is given in Algorithm 6.

We defer the proof of invariants (5.10) and (5.11) to the end of the next section.

5.3.4 Algorithm: Choosing ϵ^i

In this section, we show how to choose ϵ^i. Remember that, intuitively, we want to increase the cost of currently used Steiner paths that touch nodes of normalized degree at least d^i. The idea is to increase the cost of such paths by the smallest possible amount such that other non-tree paths whose length we did not increase can be used at their place. We make this idea more precise in the following.

Algorithm 6 mod-AKR($\mathcal{P}^i, \epsilon^i, y^i, \langle P^i, \overline{P}^i\rangle$): A modified version of AKR.

```
1: Assume $\overline{P}^i$ is an $s', t'$-Steiner path
2: $G^i = (R, E^i)$ where
   $E^i = \{(s,t) : \exists s,t - \text{path } P \in \mathcal{P}^i \setminus \{P^i\}\} \cup \{(s',t')\}$
3: For all $s,t$ Steiner paths $P \in \mathcal{P}^i \setminus \{P^i\}$ :
   $$l_{st}^{i+1} = \begin{cases} l^i(P) + \epsilon^i & : \quad P \in \mathcal{P}_1^i \\ l^i(P) & : \quad \text{otherwise} \end{cases}$$
4: $l_{s't'}^{i+1} = \max\{c(\overline{P}^i), l^i(\overline{P}^i)\}$
5: $y^{i+1} \leftarrow \texttt{AKR}(G^i, l^{i+1})$
6: return $y^{i+1}$
```

We first define $\mathcal{K}^i$ to be the set of connected components of

$$G\left[\bigcup_{P \in \mathcal{P}_2^i} P\right].$$

Let H^i be an auxiliary graph that has one node for each set in $\mathcal{K}^i$. Moreover, H^i contains edge (K', K'') iff there is a K', K''-Steiner path in the set $\mathcal{P}_1^i$. It can be seen that each path $P \in \mathcal{P}_1^i$ corresponds to unique edge in H^i. It then follows from the fact that $G[E^i]$ is a tree that H^i must also be a tree.

In order to compute ϵ^i that satisfies the properties mentioned in the previous section we need the following distance function for each pair $K', K'' \in \mathcal{K}^i$:

$$d^i(K', K'') = \min_{\substack{P \in \mathcal{P}_{K',K''} \\ P \cap S^i_{d^i-2} = \emptyset}} c(P). \tag{5.12}$$

Here $\mathcal{P}_{K',K''}$ again denotes the set of Steiner paths that originates at a terminal node $k' \in K'$ and ends in a terminal node $k'' \in K''$, respectively. For a pair of components $K', K'' \in \mathcal{K}^i$ we denote the path that achieves the above minimum by $P_{K',K''}$.

Definition 8 *We say that a path $\overline{P}$ is ϵ-swappable against P in iteration i if the following holds:*

1. $P \in \mathcal{P}_1^i$
2. $\overline{P} \notin \mathcal{P}^i$ *and* $\overline{P} \cap S^i_{d^i-2} = \emptyset$
3. $P \in \mathcal{P}^i(C)$ *where C is the unique cycle created in H^i by adding the edge corresponding to $\overline{P}$*
4. $c(\overline{P}) \le l^i(P) + \epsilon$

We are now looking for the smallest ϵ^i such that there exists a *witness pair* of paths $\langle P^i, \overline{P}^i\rangle$ where $\overline{P}^i$ is ϵ^i-swappable against P^i.

Formally consider all pairs $K', K'' \in \mathcal{K}^i$. Inserting the edge corresponding to $P_{K',K''}$ into H^i creates a unique cycle C. Let $\mathcal{P}^i(C) \subseteq \mathcal{P}_1^i$ be the subset of Steiner paths that correspond to edges on the cycle C. For each such path $P \in \mathcal{P}^i(C)$, let $\epsilon^i_{K',K''}(P)$ be the smallest non-negative value of ϵ such that

$$d^i(K', K'') \le l^i(P) + \epsilon. \tag{5.13}$$

We then let

$$\epsilon^i_{K',K''} = \min_{P \in \mathcal{P}^i(C)} \epsilon^i_{K',K''}(P)$$

and

$$\epsilon^i = \min_{K',K'' \in \mathcal{K}^i} \epsilon^i_{K',K''}.$$

We let $\langle P^i, \overline{P}^i \rangle$ be the pair of Steiner paths that defines ϵ^i, i.e. $\overline{P}^i$ is a K', K''-Steiner path such that

1. inserting edge (K', K'') into H^i creates a cycle C and $P^i \in \mathcal{P}^i(C)$, and
2. $c(\overline{P}^i) = l^i(P^i) + \epsilon^i$.

We are now in the position to show that (5.10) and (5.11) are maintained for our choice of $(P^i, \overline{P}^i)$ and ϵ^i. Specifically, we first show that `mod-AKR` produces a feasible dual solution (y^{i+1}, λ^{i+1}) for (D2) provided that (y^i, λ^i) was dual feasible.

