Optimization and Reconstruction over Graphs

Samantha J. Riesenfeld

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-6
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-6.html

January 14, 2008

Acknowledgement

Advisor: Richard M. Karp

Optimization and Reconstruction over Graphs

by

Samantha J. Riesenfeld

B.A. (Harvard University) 2000

A dissertation submitted in partial satisfaction of the
requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Richard M. Karp, chair
Professor Satish Rao
Professor Alistair Sinclair
Professor Dorit Hochbaum

Fall 2007

The dissertation of Samantha J. Riesenfeld is approved:

Chair _____ Date _____

_____ Date _____

_____ Date _____

_____ Date _____

University of California, Berkeley

Fall 2007

Optimization and Reconstruction over Graphs

Samantha J. Riesenfeld

Abstract

Optimization and Reconstruction over Graphs

by

Samantha J. Riesenfeld

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Richard M. Karp, Chair

We study several instances of the following combinatorial optimization problem: Efficiently find a graph that satisfies, to the extent possible, a given set of constraints.

The thesis begins with two NP-hard problems: The Minimum-Degree Minimum Spanning Tree (MDMST) problem is to find, given a graph, an MST of minimum degree. The Minimum Bounded-Degree Spanning Tree (MBDST) problem is to find, given a graph and an integer B, a minimum-cost tree in the set of spanning trees of degree at most B.

We present the first polynomial-time constant-factor approximation algorithm for the MDMST problem, which uses the push-relabel framework developed by Goldberg [20] for the max-flow problem. It improves Fischer's local-search algorithm [14]. Via an analysis by Könemann and Ravi [34], our algorithm implies the first polynomial-time constant-factor bi-criteria approximation algorithm for the MBDST problem. It also works for a new generalization of the MDMST problem.

Other results include the first true MBDST approximation algorithms: a polynomial-time algorithm incurring no error in cost, and a quasi-polynomial-time algorithm, based on augmenting paths, that significantly improves the error in degree by finding a spanning tree of optimal cost and degree $B+O(\frac{\log n}{\log \log n})$. Our cost-bounding method requires finding MSTs that meet both upper and lower degree bounds.

The second part of the thesis considers the problem of reconstructing a directed graph, given the vertices and an oracle for the reachability relation. We show this reduces to the problem of *sorting* a partially ordered set (poset).

Sorting algorithms obtain information about a poset by queries that compare two elements. We give an algorithm that sorts a width-w poset of size n and has query complexity $O(wn + n \log n)$, meeting the information-theoretic lower bound. We describe a variant of Mergesort that has query complexity $O(wnlogn)$, matching the upper bound shown by Faigle and Turán [13], and total complexity $O(w^2 n \log n)$. The exact total complexity of sorting remains unresolved.

We also give upper and lower bounds for several related problems, including finding the minimal elements in a poset, which we show has query and total complexity $\Theta(wn)$, and its generalization, k-selection.

Prof. Richard M. Karp

Dissertation Committee Chair

To Professors E. Cohen and R. F. Riesenfeld,

and to Rebecca Irene

Contents

List of Figures

Acknowledgements

I have been fortunate enough to become indebted to many people while I've been in graduate school. I'm glad for the opportunity to "publicly" thank them.

Thanks, first, to my advisor, Dick Karp, for his support. Dick finds a simple, fundamental question—and answers it elegantly—where many people would only see a complex, applied problem, rife with practical details. Working with him has been educational and inspiring. I hope I have learned something about how to find interesting research problems. From his methodical style, I have certainly learned to think and write with more care.

Satish has often served as my de facto co-advisor, and I am grateful to him for his guidance and dedication to our projects. Collaborating with him gave me an appreciation for the value of a strongly intuitive understanding of mathematics. Thanks to him also for his friendship and for mentoring me in the coaching of girls' soccer.

Thanks to the two yet unmentioned members of my dissertation and quals committees, Alistair and Dorit, for their support and forbearance. Thanks to the theory group at Berkeley for giving me exposure to such an intellectually stimulating environment. Thanks in particular to Christos who did much to encourage the relaxed, participatory nature of the group.

I would especially like to thank my co-authors: Dick, Satish, Elchanan, Kunal, Kamalika, Costis, Elad Verbin, Elitza, and Andrej. Special thanks to Kunal, who has helped me so much through his patience, generosity, and high standards. Thanks to Elchanan for his collaborative efforts, patience, and friendship.

Thanks to Tandy Warnow, Brent Mishler, and also Ruth Timme, for involving me in CIPRES and introducing me to computational biology. Thanks to Aaron Archer for his persistence and good advice. Thanks to Olivier Faugeras for giving me my

first real opportunity to do research. Thanks to Jim Carlson for giving me my first real opportunity to do math. Thanks to Michael Rabin for advising my senior thesis.

Thanks to NSF for the Graduate Fellowship that funded three of my years.

I owe much of any happiness in the last few years to the students who make Berkeley such a special place. Thanks to Andrej for his steadfast support and excellent companionship. Thanks to Lorenzo for helping me graduate and for teaching me, among many things, to juggle a soccer ball—sei grande. Thanks to Costis, Kunal, Kamalika, and Eli, who are not only esteemed co-authors but wonderful friends. Thanks to James for his friendship; without him, I would be a very different person.

Thanks to the old gang: Kris, Dror, Jordan, Ashwin, Jhala, Chris, Steve, Kevin, Lawrence, Jittat, and Eran, for welcoming me and making me feel at home. Thanks to folks from (roughly) my generation: Hoeteck, Mani, Ben, Alex F., Henry, Naveen, Dan, Keshav, and Barbara E., for the good company and support. Thanks to Jake and Sasha for invariably cheering me. And thanks to some of the next generation: Alexandre, Madhur, Grant, Lorenzo, Meromit, Yaron, and Guy B., for bringing a fresh spirit into my last year. Thanks especially to Grant for organizing the theory graduate student ski retreat last year.

There are a few others, without whom I would probably never have made it to this point: Sue Eisenberg, Lynn Alper, and especially Sandra Backovich.

Thanks to Margaret, rekindler of courage, for sharing in her observant, good-humored way many of the major and minor experiences of the past twelve years. Thanks to the one I consider my big sister, Ilana, and to Alon. Thanks to Brent for finding me a place to live and for nearly a lifetime of friendship. Thanks to Thomas and Sego, the best neighbors ever. Thanks to the "Utah" women: Laura, Rebecca, Sarah, Carolyn, Shea, Erin, Renee, and associates, for continuing to be inspiring people and for years of unwavering faith in me.

Thanks to Celine for taking my head out of the clouds. Thanks to the Adelante

soccer team and its great coach, Craig. Thanks to Etienne for his encouragement and company this last year. Thanks to Maggie for reminding me of the "all-american therapy of the highway" and other rejuvenating parts of the world. Thanks to Karen, Daphne, Becca, Bassina, Josh, Alex, Jacques Bride, and Jessica Maia for their enriching friendships. Thanks to Paul for helping me get to Berkeley.

I admit that I came to Berkeley at least as much for the place as for the institution, and it rewarded me well with a life that could not have been approximated anywhere else, even for the enormous sum I paid. Thanks to Brewed Awakening, Nefeli, The Beanery, Caffe Strada, Cafe Milano, The Musical Offering, Yali's, Village Grounds, Fertile Grounds, Berkeley Espresso, Nabolom, Espresso Roma, Mudrakers, A Cuppa Tea, Royal, Spasso, and even The Nomad, Gaylord's, and World Grounds, for pleasant working environments. Thanks to all the shops on Euclid and in Elmwood. Thanks to Berkeley Bowl, Acme, the Farmers' Market, and Monterey Market. The restaurants are too many, but should include Gordo's, Cheeseboard, Gregoire's, Cesar, and Gioia.

Thanks to my extended family: Rozzie, Allison, Seth, Jane, Bill, Debbie, Margaret, David, Betsy, and all the rest, for their support and understanding. Thanks for the sustaining memory of my grandmothers.

Thanks to those I regret to have forgotten in the haze of the last few days.

Finally, I have come to my immediate family. There is no way to express my gratitude to my parents and my sister for their love and support. They are the people I trust and admire most in the world. Their unreserved confidence in me, misguided though it may be, is reassuring.

Chapter 1

Introduction

This thesis is an examination, using a set of diverse but fundamental tools, of several instances of the following combinatorial optimization problem: *Efficiently* find a graph that satisfies, to the extent possible, a given set of constraints.

We study natural variants of well-known problems, such as finding a minimum spanning tree and sorting a linear order, that have received a thorough treatment by classical theory. Yet the questions we consider here remained unresolved for years. This thesis presents our contributions towards their resolution.

1.1 Bounded-degree spanning trees

A classic optimization problem in graph theory is to find, given a weighted graph $G = (V, E, c)$, where c is a cost function on the edges, a minimum-cost set of edges that connects all of the vertices, known as a minimum spanning tree (MST). Due partly to the broad applicability of MSTs and their special structure, the problem is by now well understood, and several polynomial-time algorithms for solving it are known [25, 44, 50].

In applications that involve MSTs, however, it may be desirable to enforce addi-

tional constraints on the degrees of vertices [45]. For example, for efficient, robust network routing, a multicast network would ideally minimize both the total cost of the network and the maximum work done by any router. In graph-theoretic terms, this translates to minimizing both the cost of the spanning tree and its degree, i..e the maximum degree of any vertex in the tree. Of course, there may not exist an MST whose degree is minimum with respect to all spanning trees.

The Minimum-Degree Minimum Spanning Tree (MDMST) problem is one way to formulate these competing constraints which guarantees that a solution exists. Given a graph G, the MDMST problem is to find an MST whose degree is minimized over all MSTs of G. Even in the unweighted case, it generalizes the Hamiltonian Path problem and is therefore NP-hard.

A more general bi-criteria formulation is the Minimum Bounded-Degree Spanning Tree (MBDST) problem, which requires, as additional input, an upper bound B on the degrees of the vertices. An MBDST solution is a minimum-cost tree among the spanning trees, if any, that respect the degree bounds. The MBDST problem generalizes the Traveling Salesman Path problem (TSPP), which corresponds to the case when the degrees are restricted uniformly to 2. (Note that we do not assume that the cost function respects the triangle inequality.) Since TSPP is NP-hard to approximate within any polynomial factor, approximations for the MBDST problem must relax the degree constraints, unless P equals NP.

The study of the MDMST and MBDST problems has encompassed several fundamental and aesthetic algorithmic techniques, including local search [14], alternating or augmenting paths [16, 5] (as in matching, e.g. [10]), graph decompositions proving structural properties [16, 5, 6], linear programming duality [34, 5], the primal-dual method [35], the push-relabel framework [6] (from max flow [20]), and most recently, polyhedral techniques [19, 48].