Claim 6 *Algorithm 6 produces a feasible dual solution (y^{i+1}, λ^{i+1}) for (D2) given that (y^i, λ^i) is dual feasible for (D2).*

Proof: The proof is by contradiction. Assume there is a Steiner path P such that $l^{i+1}(P) > c^{\lambda^{i+1}}(P)$. First of all, notice that this Steiner path must have endpoints $s \in K'$ and $t \in K''$ for $K', K'' \in \mathcal{K}^i$ such that $K' \neq K''$. The reason for this is that

$$l^{i+1}(P) = l^i(P)$$

for all Steiner paths in $\mathcal{P}^i$ that connect terminals $s, t \in K$ for some $K \in \mathcal{K}^i$. This also implies that $l^{i+1}(P) = l^i(P)$ for all s, t-Steiner paths $P \notin \mathcal{P}^i$ such that $s, t \in K$ and $K \in \mathcal{K}^i$.

Secondly, notice that P must be different from $\overline{P}^i$ since the way we choose the length of the edge corresponding to $\overline{P}^i$ in step 4 of `mod-AKR` ensures that the dual load on this path does not exceed the maximum of the previous load and the cost of the path.

We consider two main cases:

Case 1: ($P \in \mathcal{P}^{i+1} \setminus \{\overline{P}^i\}$) In this case, step 3 of `mod-AKR` determines the dual load of P. We have

$$l^{i+1}(P) = l^i(P) + \epsilon^i.$$

Now, observe that

$$c^{\lambda^{i+1}}(P) = c^{\lambda^i}(P) + |P \cap S^i_{d^i-2}|\epsilon^i \geq c^{\lambda^i}(P) + \epsilon^i$$

which shows that we must have $l^{i+1}(P) \leq c^{\lambda^{i+1}}(P)$.

Case 2: ($P \notin \mathcal{P}^{i+1}$) In this case, we can assume that $\epsilon^i > 0$ since $l^{i+1}(P) = l^i(P)$ for all Steiner paths P otherwise.

We subdivide into two sub cases. Let's first assume that $P \cap S^i_{d^i-2} = \emptyset$. Let C be the cycle in $H^i + P$. Then, there must be some path $P' \in \mathcal{P}^i(C)$ such that

$$l^{i+1}(P) = l^{i+1}(P').$$

It follows from the choice of ϵ^i in (5.13) and the fact that $P \cap S^i_{d^i-2} = \emptyset$ that

$$l^{i+1}(P') \leq c(P).$$

The right hand side of this inequality is clearly at most $c^{\lambda^i}(P) = c^{\lambda^{i+1}}(P)$.

Assume now that $P \cap S^i_{d^i-2} \neq \emptyset$. In this case, we must have that

$$l^{i+1}(P) \leq l^i(P) + \epsilon^i.$$

The claim follows since $l^i(P) \leq c^{\lambda^i}(P)$ and

$$c^{\lambda^{i+1}}(P) \geq c^{\lambda^i}(P) + \epsilon^i$$

since P contains nodes from $S^i_{d^i-2}$.

■

This shows (5.10). It is clear from the choice of ϵ^i that we do only include a Steiner path $\overline{P}^i$ into $\mathcal{P}^{i+1}$ if $l^{i+1}(\overline{P}^i) \geq c(\overline{P}^i)$. (5.11) now follows since the dual load on any path is non-decreasing as we progress.

5.3.5 Analysis: Performance guarantee

In this section we show that the cost of the tree computed by Algorithm 5 is within a constant factor of any Steiner tree satisfying all degree bounds. We ensure this by way of weak duality. In particular, our goal is to prove the inequality

$$\sum_{P \in \mathcal{P}^i} c(P) \leq 3 \sum_{S \subset R} y^i_S - 3 \sum_{v \in V} B_v \cdot \lambda^i_v \tag{5.14}$$

for all iterations i of our algorithm.

First, we observe a consequence of Claim 3 and Claim 4.

Corollary 5 *Assume that Algorithm 5 terminates after t iterations. For iteration $0 \leq i \leq t$, let $l^i_{\max} = \max_{P \in \mathcal{P}^i} l^i(P)$. We then must have*

$$\sum_{P \in \mathcal{P}^i} l^i(P) = 2 \sum_{S \subset R} y^i_S - l^i_{\max}.$$

Proof: Let $r = |R|$ and let $\mathcal{P}^i = \{P^i_1, \ldots, P^i_{r-1}\}$ be the paths computed by `mod-AKR` in iteration $i-1$. Also let y^i be the corresponding dual solution returned by `mod-AKR`. W.l.o.g. we may assume that