In the next section, we briefly summarize related work. A more illustrative and

complete review of previous techniques and related work is provided in Chapter 2. To facilitate the presentation of results, we introduce the following terminology: Given a weighted graph $G = (V, E, c)$, let n be the number of vertices, let $\Delta_{\text{opt}}(G)$ be the degree of an optimal MDMST solution, and let $c_{\text{opt}_B}(G)$ be the cost of an optimal MBDST solution for degree bound B.

Algorithms for the MBDST problem are either what we call *bi-criteria* approximation algorithms, which give guaranteed approximations of both the optimal cost $c_{\text{opt}_B}(G)$ and the degree B, or true approximation algorithms, which obtain optimal cost $c_{\text{opt}_B}(G)$ and a guaranteed approximation of the degree B.

1.1.1 A brief history

Goemans conjectured[1] more than 15 years ago that the MBDST problem could be solved approximately in polynomial time to get a tree of cost exactly $c_{\text{opt}_B}(G)$ and degree at most $B+1$, which is the optimal result if P does not equal NP. Shortly thereafter, Fürer and Raghavachari resolved the problem in the simpler case of unweighted graphs, in which the MDMST and MBDST problems are essentially equivalent. They gave a polynomial-time algorithm, reminiscent of Edmonds' algorithm for unweighted matching [10], that finds a spanning tree of degree $\Delta_{\text{opt}}(G) + 1$.

Initial efforts at solving the MBDST problem focused on finding bi-criteria approximations. Ravi et al [45, 46] described the first polynomial-time bi-criteria approximation algorithm for the MBDST problem, which achieves approximation factors of $O(\log n)$ for both degree and cost.

Prior to the initial publication [5, 6] of the results in this thesis, the best results for the MDMST and MBDST problems were by Fischer [14], and by Könemann and Ravi [34, 35], respectively. Fischer [14] gave an MDMST algorithm based on local search that, for any constant $b > 1$, finds a spanning tree of degree at most

[1]The conjecture is unpublished but is referenced in recent work [19].

$b\Delta_{\text{opt}}(G) + \log_b n$. Using a Lagrangian relaxation of the MDBST dual linear program, Könemann and Ravi showed that an MDMST algorithm can be used as a black box in a bi-criteria approximation algorithm for the MBDST problem. They rely on Ficher's MDMST algorithm to obtain a MBDST algorithm that, for any constants $b > 1$, $\beta > 0$, finds a spanning tree of cost at most $(1 + \frac{1}{\beta})c_{\text{opt}_B}(G)$ and degree $bB(1+\beta) + \log_b n$. While this expression represents a continuum in the cost-degree trade-off, one can observe that the algorithm always either violates the degree constraints or exceeds the optimal cost by a factor of two.

1.1.2 Main results

In Chapters 3 and 4, we present several algorithms for the MDMST and MBDST problems, including: the first polynomial-time constant-factor approximation algorithm for the MDMST problem; the first polynomial-time MBDST algorithm that approximates both degree and cost to within a constant factor; the first polynomial-time true approximation algorithm for the MBDST problem, which removes the error in cost completely; and a quasi-polynomial-time MBDST approximation algorithm that significantly improves the error in degree bounds (and still obtains optimal cost).

Specifically, our major result in Chapter 3 is a polynomial-time MDMST approximation algorithm that finds an MST of degree at most $2\Delta_{\text{opt}}(G) + O(\sqrt{\Delta_{\text{opt}}(G)})$. This algorithm uses the push-relabel framework invented by Goldberg [20] for the max flow problem (and fully developed by Goldberg and Tarjan [22]). Our adaptation of this framework allows the MDMST algorithm to explore a more general, interdependent set of moves than is available to a simple local search. To our knowledge, this work is the first use of the push-relabel technique in an approximation algorithm for an NP-hard problem.

With the cost-bounding techniques of Könemann and Ravi [34], the push-relabel

MDMST algorithm implies a polynomial-time algorithm for the MBDST problem that, for any constant $\beta > 0$, finds a spanning tree of cost at most $(1 + \frac{1}{\beta})c_{\text{opt}_B}(G)$ and degree $2B(1+\beta)+O(\sqrt{B(1+\beta)})$, thus giving the first bi-criteria constant-factor approximation.

Goemans [19] recently introduced an MDMST algorithm that finds an MST of optimal cost $\Delta_{\text{opt}}(G) + 2$. His algorithm first computes an extremal solution to a linear programming relaxation and then runs matroid-intersection algorithms on instances derived from the solution.

Using a lemma of Goemans [19] that characterizes the support of an extremal solution to the linear program, we show that running our push-relabel algorithm in place of the matroid-intersection algorithms gives a different MDMST algorithm that also finds an MST of degree $\Delta_{\text{opt}}(G) + 2$.

The results in Chapter 4 are based on a different approach, leading to true approximation algorithms for the MBDST problem. The first result is a polynomial-time MBDST algorithm that, for any constant $b \in (1, 2)$, finds a spanning tree of optimal cost $c_{\text{opt}_B}(G)$ and degree $\frac{b}{2-b}B+O(\log_b n)$. The main technical contribution is a novel cost-bounding technique requiring a modified MDMST algorithm that also respects certain *lower* bounds on the degrees of vertices.

We then present a quasi-polynomial MBDST algorithm that finds a spanning tree of cost $c_{\text{opt}_B}(G)$ and degree $B + O(\frac{\log n}{\log \log n})$. If the degree bound B is $\omega(\frac{\log n}{\log \log n})$, our algorithm guarantees an approximation factor of $(1 + o(1))$ while still maintaining optimal cost. In addition to the cost-bounding techniques already described, the main technical contribution is the use of augmenting paths, as opposed to the single edge-swaps of local search [14], to find low-degree MSTs.

1.1.3 A generalization of the MDMST problem

Regarded abstractly, the MDMST problem requires us to optimize the maximum degree in a graph G of a minimum-cost base in the graphic matroid, or spanning-tree matroid, of G. We also treat a more general setting where the (hyper)graph and the matroid are not specially related: Given a k-ary hypergraph $G = (V, E)$ and a weighted matroid $M = (E, \mathcal{I}, c)$ such that the ground set of M is the edge set of G, the Minimum-Degree Minimum Cost Base (MDMCB) problem is to find a minimum-cost base T of M that minimizes the degree of T in G. (Complete definitions of the terms and problem are given in Chapter 3, Section 3.6.)

One concrete example of this setting is a network in which each link is controlled by a subset of a set of autonomous entities, with the restriction that no link is controlled by more than k entities. The MDMCB problem in this case is to build an MST of the network such that the maximum number of links controlled by a single entity is minimized. Other natural combinatorial optimization problems can also be formalized as instances of the MDMCB problem.

To our knowledge, the MDMCB problem has not been addressed previously.

The push-relabel algorithm for the MDMST problem generalizes in a straightforward way to the MDMCB problem. Given a k-ary hypergraph $G = (V, E)$ and a weighted matroid M with E as the ground set, the push-relabel MDMCB algorithm outputs in polynomial-time an MCB of M that has degree in G at most $k^2\Delta_{\text{opt}}(G, M) + O(k^{\frac{3}{2}}\sqrt{\Delta_{\text{opt}}(G, M)})$, where $\Delta_{\text{opt}}(G, M)$ is the degree of an optimal solution.

1.1.4 Recent work

Since the initial publication of these results [5, 6], the landscape has changed dramatically.

Ravi and Singh [47] presented an MDMST algorithm that outputs an MST of degree at most $\Delta_{\text{opt}}(G) + k$, where k is the number of distinct weight classes. This bound is incomparable with the results presented here.

The techniques in the aforementioned work of Goemans [19] also work for the MBDST problem, giving an algorithm that finds a spanning tree of cost $c_{\text{opt}_B}(G)$ and degree $B + 2$.

Progress culminated in an MBDST algorithm by Singh and Lau [48] that finds a spanning tree of optimal cost $c_{\text{opt}_B}(G)$ and degree $B + 1$. Their result is based on an adaptation of an iterative rounding technique introduced by Jain [26].

While the recent results of Goemans [19] and of Singh and Lau [48] dominate our results for the MDMST and MBDST problems, the techniques developed in this thesis may be of independent interest. Moreover, we do not see an obvious way to adapt their techniques—the uncrossing lemma and the use of the Nash-Williams theorem [40]—to the more general setting of the MDMCB problem.

1.2 Sorting and selection in posets

The previous section deals with the problem of, given a graph, finding a subgraph that meets certain constraints. In this section, we turn to the problem of reconstructing a directed graph $G = (V, E)$, given only the set V and an *oracle* for the reachability relation in G, that is, a procedure that tells, for a given pair $x, y \in V$, whether x is reachable from y in G, or vice versa. Clearly, the problem is solvable in polynomial-time: a simple algorithm might query the oracle on every potential edge. The question is *how efficiently* G can be recovered.

As an example, suppose we are given a set of biological variables in a system and an experimental procedure which can determine, for any pair of variables, if one is dependent, directly or indirectly, on the other. How many experiments do we need

to perform in order to reconstruct the web of dependencies?

In Chapter 5, Section 5.8.2, we show that these problems essentially reduce to the problem of *sorting* a partially ordered set.

1.2.1 Sorting

Classically, sorting is the process of determining the underlying linear ordering of a set S of n elements. *Comparison algorithms*, in which direct comparisons between pairs of elements of S are the only means of acquiring information about the linear ordering, form an important subclass, including such familiar algorithms as Heapsort, Quicksort, Mergesort, Shellsort and Bubblesort.

We extend the theory of comparison sorting to the case where the underlying structure of the set S is a partial order, in which an element may be larger than, smaller than, or incomparable to another element, and the "larger-than" relation is transitive and irreflexive. Such a set is called a partially ordered set, or poset.

This extension is applicable to many ranking problems where certain pairs of elements are incomparable. Examples include ranking college applicants, conference submissions, tennis players, strains of bacteria according to their evolutionary fitness, and points in R^d under the coordinate-wise dominance relation.

The algorithms described in Chapters 5 and 6 gather information by queries to an oracle. The oracle's response to a query involving elements x and y is either the relation between x and y or a statement of their incomparability. In many applications, a query may involve extensive effort (for example, running an experiment to determine the relative evolutionary fitness of two strains of bacteria, or comparing the credentials of two candidates for nomination to a learned society). We therefore consider two measures of complexity for an algorithm or problem: the *query complexity*, which is the number of queries performed, and the *total complexity*, which is the

number of computational operations of all types performed (basic operations include standard data structure operations involving one or two elements of the poset).