$$l^i(P^i_1) \leq \ldots \leq l^i(P^i_{r-1}).$$

Using Claim 4 it is not hard to see that

$$\begin{aligned}
\sum_{S \subset R} y^i_S &= \frac{1}{2} \cdot \sum_{j=1}^{r-1} (l^i(P^i_j) - l^i(P^i_{j-1})) \cdot (r-j+1) \qquad (5.15)\\
&= \frac{1}{2} \cdot \sum_{j=1}^{r-1} l^i(P^i_j)\,((r-j+1)-(r-j)) + \frac{1}{2} l^i(P^i_{r-1})\\
&= \frac{1}{2} \cdot \sum_{j=1}^{r-1} l^i(P^i_j) + \frac{1}{2} l^i(P^i_{r-1})
\end{aligned}$$

where we define $l^i(P^i_0) = 0$. The last equality (5.15) can be restated as

$$\sum_{P \in \mathcal{P}^i} l^i(P) = 2 \sum_{S \subset R} y^i_S - l^i_{\max}$$

and that yields the correctness of the corollary. ■

We now proceed with proving (5.14) for all $1 \leq i \leq t$. Notice that Corollary 5 together with (5.11) implies (5.14) for $i = 0$. We concentrate on the case $i \geq 1$.

The proof is based on the following invariant that we maintain inductively for all $0 \leq i \leq t$:

$$3 \cdot \sum_{v \in V} B_v \lambda_v^i \leq \sum_{S \subset R} y_S^i. \tag{Inv2}$$

Since, $\lambda_v^0 = 0$ for all $v \in V$ by definition, (Inv2) holds for $i = 0$.

Growing λ_v^i by ϵ^i at nodes $v \in S^i_{d^i-2}$ decreases the right hand side of (5.14) by $\epsilon \cdot \sum_{v \in S^i_{d^i-2}} B_v$. Still the cost of the Steiner tree E^{i+1} is potentially higher than the cost of the old tree E^i. We must show that the first term on the right hand-side of (5.14), i.e. $\sum_{S \subset R} y_S^i$ grows sufficiently to compensate for the decrease in the second term and the increased Steiner tree cost. In order to show this we need the following technical lemma that lower-bounds the number of paths that contain nodes of degree at least d^i in terms of the number of nodes of normalized degree at least $d^i - 2$.

Lemma 7 *In each iteration $1 \leq i \leq t$ we must have*

$$|\mathcal{P}_1^i| \geq \alpha \cdot \sum_{v \in S^i_{d^i-2}} B_v$$

for an arbitrary parameter $\alpha > 0$ by setting

$$\beta_v \geq 2\alpha b + 1/B_v$$

for all $v \in V$ in the definition of $\texttt{ndeg}_T(v)$ in (5.8).

Proof: We first define a set of marked edges

$$M \subseteq \bigcup_{v \in S^i_{d^i}} \delta(v)$$

and then show that each Steiner path that contains nodes from $S^i_{d^i}$ has at most two marked edges. This shows that the cardinality of the set of marked edges is at most twice the number of paths in $\mathcal{P}_1^i$, i.e.

$$|M| \leq 2 \cdot |\mathcal{P}_1^i|. \tag{5.16}$$

In the second part of the proof we argue that M is sufficiently large.

First, we include all edges that are incident to terminal nodes from $S^i_{d^i}$ into M. Secondly, we also mark edges $uv \in E^i$ that are incident to non-terminal nodes in $S^i_{d^i}$ and that in addition satisfy that there is no Steiner path

$$P = \langle P_1, uv, P_2 \rangle \in \mathcal{P}^i$$

such that both P_1 and P_2 contain nodes from $S^i_{d^i}$.

It is immediately clear from this definition that each Steiner path $P \in \mathcal{P}^i$ has at most two edges from M.

We now claim that M contains at least

$$b\alpha \cdot \sum_{v \in S^i_{d^i}} B_v \tag{5.17}$$

edges. To see this, we let T be the tree on node set $S^i_{d^i}$ that is induced by E^i: For $s, t \in S^i_{d^i}$ we insert the edge st into T iff the unique s,t-path in E^i has no other nodes from $S^i_{d^i}$. We let $P_e \subseteq E^i$ be the path that corresponds to an edge $e \in E[T]$.

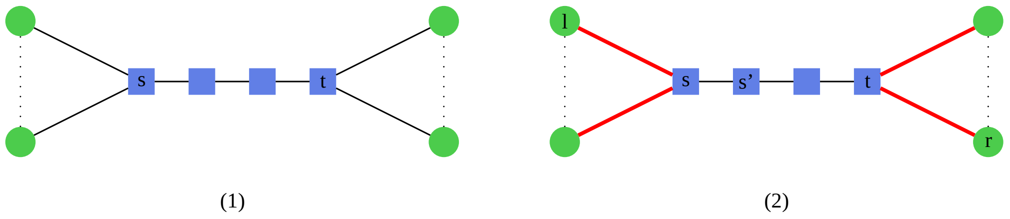

Figure 5.2: Figure (1) shows a Steiner tree where circles represent terminals and squares represent Steiner nodes. We assume that there are exactly two nodes of high normalized degree: s and t. Figure (2) shows the set M of marked edges in red. Notice that the edge between Steiner nodes s and s' is not marked since there must be a Steiner path connecting a terminal node l on the left side and a terminal node r on the right side. This Steiner path has the form $\langle P_{ls}, ss', P_{s'r} \rangle$ and P_{ls} contains node s which has high normalized degree.