1.2.2 Reconstructing directed graphs

A partial order on a set can be thought of as the reachability relation of a directed acyclic graph (DAG). More generally, a transitive relation (which is not necessarily irreflexive) can be thought of as the reachability relation of a general directed graph. In applications, such as the system of biological variables described earlier, the relation represents the direct and indirect causal influences among a set of variables, processes, or components of a system.

We show that with negligible overhead, the problem of sorting a transitive relation reduces to the problem of sorting a partial order. Our algorithms thus allow one to reconstruct general directed graphs, given an oracle for queries on reachability from one node to another. As directed graphs are the basic model for many real-life networks including social, information, biological and technological networks (see [41] for a survey), our algorithms provide a potential tool for the reconstruction of such networks.

1.2.3 Posets in other contexts

There is a vast literature on algorithms for determining properties of an initially unknown total order by means of comparisons. Partial orders often arise in these studies as a representation of the "information state" at a general step of such an algorithm. In such cases the incomparability of two elements simply means that their true relation has not been determined yet. The present work is quite different, in that the underlying structure to be discovered is a partial order, and incomparability of elements is inherent, rather than representing temporary lack of information. Never-

theless, the body of work on comparison algorithms for total orders provides valuable tools and insights that can be extended to the present context (e.g. [1, 15, 28, 32, 39]).

1.2.4 Related work

The model considered here was previously considered by Faigle and Turán [13], who presented two algorithms for the problem of sorting a partial ordered set, which is termed "identification" of a poset. Only the query complexity of these algorithms is analyzed. We formally describe their results in Chapter 5.

A recent paper [42] considers an extension of the searching and sorting problem to partial orders that are either trees or forests.

1.2.5 The k-selection problem

A natural problem closely related to sorting is the problem of finding the minimal elements of a poset. For example, given a set of college applications, one may only be interested in finding the set of best applications, which are mutually incomparable, rather than in ranking all applications.

This problem generalizes to the k-selection problem, which is to find the set of elements in the bottom k levels of a given poset. We do not know of any previous results for k-selection in posets, though the problem has been studied in the setting of linear orders [17].

1.2.6 Main results

A precise statement of the problem and our results requires a few definitions which we defer to Chapter 5. Roughly speaking, the *width* of a poset is a measure of its complexity. The sorting problem is then defined as: Given a set of n elements and a bound w on the width, completely determine the partial order on the elements.

In Chapter 5, we give the first algorithm of optimal query complexity $O(wn + n \log n)$, meeting the information-theoretic lower bound. This algorithm is based on a careful analysis of the structure of the poset. Natural generalizations of techniques from sorting linear orders do not yield optimal query complexity. We are able to give a generalization of Mergesort, however, which has query complexity $O(wnlogn)$, matching the best previously obtained, and total complexity $O(w^2 n \log n)$. It remains an open question whether optimal query complexity can be achieved efficiently.

We also give algorithms, which are loosely based on a generalization of Quicksort, for the related problems of computing a linear extension of a given poset and computing the heights of all elements.

Finally, we generalize our sorting algorithms to two variants of the sorting model: the case when an upper bound on w is not known a priori and the case of a general transitive relation, or equivalently, the reachability relation for a directed graph.

In Chapter 6, we focus on the k-selection problem. We give upper and lower bounds on the query and total complexity within both deterministic and randomized models of computation. For the case of $k = 1$, we show that the query and total complexity are $\Theta(wn)$. Though we make use of results known for the special case of linear orders, our results require new techniques, particularly to meet the challenge of proving lower bounds.

1.3 Bibliographic notes

The results in Chapters 2–4 first appeared in [5, 6] and are based on joint work with Kamalika Chaudhuri, Satish Rao, and Kunal Talwar. The results in Chapter 5 first appeared in [8] and are based on joint work with Constantinos Daskalakis, Elchanan Mossel, Richard M. Karp, and Elad Verbin.

Part I

Bounded-degree spanning trees

Chapter 2

Technical Introduction

In this chapter, we give a formal introduction to the Mininum-Degree Minimum Spanning Tree and the Minimum Bounded-Degree Spanning Tree problems, for which several algorithms are presented in Chapters 3 and 4.

We begin by reviewing basic definitions and terminology. In Section 2.2, we cover in detail the previous work on these problems, in order to give a context for our results and techniques. Our main contributions to the area are described in Section 2.3. We also give a synopsis of more recent work in Section 2.4. Section 2.5 introduces a combinatorial concept used by all of our algorithms to guarantee the near-optimality of their output. In Section 2.6, we present a linear programming analysis which explains our cost-bounding techniques and illustrates the close relationship between the two problems. Section 2.7 ends the chapter with a proof that the analysis of the best previously known MDMST algorithm is tight.

2.1 Preliminary definitions

Let V be a set of vertices, $E \subseteq \{\{u, v\} : u, v \in V\}$ a set of edges, and $c : E \to \mathbb{R}^+$ a function that assigns a nonnegative real number, called a *cost* or *weight*, to each edge.

The triple $G = (V, E, c)$ is a *weighted graph*. A *spanning tree* $T \subseteq E$ is a minimal subset of edges that connects all of the vertices. For a subset $E' \subseteq E$ of edges, we denote by $c(E') = \sum_{e \in E'} c(e)$ the cost of E'. A *minimum spanning tree* (MST) is a spanning tree whose cost is minimized over all spanning trees.

For a vertex v and subset $E' \subseteq E$ of edges, we denote by $\deg_{E'}(v)$ the *degree* in E' of v, i.e. the number of edges in E' incident on v. For a tree T, the *degree* of T is defined as the maximum degree in T of any vertex, and we denote it by $\Delta(T) = \max_{u \in V} \deg_T(u)$. When T is obvious from context, we simply write its degree as Δ.

A *matroid* is defined to be a pair $M = (E, \mathcal{I})$, where E is a *ground set* of elements and $\mathcal{I}$ is a family of *independent sets* such that (i) $\emptyset \in \mathcal{I}$, (ii) $A \in \mathcal{I}$, $B \subseteq A$ imply that $B \in \mathcal{I}$, and (iii) $A, B \in \mathcal{I}$, $|A| > |B|$ imply that there exists $e \in A \setminus B$ with $B \cup \{e\} \in \mathcal{I}$. A maximum-cardinality independent set of M is called a *base* of M. We recall the fact that, as a collection of bases, the MSTs of a graph G form a matroid on the edges of G.

Definitions related only to the MDMCB problem are reserved for Chapter 3, Section 3.6.

We also recall a few definitions introduced in Chapter 1: For a given graph $G = (V, E)$, let $\Delta_{\text{opt}}(G) = \min_{\text{MST } T} \Delta(T)$ be the minimum degree of an MST of G, and let $c_{\text{opt}_B}(G)$ be the minimum cost of a spanning tree in the set $\{T : \Delta(T) \leq B\}$.

2.1.1 Swaps

A basic approach to exploring the space of MSTs is to compute an arbitrary MST T, and then repeatedly update T by performing what we call edge *swaps*.

Let e be an edge in an MST T. For some edge $e' \in E$, we say that the pair (e, e') is a *swap* with respect to T, or that e and e' may be *swapped*, if the following conditions

hold: (a) $e' \notin T$, (b) the unique cycle in $T \cup \{e'\}$ contains e, and (c) $c(e) = c(e')$. When we perform the swap (e, e'), we remove e from T and add e' to produce another MST T'.

We shall repeatedly make use of the *exchange property* of the minimum spanning tree matroid. This property states that for any two MSTs T, T', and for any edge $e' \in T'$ such that $e' \notin T$, there exists an edge $e \in T$ such that $e \notin T'$ and (e, e') is a swap. Moreover, for any edge $e \in T$ such that $e \notin T'$, there is an edge $e' \in T'$ such that (e, e') is a swap with respect to T. Additionally, the exchange property continues to hold if we force T, T' to contain a set F of edges, and exclude a set R of edges; in this case e (respectively e') lies outside $F \cup R$ if e' (respectively e) does.

2.1.2 Problem definitions

Chapter 1 introduced the Minimum-Degree Minimum Spanning Tree (MDMST) and Minimum Bounded-Degree Spanning Tree (MBDST) problems. We recall their definitions here:

MDMST Problem: Given a weighted graph $G = (V, E, c)$, find a minimum spanning tree of minimum degree $\Delta_{\text{opt}}(G)$.

MBDST Problem: Given a weighted graph $G = (V, E, c)$ and a degree bound $B > 0$, find a spanning tree of minimum cost $c_{\text{opt}_B}(G)$ in the set $\{T \subseteq E : \Delta(T) \leq B\}$.

2.2 Previous techniques

We look first at the special case of unweighted graphs, for which a solution to the MDMST problem immediately gives a solution to the MBDST problem.

2.2.1 Unweighted graphs

The MBDST and MDMST problem are different generalizations of the same unweighted problem: given an unweighted graph $G = (V, E)$, find a spanning tree of G of minimum degree $\Delta_{\text{opt}}(G)$.

Fürer and Raghavachari [16] gave an elegant combinatorial algorithm for this problem that outputs an MST with degree $\Delta_{\text{opt}}(G) + 1$. Like Edmonds' classic algorithm [10] for unweighted matching, it finds an alternating sequence of edge additions and deletions in a laminar family of subtrees of G such that the sequence results in the *improvement* of a node, which in this case means a reduction in the node's degree. The sequence has the attribute that it improves a high-degree node without creating any new high-degree nodes.

While there is a node in the current tree of degree at least $\Delta_{\text{opt}}(G) + 1$, the Fürer -Raghavachari algorithm either finds an improving sequence of edge swaps or it produces a certificate that the maximum degree of *any* spanning tree must be at least $\Delta_{\text{opt}}(G)$. The certificate consists of a set S of nodes whose removal leaves at least $\Delta_{\text{opt}}(G)|S|$ connected components in the graph, implying that the average degree of S in any spanning tree must be at least $\Delta_{\text{opt}}(G)$. We call such a combinatorial certificate a *witness*.

The laminar structure underlying the sequence of swaps depends on the property that an edge $e' \in E$ that is not in a spanning tree T can replace *any* tree edge e on the unique cycle in $T \cup \{e'\}$. In other words, for every edge e, $e \neq e'$, on the cycle in $T \cup \{e'\}$, (e, e') is a swap. This property is not maintained in weighted graphs because a non-tree edge can only replace other tree edges of equal cost. The structure of an improving sequence of swaps in a weighted graph can therefore be significantly more complicated.