In the following, we let $E^i_{d^i} \subseteq E^i$ be the set of tree edges that are incident to nodes of normalized degree at least d^i, i.e.

$$E^i_{d^i} = \bigcup_{v \in S^i_{d^i}} \delta(v).$$

Now let $U \subseteq E^i$ be the set of unmarked tree edges that are incident to nodes of normalized degree at least d^i, i.e. $U = E^i_{d^i} \setminus M$.

First observe that, by definition of M, for each unmarked edge $e \in U$ there must be an edge $e^t \in E[T]$ such that e is an edge on the path P_{e^t}. Moreover, for all $e_t \in E[T]$ there are at most two unmarked edges on the path P_{e^t}. Since T has $|S^i_{d^i}| - 1$ edges we obtain

$$|U| \leq 2 \cdot (|S^i_{d^i}| - 1). \tag{5.18}$$

Each node in $S^i_{d^i}$ has at least $\beta_v B_v + d^i$ edges incident to it. On the other hand, since E^i is a tree, at most $(|S^i_{d^i}| - 1)$ of the edges in $E^i_{d^i}$ are incident to exactly two nodes from $S^i_{d^i}$. Hence, we obtain

$$\begin{aligned} |E^i_{d^i}| &\geq \left(\sum_{v \in S^i_{d^i}} \beta_v B_v + d^i \right) - (|S^i_{d^i}| - 1) \qquad (5.19) \\ &= \left(2\alpha b \cdot \sum_{v \in S^i_{d^i}} B_v \right) + d^i \cdot |S^i_{d^i}| + 1 \end{aligned}$$

where the last equality uses the definition of β_v.

Now observe that $|M| = |E^i_{d^i}| - |U|$ and hence

$$|M| \geq \left(2\alpha b \cdot \sum_{v \in S^i_{d^i}} B_v \right) + |S^i_{d^i}|(d^i - 2) - 1. \tag{5.20}$$

using (5.18) and (5.19). Notice that $d^i \geq \Delta^i - \lceil 4 \log_b n \rceil + 2$ and $\Delta^i > \lceil 4 \log_b n \rceil$ and hence $d^i \geq 3$. This together with (5.20) and the fact that $S^i_{d^i}$ is non-empty implies

$$|M| \geq 2\alpha b \cdot \sum_{v \in S^i_{d^i}} B_v. \tag{5.21}$$

Finally we combine (5.16) and (5.21) and obtain

$$|\mathcal{P}^i_1| \geq \alpha b \cdot \sum_{v \in S^i_{d^i}} B_v.$$

Using the fact that $\sum_{v \in S^i_{d^i-2}} B_v \leq b \cdot \sum_{v \in S^i_{d^i}} B_v$ finishes the proof of the lemma. ■

The following claim now presents the essential insight that ultimately yields the validity of (5.14).

Claim 7 *Let α be as in Lemma 7. We then must have*

$$\sum_{S \subset R} y^{i+1}_S \geq \sum_{S \subset R} y^i_S + \frac{\alpha}{2} \epsilon^i \cdot \sum_{v \in S^i_{d^i-2}} B_v$$

for all $0 \leq i \leq t$.

Proof: We can use (5.15) to quantify the change in dual in iteration i.

$$\begin{aligned}
\sum_{S \subset R} (y^{i+1}_S - y^i_S) &= \frac{1}{2} \cdot \sum_{j=1}^{r-1} (l^{i+1}(P^i_j) - l^i(P^i_j)) + \frac{1}{2}(l^{i+1}(P^i_{r-1}) - l^{i+1}(P^i_{r-1})) \\
&\geq \frac{\epsilon^i}{2} \cdot |\mathcal{P}^i_1|
\end{aligned}$$

where the inequality follows from the fact that we increase the length of all paths in $\mathcal{P}^i_1$ by ϵ^i and the length of all other paths are non-decreasing as we progress.

We can now use Lemma 7 and conclude that

$$\sum_{S \subset R} y^{i+1}_S \geq \sum_{S \subset R} y^i_S + \frac{\alpha}{2} \epsilon^i \cdot \sum_{v \in S^i_{d^i}} B_v.$$

This finishes the proof of the claim. ■

As `mod-AKR` finishes with cut metric l^{i+1}, we obtain

$$l^{i+1}(\mathcal{P}^{i+1}) = \sum_{P \in \mathcal{P}^{i+1}} l^{i+1}(P) \leq 2 \sum_{S \subset R} y^{i+1}_S \tag{5.22}$$

from Claim 3. Observe that the real cost of the Steiner tree E^{i+1} is much smaller than $l^{i+1}(\mathcal{P}^{i+1})$. In fact, notice that we have

$$\begin{aligned}
c(\mathcal{P}^{i+1}) &\leq l^{i+1}(\overline{P}^i) + c(\mathcal{P}^i \setminus \{P^i\}) \\
&\leq l^{i+1}(\overline{P}^i) + l^i(\mathcal{P}^i \setminus \{P^i\})
\end{aligned} \tag{5.23}$$

where the inequalities follow from (5.11), i.e. the l-cost of a Steiner path in $\mathcal{P}^i$ always dominates its c-cost. Also, observe that

$$l^{i+1}(\mathcal{P}^i \setminus \{P^i\}) = l^i(\mathcal{P}^i \setminus \{P^i\}) + \epsilon^i \cdot |\mathcal{P}^1_i| \geq l^i(\mathcal{P}^i \setminus \{P^i\}) + \alpha \epsilon^i \cdot \sum_{v \in S^i_{d^i-2}} B_v \tag{5.24}$$

using Lemma 7.

Combining (5.22), (5.23) and (5.24) yields

$$c(\mathcal{P}^{i+1}) \le l^{i+1}(\mathcal{P}^{i+1}) - \alpha\epsilon^i \cdot \sum_{v \in S^i_{d^i-2}} B_v \le 2 \cdot \sum_{S \subset R} y_S^{i+1} - \alpha\epsilon^i \cdot \sum_{v \in S^i_{d^i-2}} B_v.$$

We can now add (Inv2) to the last inequality and get

$$c(\mathcal{P}^{i+1}) \le 3 \sum_{S \subset R} y_S^{i+1} - 3 \cdot \sum_{v \in V} B_v \lambda_v^i - \alpha\epsilon^i \cdot \sum_{v \in S^i_{d^i-2}} B_v.$$

Finally notice that $\lambda_v^{i+1} = \lambda_v^i + \epsilon^i$ if $v \in S^i_{d^i-2}$ and $\lambda_v^{i+1} = \lambda_v^i$ otherwise. Now choose $\alpha \ge 3$ and it follows that

$$c(\mathcal{P}^{i+1}) \le 3 \sum_{S \subset R} y_S^{i+1} - 3 \cdot \sum_{v \in V} B_v \lambda_v^{i+1}.$$

We have to show that (Inv2) is maintained as well. Observe that the left hand side of (Inv2) increases by $3\epsilon^i \cdot \sum_{v \in S^i_{d^i-2}} B_v$. We obtain from Claim 7 that

$$\sum_{S \subset R} y_S^{i+1} - y_S^i \quad \ge \quad \frac{\alpha}{2} \cdot \epsilon^i \cdot \sum_{v \in S^i_{d^i-2}} B_v.$$

Choosing $\alpha \ge 6$ shows that the right hand side of (Inv2) increases sufficiently and (Inv2) holds in iteration $i+1$ as well.

5.3.6 Analysis: Running time

In this section, we show that Algorithm 5 terminates within a quasi-polynomial number of steps. We accomplish this by showing that there will be only a quasi-polynomial number of iterations of the main loop in Algorithm 5.

Claim 8 *Algorithm 5 terminates after $O(n \log(|R|) \cdot |R|^{\lceil 4 \log n \rceil})$ iterations.*

Proof: For a Steiner tree $\mathcal{P}$ in path representation, we define its potential value as

$$\Phi(\mathcal{P}) = \sum_{P \in \mathcal{P}} |R|^{\max_{v \in P} \mathtt{ndeg}_{\mathcal{P}}(v)}$$

where $\mathtt{ndeg}_{\mathcal{P}}(v)$ is the normalized degree of node v in the Steiner tree defined by $\mathcal{P}$.

Now notice that an iteration of Algorithm 5 swaps out a path $P^i \in \mathcal{P}_1^i$ that has at least one node of normalized degree at least d^i. Hence, the maximum normalized degree of any node on P^i must be at least d^i. The path $\overline{P}^i$ on the other hand does not touch any node of normalized degree more than $d^i - 3$.

This means that a swap may potentially raise the maximum normalized node-degree of at most $|R|-1$ Steiner paths to $d^i - 1$. The reduction in the potential is hence at least

$$(|R|^{d^i} - (|R|-1)|R|^{d^i-1}) \ge |R|^{d^i-1}$$

Using the fact that $d^i \ge \Delta^i - \lceil 4 \log_b n \rceil + 2$, we can lower-bound the right hand side of the last inequality by

$$|R|^{\Delta^i - \lceil 4 \log_b n \rceil + 1} = \Omega\left(\frac{|R|^{\Delta^i+1}}{|R|^{\lceil 4 \log n \rceil}}\right).$$

The initial potential Φ^i is at most

$$|R| \cdot |R|^{\Delta^i}$$

and the decrease in the potential is at least

$$\Omega\left(\frac{\Phi^i}{|R|^{\lceil 4\log n\rceil}}\right).$$

In other words, in $O(|R|^{\lceil 4\log n\rceil})$ iterations, we reduce Φ by a constant factor. Hence, the algorithm needs

$$O(|R|^{\lceil 4\log n\rceil} \cdot \ln \Phi^0) = O(n\log(|R|) \cdot |R|^{\lceil 4\log n\rceil})$$

iterations total. ■

Conclusions and Open Issues

In this thesis we make a first step towards developing general techniques to approximate fundamental network design problems in the presence of side constraints. More precisely, we first study the minimum-cost spanning tree problem for which efficient algorithms are known but once we add constraints limiting the degree of the nodes in the graph, the problem becomes NP-hard.