2.2.2 Local search

In the general case of the MDMST problem, Fischer [14] gave the only approximation algorithm known prior to the initial publication [5, 6] of the results in this thesis. Given a graph G, his algorithm begins by computing an arbitrary MST and proceeds by executing any swap that improves a degree-d node without introducing new degree-d nodes, for selected high values of d. He shows that when the tree is locally optimal, the maximum degree of the tree is at most $b\Delta_{\text{opt}}(G) + \log_b n$, for any $b > 1$. In Section 2.7, we show that this analysis is tight; hence, to improve the approximation factor, a search algorithm must look beyond single-swap improvements.

2.2.3 Bi-criteria approximation algorithms

The first bi-criteria approximation algorithm for the MBDST problem was given by Ravi et al [45]. Based on a technique for augmenting matchings, the algorithm finds a tree of cost $O(c_{\text{opt}_B}(G) \cdot \log n)$ and degree $O(B \log n)$.

Before the results in this thesis were first published [5, 6], the best results for the MBDST problem were given by Könemann and Ravi [33, 34], via an analysis of a linear programming relaxation for the MBDST problem. The analogy with the general matching problem is perhaps made even more clear in the weighted case, as the approach taken by Könemann and Ravi follows the line taken by Edmonds' [9] in his weighted matching algorithm. The weighted-matching algorithm can be viewed as first finding a solution to the dual of the matching problem: it finds an assignment of penalties to nodes that can be thought of as increasing the cost of adjacent edges, and then it finds a maximal packing of dual variables for subsets of nodes of the graph. Once the dual solution is known, one can ignore the values of the weights on the tight edges and rely solely on an un-weighted matching algorithm.

Similarly, the algorithm of Könemann and Ravi first solves the dual of a linear

programming relaxation for the MBDST problem[1], where the degree constraint B is relaxed slightly to $B(1+\beta)$, for $\beta > 0$. The dual solution is likewise interpreted as an assignment of penalties to nodes, and the algorithm then runs a combinatorial subroutine on a graph in which the edge costs are modified by the dual penalties. In this case, however, the combinatorial subroutine must produce an MST that has minimum degree relative to all MSTs in the graph with modified edge costs. In other words, the combinatorial subroutine must solve the MDMST problem. Könemann and Ravi show that the cost of an MST in the graph with modified edge costs is at most $(1+\frac{1}{\beta})c_{\mathrm{opt}_B}(G)$. By using Fischer's algorithm [14] to find an MST that has approximately minimum degree, they obtain an algorithm that, for any constants $b > 1, \beta > 0$, finds a spanning tree T of the original graph such that T has cost at most $(1+\frac{1}{\beta})c_{\mathrm{opt}_B}(G)$ and degree $bB(1+\beta)+\log_b n$.

In a subsequent paper [35], Könemann and Ravi use primal-dual techniques to avoid the initial step of solving the linear program and obtain similar results for non-uniform degree bounds.

The Euclidean version of the MBDST problem has also been widely studied (for example, [43, 30, 4, 27, 49]). These results are generally of a different nature than those presented here, since they rely strongly on the fact that the cost function respects the triangle inequality.

2.3 Our results and techniques

Our results for the MDMST and MBDST problems are based on two different approaches. The first approach is to significantly improve upon Fischer's approximation algorithm [14] for the MDMST problem. The result is the first constant-factor approximation algorithm for the MDMST problem, which finds an MST of degree

[1] As in the case of Edmonds' non-bipartite matching algorithm, the linear program and its dual are of exponential size, though their solutions can be found in polynomial time.

$2\Delta_{\text{opt}}(G) + O(\sqrt{\Delta_{\text{opt}}(G)})$. This algorithm uses the push-relabel framework developed by Goldberg [20] for the max flow problem; the details of the algorithm are presented in Chapter 3. In Section 2.3.1 below, we discuss the techniques involved. We also show that the push-relabel MDMST algorithm works in a more general setting, described formally in Chapter 3, Section 3.6.

Using the analysis of Könemann and Ravi [34], the push-relabel MDMST algorithm implies an MBDST algorithm that finds a spanning tree of cost at most $(1 + \frac{1}{\beta})c_{\text{opt}_B}(G)$ and of degree $2B(1 + \beta) + O(\sqrt{B(1 + \beta)})$, for any $\beta > 0$. It is the first algorithm to approximate both the degree and the cost to within a constant factor.

For the special case of $B = 2$, i.e. for TSPP, this MBDST algorithm outputs a tree of cost within a $(1 + \epsilon)$-factor of the optimal solution and of maximum degree $O(\frac{1}{\epsilon})$ for any $\epsilon > 0$.[2] Previous algorithms would produce a tree with near-logarithmic degree and cost within a constant factor of the optimal.

Subsequent to the initial publication [6] of this result, Goemans [19] gave an algorithm for the MDMST problem that outputs an MST of degree at most $\Delta_{\text{opt}}(G) + 2$. His algorithm first computes an extremal solution to a natural linear-programming relaxation for the problem, and then runs matroid-intersection algorithms on instances derived from the LP solution. Using a lemma of Goemans [19], we show that running our push-relabel MDMST algorithm in place of the second step gives a different algorithm that also finds an MST of degree at most $\Delta_{\text{opt}}(G) + 2$.

As discussed in Section 2.2.3, the method used by Könemann and Ravi [34] to solve the MBDST problem introduces a trade-off between degree and cost that always creates a constant-factor error either in the degree or the cost. Our second approach

[2] Our work does not assume that the cost function respects the triangle inequality; when the triangle inequality holds, Hoogeveen [23] gives a $\frac{3}{2}$-approximation to TSPP based on Christofides' algorithm.

to the MBDST problem is based on an adaptation of the linear programming analysis that shows how this error can be avoided.

Recall that MBDST algorithm by Könemann and Ravi includes a subroutine that finds an approximate MDMST on a graph whose edge costs have been modified by the dual penalties. We show that an MST (of the modified graph) in which the nodes with positive dual penalties have high degree has low cost in the original graph. Thus, an algorithm that finds an MST of low degree which also meets certain *lower* degree bounds on the nodes with positive dual penalties, can be used as a subroutine to find an MBDST of *optimal* cost. In other words, it can be used to obtain a true approximation algorithm for the MBDST problem. We present this argument, along with the analysis of Könemann and Ravi, in Section 2.6.

This cost-bounding technique motivates the group of algorithms presented in detail in Chapter 4, which are designed to find MSTs that meet both upper and lower degree bounds. Together these results imply two true approximation algorithms for the MBDST problem: a polynomial-time algorithm that finds a spanning tree of cost $c_{\mathrm{opt}_B}(G)$ and degree $\frac{b}{2-b}B + O(\log_b n)$ for any constant $b \in (1, 2)$, and a quasi-polynomial-time algorithm that finds a spanning tree of cost $c_{\mathrm{opt}_B}(G)$ and degree $B + O(\frac{\log n}{\log \log n})$. A discussion of the techniques involved in these algorithms follows below, in Section 2.3.2.

For the sake of a simpler exposition, we describe our MBDST results in the setting of a uniform degree bound B. The results given in Chapter 4 imply analogous results even in the case of more general non-uniform degree bounds. Since our polynomial-time approximation algorithm is based on the Fischer algorithm, it extends as does the algorithm by Könemann and Ravi [35] to non-uniform degree bounds, except that it achieves cost optimality. Our quasi-polynomial-time algorithm extends in an even more straightforward way to non-uniform degree bounds since the error is additive rather than multiplicative.

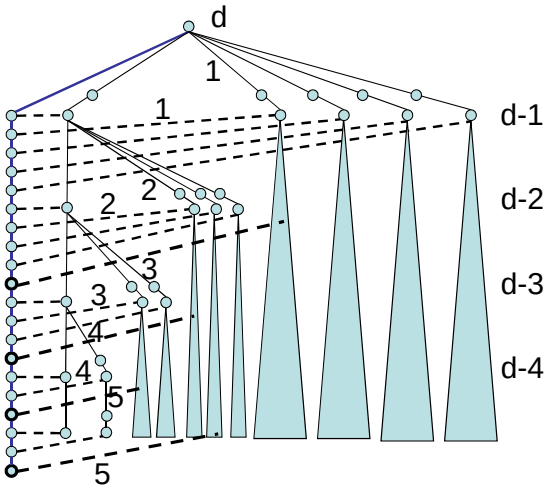

Figure 2.1: Graph G and a locally optimal tree. The shaded triangles represent subtrees identical to the corresponding ones shown rooted at the same level. The bold nodes represent a path and the bold dotted edges correspond to a set of edges going to similar nodes in the subtrees denoted by shaded triangles.

2.3.1 The push-relabel framework

While Fischer's MDMST algorithm [14] is locally optimal with respect to single edge-swaps, the push-relabel MDMST algorithm, described in detail in Chapter 3, explores a more general set of moves that may consist of long sequences of branching, inter-dependent changes to the tree.

To illustrate the limitations of other algorithms, we describe here a pathological MST T in a graph G (see Figure 2.1): The tree T has a long path consisting of $O(n)$ nodes ending in a node u of degree d. Each edge on the path has cost ϵ. Each child of u has degree 2. An edge between u and one of its children has cost 1. The child of each child of u has degree $d-1$. An edge between a degree-2 node and its child has cost ϵ. Each child of a degree-$(d-1)$ node has degree 2. An edge between a degree-$(d-1)$ node and one of its children costs 2. The child of each child of a degree-$(d-1)$ node has degree $(d-2)$. An edge between a degree-2 node and its child has cost ϵ. In general, each child of a degree-$(d-i)$ node has degree 2, and an

edge between a degree $(d-i)$-node and one of its children costs $i+1$. The child of a child of a degree-$(d-i)$ node has degree $d-i-1$. An edge between a degree-2 node and its child has cost ϵ. And so on, until we get to the leaves. In addition, every degree-$(d-i)$ node has a cost-i edge to one of the nodes on the path. For some d such that $d = O(\frac{\log n}{\log \log n})$, the number of nodes in the graph is $O(n)$.

Note that an MST of G with optimal degree consists of the path, the other cost-ϵ edges, and the non-tree edges. It has maximum degree three. On the other hand, every cost-neutral swap that improves the degree of a degree-$(d-i)$ node in the current tree increases the degree of a degree-$(d-i-1)$ node. Hence the tree T is locally optimal for the algorithms of [14, 34]. Moreover, all the improving edges are incident on a single component of low-degree nodes; one can verify that even the algorithms given in Chapter 4, Section 4.5, starting with this tree will not be able to improve its degree. In fact, a slightly modified instance, G', where several of the non-tree edges are incident on the same node on the path, is not improvable beyond $O(d)$. Techniques that do not discriminate between different nodes of degree less than $d-1$ cannot distinguish between G and G'.

On the other hand, the push-relabel MDMST algorithm may perform a swap that improves the degree of a degree-d node by creating one or more new degree-d nodes. In turn, it attempts to improve the degree of these new degree-d nodes, which cannot necessarily be improved independently since their improvements may rely on the same edge or use edges that are incident to the same node. Moreover, this effect snowballs as more and more degree-d nodes are created.

It is surprising that the push-relabel framework can be delicately adapted to explore these sequences. In this framework, each node is assigned a label and each high-degree node is assigned an "excess". The basic idea that we borrow from Goldberg [20] is that "excess" is permitted to flow only from a higher-labeled node to

lower-labeled nodes, that is, it is "pushed" downward. A node is allowed to increase its label when it is unable to get rid of its excess. For max flow, the excess assigned to a node comes from a preflow, while in our case, the excess refers to excess degree.

Thus, a high-degree node may only relieve a unit of excess degree using a non-tree edge that is incident to nodes of lower labels. While two high-degree nodes may be created by a swap, they are guaranteed to have lower labels than the label of the node initiating the swap. Though the algorithm may end up undoing a previous swap, the labels ensure that this process cannot continue indefinitely.

Intuitively, the difference between this algorithm and a local-search algorithm like that of Fischer[14] is that the push-relabel scheme distinguishes between the degree of a node and the node's ability to decrease its degree. The label acts roughly as a proxy for the ease with which a node is able to absorb or rid itself of degree. While a local-search algorithm avoids moves that increase the degree of a high-degree node, the push-relabel algorithm tries to avoid moves that increase the degree of a node with a high label.

We define a notion of a *feasible* labeling and prove that our MDMST algorithm maintains one. This property is crucial for proving that the algorithm outputs a *witness*, or a combinatorial certificate that the tree has near-optimal degree, when it terminates. During the course of the algorithm, there is eventually a label p^* such that the number of nodes with label at least p^* is not much larger than the number of nodes with label at least $p^* + 1$. We use feasibility to show that all nodes with labels at least p^* must have high average degree in *any* MST, thus obtaining a lower bound on $\Delta_{\text{opt}}(G)$. This degree lower bound also holds for any fractional MST of the graph G.

To our knowledge, this algorithm represents the first application of the push-relabel technique to an NP-hard problem. We are intrigued by the possibility that the push-relabel framework may be extended to search what may appear to be com-

plicated neighborhood structures for other optimization problems.

2.3.2 True MBDST approximation algorithms

In Section 2.6, we reduce the problem of getting a true approximation algorithm for the MBDST problem, i.e. one that produces a tree of optimal cost $c_{\mathrm{opt}_B}(G)$, to the problem of finding an MST that meets both upper and lower bounds on degree. Hence, the algorithms presented in detail in Chapter 4 are designed to solve the following problem, which may be of independent interest. It is called the Minimum Spanning Tree with Degree Bounds (MSTDB) problem:

MSTDB Problem: Given a weighted graph $G = (V, E, c)$, a degree upper bound B_H, a subset of vertices $L \subseteq V$, and a degree lower bound B_L, find, if one exists, a minimum spanning tree of G such that no vertex has degree more than B_H and all vertices in L have degree at least B_L.

To solve the MSTDB problem, we develop algorithms for enforcing upper and lower bounds separately and then show how to carefully combine them to simultaneously enforce the high- and low-degree constraints. Despite the appearance of symmetry, the algorithms enforcing lower bounds on degree require new ideas and may be of independent interest. When each distinct algorithm terminates, it provides a combinatorial certificate, which we call a *witness*, of the near-optimality of its output. To introduce the structure of a witness and its use, we describe a simplified version of a witness below in Section 2.5.

Our polynomial-time algorithm for the MSTDB problem, described in detail in Chapter 4, Section 4.3, is a combination of two adaptations of Fischer's algorithm, one of which enforces lower bounds on degrees. For any $b > 1$, this algorithm finds,

if one exists, an MST of G that has degree at most $bB_H + \log_b n$ and in which the nodes in L have degree at least $B_L/b - \log_b n$.

At the cost of a quasi-polynomial running time, we improve the error in degree bounds significantly. Our approach relies on illustrative analogy with bipartite matching: Consider the problem on bipartite graphs of assigning each node on the "left" to a node on the "right" while minimizing the maximum degree of any node on the right. Let Δ'_{opt} be the maximum degree of a node on the right in an optimal solution. This problem can naturally be solved using matching algorithms. Suppose instead that we find a locally optimal solution where, for each node on the left, all of its neighbors have degree within 1 of each other. One can then prove that every node on the right has degree at most $b\Delta'_{\text{opt}} + \log_b n$, for $b > 1$. The proof uses Hall's Theorem to show that any breadth-first search in the graph of matched edges is shallow; the depth of such a search bounds the maximum degree of this graph. This is the principle behind Fischer's algorithm [14] for finding an MST of degree $b\Delta_{\text{opt}}(G) + \log_b n$, for $b > 1$.

For the bipartite-graph problem above, an augmenting- or alternating-path algorithm for matching gives a much better solution than the locally greedy algorithm. Based on this insight, we make use of augmenting paths in the design of an MSTDB algorithm that finds an MST of degree at most $B_H + O(\frac{\log n}{\log \log n})$ and in which the nodes in L have degree at least $B_L - O(\frac{\log n}{\log \log n})$. The algorithm is presented in detail in Chapter 4, Section 4.5. In contrast to Fisher's local single-swap approach, our algorithm looks for a *sequence* of edge swaps in order to decrease the degree of one high-degree node. This introduces significant complications, since every swap in the sequence changes the structure of the tree, thereby changing the set of existing swaps.

Both MSTDB algorithms imply MBDST approximation algorithms with analogous guarantees on degree. These results are described in Chapter 4, Sections 4.4 and 4.6.

2.4 Recent progress

There has been significant progress since the results in this thesis were initially published [5, 6]. Ravi and Singh [47] gave an algorithm for the MDMST problem with an additive error of k, where k is the number of distinct weight classes. We note that this bound is incomparable to those presented here.

As described in Section 2.3, the work of Goemans [19] provides an algorithm for the MDMST problem that achieves an additive error of 2. His techniques work more generally for the MBDST problem, giving a spanning tree of cost $c_{\text{opt}_B}(G)$ and degree $B + 2$. One of the major contributions of this work is the characterization of an extremal solution to the linear programming relaxation for the MBDST problem as a laminar family of tight constraints.

Ultimately, Singh and Lau [48] achieved the goal of an MBDST algorithm that finds a spanning tree of cost $c_{\text{opt}_B}(G)$ and degree $B + 1$, which is the optimal result if P is not equal to NP. Their algorithm is an adaptation of the iterative-rounding techniques introduced by Jain [26]. The result relies on a polyhedral characterization similar to that shown by Goemans.

Though our results for the MDMST and MBDST problems have been surpassed, our techniques may be of interest in their own right and potentially useful in future work. Furthermore, some of the tools used by Goemans and by Singh and Lau, such as the uncrossing lemma and the Nash-Williams theorem [40], do not appear to generalize readily to the MDMCB problem, described in Chapter 3, Section 3.6.

There have also been significant recent developments in what is known about degree-bounded variants of other graph problems. In particular, Lau et al [38] gave a bi-criteria approximation algorithm for a generalization of the MBDST problem called the element-connectivity Survivable Network Design problem with degree constraints on vertices. This result implies, among other things, a constant-factor bi-criteria

approximation algorithm for the Minimum Bounded-Degree Steiner Tree problem.

2.5 Witnesses

An important aspect of all of the MDMST algorithms presented in this thesis is that when they terminate, they produce a proof that the degree of the resulting tree is close to optimal. We call such a combinatorial proof a *witness*.

To illustrate the structure and properties of a witness, we present here a simplified version of the witnesses used in Chapter 4 to prove a lower bound on the degree of an optimal solution. Though it differs in structure, the witness used in Chapter 3 is based on a similar idea. In Chapter 4, we also describe witnesses that prove an upper bound on the degree of a subset of nodes.

We first introduce notation for a decomposition of the graph G that forms the basis for the witnesses produced by the algorithms in Chapter 4. The decomposition is a partition of the nodes of G into a *center set* W and k other sets $C_1, C_2, \ldots, C_k$ called *clusters*.

For a high-degree witness, the partition has the property that each cluster has at least one tree edge to W. In fact, it has the stronger property that in *any* MST of G, there is at least one edge from C_i to W. See Figure 2.2.

If it is also true that k is large, then the average degree of W in any MST must be high. The following lemma formalizes this idea. A similar idea was used by [16].

Lemma 2.1. *Let V be partitioned into sets of nodes $W, C_1, C_2, \ldots, C_k$. If for every cluster C_i, any MST contains an edge connecting C_i to W, then for any MST T of G, $\Delta(T) \geq \lceil \frac{k}{|W|} \rceil$.*

Proof. Since the C_i's are disjoint and each one contains an edge from C_i to W, the total degree of W is at least k. Since the maximum degree in W is larger than the average, the claim follows. □

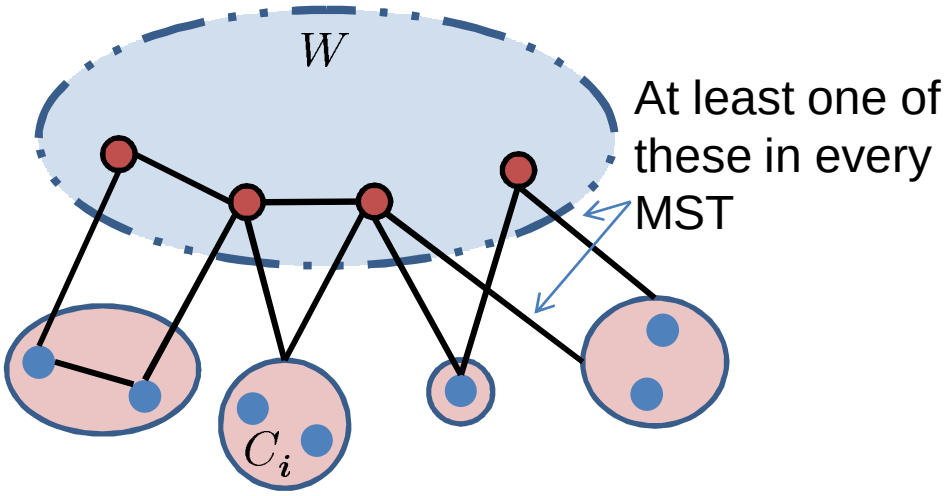

Figure 2.2: Example of a simple high-degree witness, as used in Lemma 2.1.