The first part of this thesis shows how we can use Lagrangean relaxation techniques in order to compute an approximate minimum-cost degree-bounded spanning tree. For an integer B and an n-node graph G, our method computes a spanning tree T of G whose cost is at most a constant factor worse than that of a minimum-cost degree-B-bounded spanning tree. The maximum node-degree of any node in T is $O(B + \log n)$.

The result improves previous work by Ravi et al. [58]. Moreover, to the best of our knowledge, we are the first to analyze a Lagrangean relaxation-based approximation algorithm that simultaneously dualizes a large number of constraints. We believe that the technique is of independent interest and that it can be generalized to a wider class of multicriteria optimization problems.

The second part of this thesis develops an iterated primal-dual approach to tackle the degree-bounded spanning and Steiner tree problems. Our algorithm achieves essentially the same performance guarantees as our Lagrangean relaxation based method but generalizes the previous result in that it can be extended to the setting of individual degree bounds. The technique also leads to a quasi-polynomial-time approximation algorithm for the minimum-cost degree-bounded Steiner tree problem.

In contrast to the first Lagrangean method, this algorithm is *direct* in the sense that it does not use linear programming. The analysis shows that, at any point during the execution of our algorithm, we have a spanning tree (Steiner tree) whose cost is at most a constant factor worse than that of any spanning tree (Steiner tree) that obeys all degree bounds. The maximum degree of this tree is reduced as the algorithm progresses. Hence, a practical benefit of this method is that we can stop the algorithm at any point and obtain a low-cost tree whose maximum node-degree may be higher than our theoretical bounds, however.

The iterated primal-dual approach is interesting in its own right. We are the first to analyze a method that repeatedly runs a primal-dual algorithm and – in between two such runs – refines parts of the dual solution. Many fundamental network-design problems possess primal-dual algorithms (e.g. see Chapter 4 of [33]). The hope is that our new *iterative* primal-dual approach gives rise to new algorithms for these problems in the presence of additional side constraints.

6.1 OPEN ISSUES

While our work is a first step towards a more complete body of techniques to approximate constrained network design problems, many open issues remain. In this section we list some of the questions that we encountered in the process of writing this thesis and that still need to be resolved.

- We believe that the Lagrangean techniques from Chapter 3 can be generalized to apply to a broader class of multicriteria problems. A central point in the development of a more general framework is the identification of key properties of suitable optimization problems; in the BMST problem, the dualization of the degree constraints yields a tractable subproblem. Furthermore, the compact form of the objective function of this subproblem proved to be a key for the analysis.

- A fundamental fact used in the analysis of the Lagrangean algorithm in Chapter 3 is that the subproblem resulting from dualizing degree constraints needs to be in P. This requirement prohibits us from using this approach to obtain an approximation algorithm for the minimum-cost degree-bounded Steiner tree problem. Can we extend the techniques from Chapter 3 to apply also when the Lagrangean subproblems are NP-hard?

- In Chapter 5 we prove that our approximation algorithm for the minimum-degree Steiner tree problem needs a quasi-polynomial number of iterations to terminate. We conjecture that the algorithm has polynomial running time and that our analysis is not exhibiting the *right* potential function to prove this fact.

- We believe that the iterated primal-dual framework leads to improved approximation algorithms for minimum-cost in-degree-bounded arborescences. The two main reasons for our belief are the existence of a primal-dual algorithm for minimum-cost arborescences [16] and a recent local-search method due to Krishnan and Raghavachari [46] for the minimum in-degree arborescence problem.

- In [20], Fischer shows how to adapt the local search method from [22] to compute minimum-cost spanning trees in a given weighted undirected graph such that the maximum degree is at $O(\texttt{opt} + \log n)$ where `opt` denotes the minimum possible maximum degree of any minimum-cost spanning tree. In [22], Fürer and Raghavachari also present a more complex algorithm that computes a spanning tree of the input graph G with maximum degree at most $\texttt{opt} + 1$. Is there an algorithm that computes a minimum-cost spanning tree with degree $\texttt{opt} + 1$?

- This question is along the same lines as the previous one: We think that the additive $\log(n)$ that appears in all of the degree-bounds that our algorithms achieve is an artefact of the way we select witness sets. Is there a better way of finding witness sets that does not require low expansion properties? A more difficult question is: Can we compute a spanning tree (Steiner tree) whose cost is at most a constant factor times that of a spanning tree (Steiner tree) that obeys all degree constraints and in addition has maximum degree at most $B + 1$ for a given degree target B?