2.6 Cost analysis via linear programs

Given a graph $G = (V, E, c)$, we show here that an MST for a modified cost function which meets both upper and lower degree bounds is also an MBDST with analogous degree bounds and optimal cost in G.

For the sake of completeness, we also present the argument of Könemann and Ravi [34] that an MDMST for a modified cost function is an MBDST with guarantees on degree and with cost within a constant factor of the optimal.

An integer linear program for the MBDST problem is given by:

$$\begin{aligned} c_{\mathrm{opt}_B}(G) \quad = \quad \min \quad & \sum_{e \in E} c_e x_e \\ \text{s.t.} \quad & \sum_{e \in \delta(v)} x_e \leq B \quad \forall v \in V \\ & x \in \mathrm{SP}_G, \\ & x_e \in \{0, 1\} \end{aligned} \tag{2.1}$$

where $\delta(v)$ is the set of edges of E that are incident to v, and SP_G is the convex hull of edge-incidence vectors of spanning trees of G.

A vector $x \in \text{SP}_G$, the entries of which are not necessarily all integer, is called a *fractional* spanning tree. It can be written as a convex combination of spanning trees of G. In general we use the notation τ and T to refer to a fractional spanning tree and a strictly integral spanning tree, respectively. For a fractional tree τ with edge incidence vector $x \in \text{SP}_G$, let $\deg_\tau(v) = \sum_{e \in \delta(v)} x_e$. Now we can write the linear program relaxation of (2.1) by replacing the integrality constraint by $0 \leq x_e \leq 1$. Because of the integrality of the spanning tree polytope, we can rewrite the relaxation as:

$$\begin{aligned} \min \quad & \sum_{T \in E} c(T)\alpha_T \\ \text{s.t.} \quad & \sum_T \deg_T(v)\alpha_T \leq B \sum_T \alpha_T \quad \forall v \in V \\ & \textstyle\sum_T \alpha_T = 1 \\ & \alpha_T \geq 0 \end{aligned} \tag{2.2}$$

where we have a variable α_T for each spanning tree T of G, i.e. for each vertex of SP_G. Now one can write the dual of this linear program:

$$\begin{aligned} \text{opt}_{LD(B)} \;=\; \max \quad & \nu \\ \text{s.t.} \quad & \nu \leq c(T) + \textstyle\sum_{v \in V} \lambda_v(\deg_T(v) - B) \qquad \forall T \\ & \lambda_v \geq 0 \qquad\qquad\qquad\qquad\qquad\quad \forall v \in V \end{aligned} \tag{2.3}$$

The value $\text{opt}_{LD(B)}$ of an optimal solution to the dual program is a lower bound on $c_{\text{opt}_B}(G)$. This dual linear program, though it has exponentially many constraints, can be solved in polynomial time; in fact it is equivalent to the Lagrangean dual of (2.1) used by Könemann and Ravi [34]:

$$\max_{\lambda \geq 0} \min_{\tau \in \text{SP}_G} \{c(\tau) + \sum_{v \in V} \lambda_v(\deg_\tau(v) - B)\}. \tag{2.4}$$

Let (α^B, λ^B) be a pair of optimal solutions to the primal and dual. Following the

analysis of [34], let us define a new cost function c^{λ^B}, where $c^{\lambda^B}(e) = c_e + \lambda_u^B + \lambda_v^B$ for an edge $e = (u, v)$. Let $\mathcal{O}^B$ be the set $\{T : \alpha_T^B > 0\}$ of spanning trees with positive α_T^B values. The complementary slackness conditions then imply that every T in $\mathcal{O}^B$ minimizes $c(T) + \sum_{v \in V} \lambda_v(\deg_T(v) - B)$. Since $\sum_{v \in V} \lambda_v B$ is independent of T, we conclude that each tree in $\mathcal{O}^B$ minimizes $c(T) + \sum_{v \in V} \lambda_v \deg_T(v) = c^{\lambda^B}(T)$; i.e. every tree in $\mathcal{O}^B$ is a minimum spanning tree under the cost function c^{λ^B}. The optimal solution to the linear program is then a fractional MST (again under c^{λ^B}): $\tau^B = \sum_{T \in \mathcal{O}^B} \alpha_T^B T$. Note that the degree of v in τ^B is $\deg_{\tau^B}(v) = \sum_T \alpha_T^B \deg_T(v)$.

Let $L^B = \left\{v : \lambda_v^B > 0\right\}$ be the set of nodes with positive dual variables in the optimal solution. Complementary slackness conditions then imply the following claim.

Lemma 2.2. *For all $v \in V$, $\deg_{\tau^B}(v) \leq B$, and for all $v \in L^B$, $\deg_{\tau^B}(v) = B$.*

So we are given the existence of a fractional MBDST τ of G such that τ is in fact a fractional MST under the cost function c^{λ^B} and it meets both upper and lower degree bounds (i.e. no node in τ has degree more than B and every node in L^B has degree exactly B). We show now that an *integral* MST, under a slightly different cost function, that meets the lower degree bounds has cost equal to the cost of τ.

More precisely, let $B^* = B(1 + \beta)$, for some $\beta > 0$. Let $T^{B^*} \in \mathcal{O}^{B^*}$, so T^{B^*} is an MST under the cost function $c^{\lambda^{B^*}}$. Let L^{B^*} be the set of nodes with positive dual variables in the optimal fractional LP solution with degree bound B^*. Since λ^{B^*} is a feasible solution for the dual LP (2.3), it is clear that

$$c(T^{B^*}) + \sum_{v \in V} \lambda_v^{B^*}(\deg_{T^{B^*}}(v) - B) \leq \text{opt}_{LD(B)}\,. \tag{2.5}$$

Further, if T^{B^*} has the property that for every node $v \in L^{B^*}$, $\deg_{T^{B^*}}(v)$ is at least B, the second term in the above expression is non-negative and hence $c(T^{B^*})$ is at most $\text{opt}_{LD(B)}$. To summarize:

Theorem 2.3. *Let T be an MST of G under the cost function $c^{\lambda^{B^*}}$ such that for every $v \in L^{B^*}$, $\deg_T(v) \geq B$. Then $c(T) \leq \text{opt}_{LD(B)} \leq c_{\text{opt}_B}(G)$.*

Moreover, Könemann and Ravi show that even if T^{B^*} does not meet lower bounds on the degrees of vertices, the cost of T^{B^*} is not too large compared to the cost of τ.

Theorem 2.4 (Könemann -Ravi [34])**.** *Let T be an MST of G under the cost function $c^{\lambda^{B^*}}$. Then $c(T) \leq \left(1 + \frac{1}{\beta}\right) \text{opt}_{LD(B)}$.*

For the sake of completeness, we include a proof of this result. Let $\tau^{B^*} = \sum_{T \in \mathcal{O}^{B^*}} \alpha_T^{B^*} T$ be an optimal solution to the LP (2.2) for degree bound B^*. Since λ^{B^*} is a feasible solution for the dual LP (2.3) with degree bound B, it is clear that

$$c(\tau^{B^*}) + \sum_{v \in V} \lambda_v^{B^*} (\deg_{\tau^{B^*}}(v) - B) \leq \text{opt}_{LD(B)}. \tag{2.6}$$

Recall that we are guaranteed by complementary slackness conditions that if $\lambda_v^{B^*} > 0$ then $\deg_{\tau^{B^*}}(v) = B^*$. Using this fact along with (2.6), we get

$$\begin{aligned} \text{opt}_{LD(B)} &\geq \sum_{v \in V} \lambda_v^{B^*} (\deg_{\tau^{B^*}}(v) - B) \\ &= \sum_{v \in V} \lambda_v^{B^*} (B^* - B) \\ &= \beta \sum_{v \in V} \lambda_v^{B^*} B. \end{aligned} \tag{2.7}$$

Now let T be any MST of G under the cost function $c^{\lambda^{B^*}}$. Then we arrive at the cost

bound of T in [34] as follows:

$$\begin{aligned} c(T) &= c^{\lambda^{B^*}}(T) - \sum_{v \in V} \lambda_v^{B^*} \deg_T(v) \\ &\leq c^{\lambda^{B^*}}(\tau^{B^*}) \\ &\leq \mathrm{opt}_{LD(B)} + B \sum_{v \in V} \lambda_v^{B^*} && \text{by (2.6)} \\ &\leq \left(1 + \frac{1}{\beta}\right) \cdot \mathrm{opt}_{LD(B)} && \text{by (2.7)}, \end{aligned}$$

from which we conclude Theorem 2.4.

2.7 Tightness of Fischer's analysis

In this section we show that Fischer's analysis [14] of the local swap heuristic is nearly tight. Thus previous search techniques that do not consider multiswap improvements, i.e. *augmenting paths*, can provably not get a constant multiplicative factor.

Theorem 2.5. *Given any integer $t > 1$, there exists an asymptotic family of (unweighted) graphs $\langle G_i \rangle$ and locally optimal trees $\langle T_i \rangle$ such that the maximum degree in T_i is at least t times the optimum for G_i.*

Proof. We first show a family of instances showing a gap of two. We'll show an instance where a locally optimal tree has maximum degree $c \log n$ while the optimal solution has degree $c' \log n$ for some $c' \leq \frac{c}{2}$. The vertex set consists of a root r, and sets L and R. The vertices in L are labelled by binary strings of length at most d and the vertices in R are labelled by tuples (x, j) where x is a binary string of length at most $(d-1)$ and j is a number between 1 and $(d - |x| - 1)$. It is easy to check that the total number of vertices is at most $(d+1)2^d$ which is $O(n)$ for $d = c \log n, c < 1$. There is a white edges from r to each vertex in L. There are red edges between x

and (x, j) for each j and blue edges between $(x, 2i)$ and $x0$ and between $(x, 2i+1)$ and $x1$. There are no L-L or R-R edges.

The locally optimal solution uses all white and red edges and thus the node corresponding to the empty string has degree d.[3] It is easy to check that this solution is locally optimal for the single swap heuristic. On the other hand, the solution using all white and blue edges has maximum degree (on vertex labelled 1) equal to $1 + \lceil \frac{d-1}{2} \rceil$. Further, a slight modification of this tree gives one of degree at most $\frac{d}{2}$. Thus the local optimum has degree at least twice the global one.

To get a gap of t, we use t-ary strings in the definitions of L and R and have blue edges from (x, j) to xi when $(j \mod t = i)$. It is easy to check that for $d = c \log_t n, c < 1$, this construction gives the claimed gap. □

Moreover, note that since the above gap is shown for $\Delta_{\text{opt}}(G) = O(\log n)$, the additive error of the one-swap heuristic is $\Omega(\log n)$.

[3]Strictly speaking, r has the highest degree. However, by replacing the r-L edges by a binary tree with root r and L as the set of leaves, we can get an instance where it has degree 3.

Chapter 3

A push-relabel MDMST algorithm

In this chapter, we present a constant-factor approximation algorithm for the MDMST problem.

We begin by defining notation needed to present the algorithm. Section 3.2 describes our push-relabel algorithm for the MDMST problem. We give two different (and incomparable) bounds on the performance of the algorithm in Sections 3.3 and 3.4. In Section 3.4.2, we use a result of Goemans to derive an algorithm with an additive error of 2. The push-relabel MDMST algorithm implies a result for the MBDST problem which is summarized in Section 3.5. Finally, in Section 3.6, we define a generalization of the MDMST problem, called the Minimum-Degree Minimum-Cost Base (MDMCB) problem, and show that the push-relabel algorithm extends to this setting.

3.1 Additional notation

For a subset $F \subseteq E$ of edges and a subset $U \subseteq V$ of nodes, let $F(U)$ denote the set of edges in F that have both endpoints in U, and let $F[U]$ denote the set of edges incident on F, i.e. $F(U) = \{(u, v) \in F : u, v \in U\}$ and $F[U] = \{(u, v) \in F : u \in U \text{ or } v \in U\}$.

For a node u and a tree T, let S_u^T denote the set of swaps (e, e') in T such that e is incident on u and e' is not incident on u. We call a swap (e, e') in S_u^T *useful* for u because it can be used to decrease the degree of u; i.e. the degree of u in $T \setminus \{e\} \cup \{e'\}$ is one less than that in T.

Let $\mathbf{N}$ be the set of nonnegative integers. A *labeling* l of the nodes is a function $l : V \to \mathbf{N}$. For a labeling l and an integer p, let *level* p be defined as the set $\{v : l(v) = p\}$ of nodes that have label p, and let $W_p = \{v : l(v) \geq p\}$ be the set of nodes with labels at least p. For a real number $\mu \geq 1$, level p is called *μ-sparse* if $|W_p| \leq \mu\,|W_{p+1}|$.

Given a labeling l on V, we extend it to a labeling on E by defining $l(e) = \max\{l(u), l(v)\}$ for $e = (u, v)$. Given a labeling l and an MST T, a swap $(e, e') \in S_u^T$ is called *permissible* for u if $l(u) \geq l(e') + 1$. We say that a labeling l is *feasible* for a tree T if for all nodes $u \in V$, for every swap $(e, e') \in S_u^T$, $l(e) \leq l(e') + 1$.

We defer the definitions for the MDMCB problem to Section 3.6.1.

3.2 The algorithm

Starting with an arbitrary MST of the graph, our algorithm runs in phases. The idea is to reduce the maximum degree of the tree in each phase using a push-relabel technique. If we fail to make an improvement in some phase, we find a certificate of near-optimality.

More formally, let Δ_i be the maximum degree of any node in the tree T_i at the beginning of phase i, also called the Δ_i-phase. During the Δ_i-phase, either we modify T_i to get T_{i+1} such that the maximum degree in T_{i+1} is less than Δ_i, or we output a certificate that Δ_i is close to optimal.

For a general phase of the algorithm, let T be the tree at the beginning of the phase, and let Δ be the degree of T. The algorithm maintains a labeling l. In

addition, each node is given an initial *excess*. (Excess can also be formally defined as a function from V to $\mathbf{N}$.) The excess of a node is positive if its degree is at least Δ. We call a vertex that has positive excess *overloaded*. The algorithm maintains the invariant that the labeling l is feasible with respect to the current tree T. This notion of feasibility is crucial in establishing a lower bound on the optimal degree when the algorithm terminates.

The PR-MDMST algorithm takes as input a graph G and a real-valued parameter $\mu \geq 1$. The parameter μ determines the termination condition of the main loop of the algorithm. In Sections 3.3 and 3.4, we derive two different bounds on the approximation ratio of the algorithm; the parameter μ is chosen appropriately in the two cases.

We now describe a general phase of the algorithm. See Figure 3.1 for a formal description. The phase proceeds as follows: The label $l(u)$ of each node u is initialized to zero. The excess of each node of degree Δ is initialized to one; the excess of every other node is initialized to zero.

Let p be the label of the lowest level containing overloaded nodes. If there is an overloaded node u in level p that has a permissible, useful swap $(e, e') \in S_u^T$, modify T by deleting $e = (u, v)$ and adding $e' = (u', v')$. Then decrease the excess on u by one; if v has positive excess, decrement its excess as well. If u' now has degree Δ or more, add one to its excess; if v' has degree Δ or more, add one to its excess. If no overloaded node in level p has a permissible, useful swap, then *relabel* to $p + 1$ all overloaded nodes in level p. Repeat this loop until either there are no overloaded nodes or there is a μ-sparse level. Note that if the phase ends for the former reason, then the tree at the end of the phase has maximum degree at most $\Delta - 1$. In the latter case, we show that Δ is close to the optimal degree.

If some node gets label n, there is guaranteed to be a 1-sparse level. Thus each node gets relabeled at most n times per phase, for any choice of the input parameter

Algorithm PR-MDMST(G, μ)

$T \leftarrow$ arbitrary MST of G.
Repeat
  $\Delta \leftarrow$ maximum degree over nodes in T.
  Initialize labels to 0.
  Put excess of 1 on nodes with degree Δ.
  Put excess of 0 on all other nodes.
  Repeat
    $p \leftarrow$ lowest level that contains an overloaded node.
    Select the set $U_p \leftarrow$ overloaded nodes with label p.
    If there is a node $u \in U_p$ that has a permissible, useful swap (e, e')
        where $e = (u, v)$
      $T \leftarrow T \setminus \{e\} \cup \{e'\}$;
      Set excess on the endpoints of e to 0;
      For each endpoint of e' that has degree Δ or more
        set its excess to 1.
    else
      Relabel all nodes in U_p to $p + 1$.
  until there are no more overloaded nodes
    or there is a μ-sparse level.
until there is a μ-sparse level p^*.
Let $F \subset T$ be the edges in T *not* incident on W_{p^*+1}.
Output tree T and the pair $\mathcal{W}^{p^*} = (F, W_{p^*})$.

Figure 3.1: A push-relabel algorithm for the MDMST problem.

$\mu \geq 1$. The total number of iterations of the inner loop in any phase of the algorithm is therefore bounded by n^2. Since each phase (except the last) decreases the maximum degree of T by one, there are at most n phases. The algorithm therefore runs in polynomial time.

The algorithm outputs a tree T and a pair $\mathcal{W}^{p^*} = (F, X)$, where F is a forest on G and X is a subset of nodes. In the rest of this section, we show how to interpret (F, X) as a certificate that the degree of T is close to $\Delta_{\text{opt}}(G)$. We do this in two different ways, in Sections 3.3 and 3.4, leading to two incomparable bounds on the approximation ratio of the algorithm.

Remark. In Section 3.3, we set μ to a constant larger than 2. In this case, the number of relabels per node is bounded by $\log_2 n$, resulting in a faster algorithm.

Remark. The argument in Section 3.4 works even if the algorithm chooses an arbitrary set of overloaded nodes in the Select step of the algorithm in Figure 3.1 (instead of using the lowest overloaded level). This leads to a simpler algorithm.

3.2.1 Feasibility

We first prove a crucial lemma.

Lemma 3.1. *The algorithm always maintains a feasible labeling.*

Proof. We prove this by induction on the number of iterations in a phase. At the beginning of any phase, all labels are zero, which is a feasible labeling. In one step of the algorithm, we either update a label or perform a permissible swap (e, e'). Since we increment the label of a node only when it has no permissible swaps, feasibility is maintained in the first case.

In the second case, since we change the structure of the tree, the set of available swaps may change. Consider a feasible swap (e, e'), where $e = (u, v)$ and $e' = (u', v')$, and the swap is permissible for u or v (or both). Let T be the tree before the (e, e') swap, and let T' be the tree after the swap. Consider a swap (f, f') in T'. To show that feasibility is maintained, we need to show that $l(f) \leq l(f') + 1$.

If the swap (f, f') already exists in T, feasibility holds inductively. However, the swap may have been missing in the tree T, but may appear in tree T' for one of the following three reasons.

- $f' \in T$ *and hence not available for the swap:* See Figure 3.2(a). In this case, $f' = e$. The cycle formed by adding f' to T' includes e' and f, otherwise T already has a cycle. Moreover $c(f) = c(f') = c(e')$. Therefore (f, e') is a swap

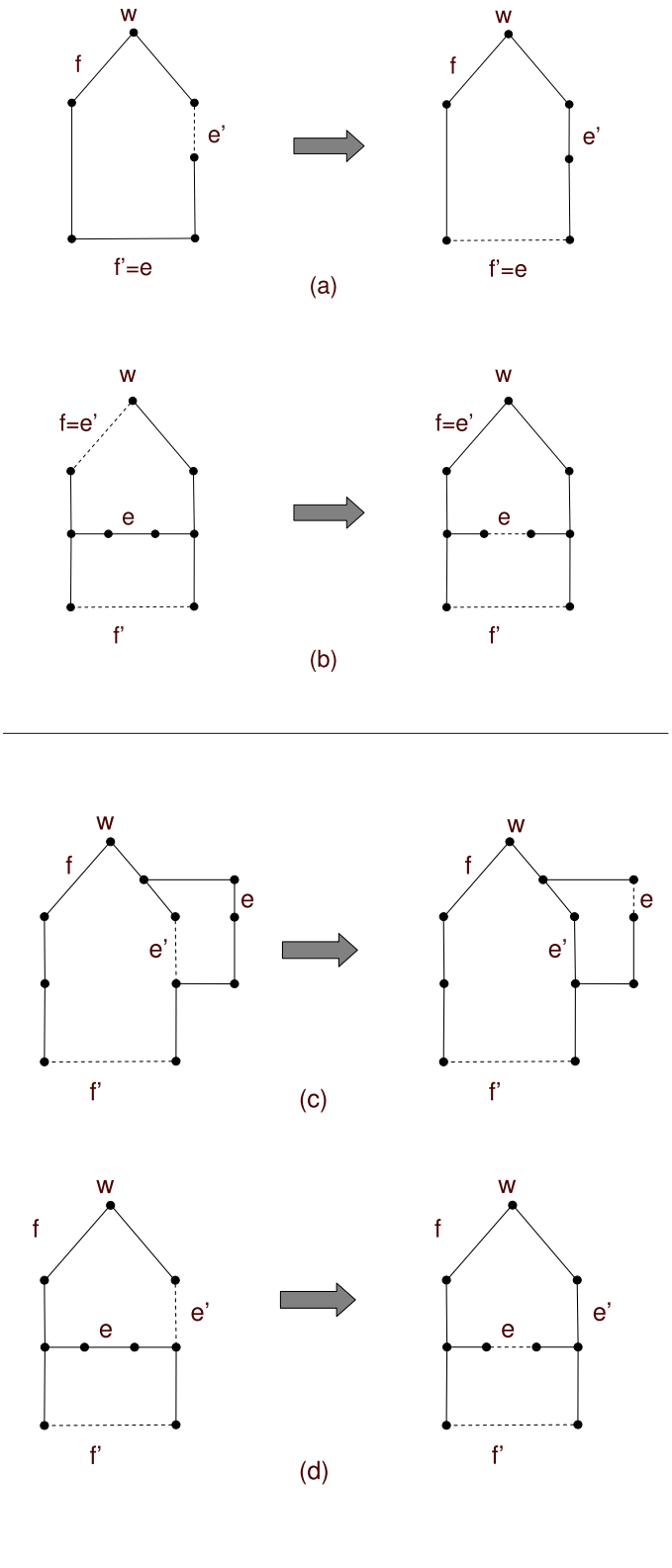

Figure 3.2: Proof of Lemma 3.1.

in T, and feasibility in T implies that $l(f) \leq l(e') + 1$. On the other hand, since (e, e') is a permissible swap in T, $l(e) \geq l(e') + 1$. Thus $l(f) \leq l(e') + 1 \leq l(e) = l(f')$.