- Minimum-degree spanning trees in planar graphs. Can we improve the results of our work in the special case of planar graphs?

- Can we use an iterated primal-dual approach to deal with other constrained network design questions?

Bibliography

1. A. Agrawal, P. Klein, and R. Ravi. How tough is the minimum-degree steiner tree? A new approximate min-max equality. Technical Report CS-91-49, Brown University, 1991.

2. A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 134–144, 1991.

3. A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem in networks. *SIAM J. Comput.*, 24:440–456, 1995.

4. D. Agrawal, A. El Abbadi, and R. C. Steinke. Epidemic algorithms in replicated databases. In *Proceedings, ACM Symposium on Principles of Database Systems*, pages 161–172, 1997.

5. F. Bauer and A. Varma. Degree-constrained multicasting in point-to-point networks. In *Proceedings, IEEE INFOCOM*, pages 369–376, 1995.

6. J. Bolot, T. Turletti, and I. Wakeman. Scalable feedback control for multicast video distribution in the internet. In *Proceedings of SIGCOMM*, pages 58–67, 1994.

7. C. Brezovec, G. Cornuéjols, and F. Glover. A matroid algorithm and its application to the efficient solution of two optimization problems on graphs. *Math. Programming*, 42, 1988.

8. S. Chopra. On the spanning tree polyhedron. *Operations Research Letters*, 8:25–29, 1989.

9. N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.

10. Y. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proceedings of SIGCOMM*, pages 55–68, 2001.

11. V. Chvátal. Tough graphs and Hamiltonian circuits. *Discrete Math.*, 5:215–228, 1973.

12. S. Deering, D. Estrin, and D. Farinacci. An architecture for wide-area multicast routing. In *Proceedings of SIGCOMM*, 1994.

13. S. E. Deering and D. R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, 8(2):85, May 1990.

14. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings, ACM Symposium on Principles of Distributed Computing*, pages 1–12, 1987.

15. Reinhard Diestel. *Graph theory.* Springer-Verlag, New York, 2nd edition, 2000.

16. J. Edmonds. Optimum branchings. *J. Res. Nat. Bur. Standards*, B71:233–240, 1967.

17. J. Edmonds. Submodular functions, matroids, and certain polyhedra. In R. Guy, H. Hanam, and J. Schonheim, editors, *Combinatorial structures and their applications*, pages 69–85, New York, 1970. Gordon and Breach.

18. S. P. Fekete, S. Khuller, M. Klemmstein, B. Raghavachari, and N. Young. A network-flow technique for finding low-weight bounded-degree spanning trees. *J. Algorithms*, 24(2):310–324, 1997.

19. C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *In Proceedings, ACM IEEE Nineteenth Design Automation Conference*, pages 175–181, June 1982.

20. T. Fischer. Optimizing the degree of minimum weight spanning trees. Technical Report TR 93-1338, Dept. of Computer Science, Cornell University, Ithaca, NY 14853, 1993.

21. M. Fürer and B. Raghavachari. An NC approximation algorithm for the minimum degree spanning tree problem. In *Proc. of the 28th Annual Allerton Conf. on Communication, Control and Computing*, pages 274–281, 1990.

22. M. Fürer and B. Raghavachari. Approximating the minimum-degree Steiner tree to within one of optimal. *Journal of Algorithms*, 17(3):409–423, November 1994.

23. H. N. Gabow. A good algorithm for smallest spanning trees with a degree constraint. *Networks*, 8:201–208, 1978.

24. H. N. Gabow and R. E. Tarjan. Efficient algorithms for a family of matroid intersection problems. *J. Algorithms*, 5(1):80–131, March 1984.

25. M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness.* W. H. Freeman and Company, San Francisco, 1979.

26. B. Gavish. Topological design of centralized computer networks- formulations and algorithms. *Networks*, 12:355–377, 1982.

27. F. Glover and D. Klingman. *B.Roy (ed.): Combinatorial Programming: Methods and Applications*, chapter pp. 191–201. D. Reidel Publishing Company, Dodrecht, Holland, 1975.

28. M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '92)*, pages 307–316, 1992.

29. M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24:296–317, 1995.

30. M. X. Goemans and D. P. Williamson. *The primal-dual method for approximation algorithms and its applications to network design problems. In D.S. Hochbaum, editor, Approximation Algorithms for NP-hard Problems*, pages 144–191. PWS publishing, Boston, 1997.

31. M. Held and R. M. Karp. The traveling-salesman and minimum cost spanning trees. *Operations Research*, 18:1138–1162, 1970.

32. B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. In Sidney Karin, editor, *Proceedings of the 1995 ACM/IEEE Supercomputing Conference, December 3–8, 1995, San Diego Convention Center, San Diego, CA, USA*, 1995.

33. D. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company, 1997.

34. F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem.* Number 53 in Annals of Discrete Mathematics. Elsevier Science Publishers B. V., 1992.

35. A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

36. D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing; part I, graph partitioning. *Operations Research*, 37:865–892, 1989.

37. D. S. Johnson, M. Minkoff, and S. Phillips. The prize collecting steiner tree problem: theory and practice. In *Proceedings, ACM-SIAM Symposium on Discrete Algorithms*, pages 760–769, 2000.

38. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

39. B. W. Kernighan and S. Lin. An efficient heuristic for partitioning graphs. *Bell Systems Technical J.*, 49:291–307, 1970.

40. S. Khuller, B. Raghavachari, and N. Young. Low degree spanning trees of small weight. In *Proceedings, ACM Symposium on Theory of Computing*, pages 412–421, 23–25 1994.

41. S. Khuller, B. Raghavachari, and N. Young. Low-degree spanning trees of small weight. *SIAM Journal on Computing*, 25(2):355–368, April 1996.

42. P. N. Klein and R. Ravi. A nearly best-possible approximation for node-weighted steiner trees. *J. Algorithms*, 19:104–115, 1995.

43. J. Könemann and R. Ravi. A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. In *Proceedings, ACM Symposium on Theory of Computing*, pages 537–546, 2000.

44. J. Könemann and R. Ravi. A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. *SIAM J. Comput.*, 31(6):1783–1793, 2002.

45. J. Könemann and R. Ravi. Primal-dual algorithms come of age: Approximating MST's with nonuniform degree bounds. To appear in proceedings, ACM Symposium on Theory of Computing, 2003.

46. R. Krishnan and B. Raghavachari. The directed minimum-degree spanning tree problem. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 21, 2001.

47. J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings, American Mathematical Society*, 7:48–50, 1956.

48. E. L. Lawler, J. K. Lenstra, A. Rinnooy-Kan, and D. B. Shmoys. *The Travelling Salesman Problem.* Wiley, New York, 1985.

49. Eugene Lawler. *Combinatorial Optimization: Networks and Matroids.* Saunders College Publishing, Fort Worth, 1976.

50. J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations Res.*, 11:972–989, 1963.

51. Madhav V. Marathe, R. Ravi, Ravi Sundaram, S. S. Ravi, Daniel J. Rosenkrantz, and Harry B. Hunt III. Bicriteria network design problems. *Journal of Algorithms*, 28(1):142–171, July 1998.

52. C. Monma and S. Suri. Transitions in geometric minimum spanning trees. In *Proceedings, ACM Symposium on Computational Geometry*, pages 239–249, 1991.

53. C. Monma and S. Suri. Transitions in geometric minimum spanning trees. *Discrete & Computational Geometry*, 8(3):265–293, 1992.

54. S.C. Narula and C.A. Ho. Degree-constrained minimum spanning tree. *Comput. Ops. Res.*, 7:239–249, 1980.

55. G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, 1988.

56. C. H. Papadimitriou. *Computatational Complexity.* Addison-Wesley, Reading, Mass., 1994.

57. C. H. Papadimitriou and U. V. Vazirani. On two geometric problems related to the travelling salesman problem. *J. Algorithms*, 5(2):231–246, June 1984.

58. R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt. Many birds with one stone: Multi-objective approximation algorithms. In *Proceedings, ACM Symposium on Theory of Computing*, pages 438–447, 1993.

59. R. Ravi, B. Raghavachari, and P. Klein. Approximation through local optimality: Designing networks with small degree. In *Proceedings, Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 652 of *LNCS*, pages 279–290. Springer, 1992.

60. D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977.

61. M. Savelsbergh and A. Volgenant. Edge exchanges in the degree-constrained minimum spanning tree problem. *Comput. Ops. Res.*, 12:341–348, 1985.

62. E. Tardos. Personal communication. April 2000.

63. V. V. Vazirani. *Approximation Algorithms.* Springer, 2001.

64. A. Volgenant. A Lagrangean approach to the degree-constrained minimum spanning tree problem. *Europ. J. Ops. Res.*, 39:325–331, 1989.

65. C. Walshaw and M. Cross. Mesh partitioning: A multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing*, 22(1):63–80, January 2000.

66. D. B. West. *Introduction to Graph Theory.* Prenctice Hall, Upper Saddle River, 2nd edition, 2001.

67. D. P. Williamson and M. X. Goemans. Computational experience with an approximation algorithm on large-scale euclidean matching instances. In *Proceedings, ACM-SIAM Symposium on Discrete Algorithms*, pages 355–364, 1994.

68. D. P. Williamson and M. X. Goemans. Computational experience with anapproximation algorithm on large-scale Euclidean matching instances. *INFORMS Journal on Computing*, 8:29–40, 1996.

69. S. Win. On a connection between the existence of k-trees and the toughness of a graph. *Graphs and Combinatorics*, 5:201–205, 1989.