- *$f \notin T$ and hence not available for the swap:* See Figure 3.2(b). In this case, $f = e'$. As (e, e') is a swap in T and (e', f') is a swap in T', the cycle formed by adding f' to T includes e, and (e, f') is a valid swap in T. $l(e) \leq l(f') + 1$ by feasibility of T, and $l(e) = l(e') + 1$ by permissibility. Thus $l(f) = l(e') \leq l(f')$.
- *$f \in T$ and $f' \notin T$:* See Figures 3.2(c) and 3.2(d). Since (e, e') is a swap in T, there is a unique cycle in $T \cup e'$ that contains e. If f does not lie on this cycle, as illustrated in Figure 3.2(c), the swap (f, f') already exists in T and the claim holds by induction. Otherwise the cycle in $T \cup e'$ contains f, and the only structure in which this happens is illustrated in Figure 3.2(d). An MST of G cannot contain the heaviest edge of any cycle in G. Since e' is the missing edge of the cycle in $T \cup e'$, it must be the case that $c(e') \geq c(f) = c(f')$. Similarly, since f' is the missing edge of the cycle in $T \cup f'$, $c(f') \geq c(e) = c(e)$. Therefore $c(e) = c(e') = c(f) = c(f')$. Moreover, (f, e') and (e, f') are both available swaps in T. Thus $l(f) \leq l(e') + 1 = l(e) \leq l(f') + 1$.

We have shown that, in all cases, the swap (f, f') is feasible. Hence the induction holds. □

3.2.2 The witness

In this section, we describe the witness produced by the push-relabel algorithm at termination to guarantee the near-optimality of the output. Its structure is different from the witness introduced in Chapter 2, Section 2.5, though it is based on a similar idea.

The witness consists of a forest $F \subseteq E$ that is contained in some MST of G, along with a subset $X \subset V$. A pair (F, X) is a witness if it has the following property: for every MST T of G containing F, every edge in $T \setminus F$ is incident on X.

Lemma 3.2. **[14]** *Let $\mathcal{W} = (F, X)$ be a witness for a graph $G = (V, E)$ as defined above. Then any (fractional) minimum spanning tree of G has maximum degree at least*

$$\frac{|V| - |F| - 1}{|X|}.$$

Proof. Consider an MST T of G, and let T' be an MST containing F that has maximal intersection with T. By the exchange property, $T' \setminus F$ is contained in T. The witness property implies that every edge in $T' \setminus F$ is incident on X. Since there are $|V|-|F|-1$ edges in $T' \setminus F$, the average degree of X in $T' \setminus F$, and therefore in T, is at least $\frac{|V|-|F|-1}{|X|}$. Since a fractional MST is a convex combination of integral ones, the claim follows. □

We now show that the pair (F, X) output by the algorithm PR-MDMST is a witness.

Lemma 3.3. *Let T be the MST of G and $l : V \to \mathbf{N}$ the labeling when the algorithm* PR-MDMST *terminates. For any integer p, let F be the subset of edges in T that are not incident on W_{p+1}, and let $X = W_p$. Then $\mathcal{W}^p = (F, X)$ is a witness.*

Proof. Assume the contrary, and let T' be an MST of G that contains F and also contains an edge $e' \notin F$ not incident on W_p. By the exchange property, there is an edge $e \in T \setminus F$ such that (e, e') is a swap in T. Since $e \in T \setminus F$, it is incident on W_{p+1} and thus $l(e) \geq p + 1$. On the other hand, e' is not incident on W_p and thus $l(e') \leq p - 1$. This, however, contradicts the feasibility of the labeling. □

3.2.3 Involuntary losses

Let p^* be the μ-sparse level used by the algorithm to compute a witness. From Lemma 3.2 and Lemma 3.3, it follows that any (fractional) MST of G has degree at least

$$\frac{|V| - |F| - 1}{|W_{p^*}|}.$$

This ratio can be rewritten as

$$\frac{(|V| - |F| - 1)}{|W_{p^*+1}|} \cdot \frac{|W_{p^*+1}|}{|W_{p^*}|}.$$

Note that the numerator of the first term is precisely the number of edges incident on W_{p^*+1} in T. The second term is bounded by $\frac{1}{\mu}$, where μ is the sparseness of level p^*. The next lemma follows immediately.

Lemma 3.4. *Let T be the MST of G, $l : V \to \mathbf{N}$ the labeling, and p^* the μ-sparse level when the algorithm* PR-MDMST *terminates. Let $T[W_{p^*+1}]$ be the set of edges in T incident on W_{p^*+1}. Then any (fractional) MST of G has degree at least*

$$\frac{1}{\mu} \cdot \frac{|T[W_{p^*+1}]|}{|W_{p^*+1}|}.$$

Thus, to prove a lower bound on $\Delta_{\text{opt}}(G)$, we need to lower bound $|T[W_{p^*+1}]|$. Towards this end, we distinguish between two different ways a node can lose degree during the course of the algorithm.

We say that a swap (e, e') executed by the algorithm *causes* a loss in degree to a node u if e is incident on u. A loss in degree to a node u that is caused by a swap (e, e') is called a *voluntary loss* if u is overloaded before the swap is executed; otherwise it is called an *involuntary loss*. By definition, voluntary losses do not decrease the degree

of a node below $\Delta - 1$. Note that every swap (e, e') executed by the algorithm causes a voluntary loss to at least one endpoint of e (and an involuntary loss to at most one endpoint of e).

Suppose the algorithm terminates with a μ-sparse level in the Δ-phase. The last time it is relabeled, each node in W_{p^*+1} has degree at least Δ and is therefore overloaded. If each node in W_{p^*+1} suffered only voluntary losses in degree since its last relabeling, then its degree in T would be at least $\Delta - 1$. However, a node in W_{p^*+1} may also suffer from involuntarily losses, which may decrease its degree arbitrarily. Hence a node-by-node analysis is insufficient. To get a lower bound on the average degree of W_{p^*+1} in T, we instead bound the total number of involuntary losses to nodes in W_{p^*+1}. We do this in two different ways in the next two sections.

3.3 A constant-factor approximation

In this section, we show that the algorithm outputs a tree T of degree

$$\Delta \leq 2\Delta_{\text{opt}}(G) + O(\sqrt{\Delta_{\text{opt}}(G)}).$$

To bound the number of involuntary losses, we define a partitioning of the swaps executed by the algorithm into *cascades*. Each cascade can be charged to a relabel, which enables us to bound the number of involuntary losses to W_{p^*+1} in terms of the size of this set.

3.3.1 Cascades

Recall that, for an integer p, U_p is defined to be the set of overloaded nodes in level p. For the purpose of analysis, we introduce the notion of flagging a node. The flag indicates that the node has been relabeled but its excess has not been removed. In

addition, we give each node an *overloading-swap* field. The overloading-swap field of a node u points to the swap that put excess on it after its last relabel. We start with all the flags cleared and all overloading-swap fields set to null.

In each iteration of the Δ-phase of the algorithm, we find the lowest p such that U_p is non-empty, i.e. there is an overloaded node with label p. If we can find any swap (e, e') that is permissible and useful for a node in U_p, we execute the swap and clear flags (if set) on the endpoints of e. Moreover, for each endpoint u' of e' that is now overloaded, we set the overloading-swap field of u' to (e, e'). If there is no swap that is permissible and useful for a node in U_p, we increment the label of, set the flag on, and clear the overloading-swap field for each node in U_p.

The following lemma shows that no node has excess larger than one during the course of the algorithm, which implies, in particular, that the overloading-swap field is never overwritten before it is cleared to null.

Lemma 3.5. *During the Δ-phase, no node ever has degree more than Δ.*

Proof. We use induction on the number of swaps executed during the phase. In the beginning of the phase, the maximum degree is Δ. Any swap (e, e') decreases the degree of a node in U_p and adds at most one to the degree of a node with strictly lower label. By choice of p, all nodes with lower labels have degree at most $\Delta - 1$ before the swap. Since a swap adds at most one to the degree of any vertex, the induction holds. The lemma follows. □

We define the *label of a swap* (e, e') to be the label of e when the swap is executed. We call a swap a *root swap* if it is useful for a flagged node. Note that a flagged node has its overloading-swap field set to null. Let (e, e') be a non-root swap that occurs in the sequence of swaps executed by the algorithm. The swap (e, e') is executed in order to relieve the excess of an endpoint u of e. Let (f, f') be the swap pointed to by the overloading-swap field for node u when swap (e, e') is executed. Thus (f, f') is

the last swap in the sequence that increases the degree of u and precedes (e, e'). We call (f, f') the *parent swap* of (e, e'). Note that the flagging procedure ensures that every non-root swap executed by the algorithm has a parent.

A label-p swap, by definition, reduces the degree of a node with label p. Since excess flows from a higher-labeled node to a lower-labeled node, the label of every non-root swap is strictly smaller than the label of its parent. The parent relation naturally defines a directed graph on the set of swaps, each component of which is an in-tree rooted at one of the root swaps. We define a *cascade* to be the set of swaps in a component of this DAG. In other words, a cascade corresponds to the set of swaps sharing the same root swap as an ancestor. Note that the cascades may be interleaved in the sequence of swaps executed by the algorithm. The *label of a cascade* is defined to be the label of the root swap in it.

Each swap is the parent of at most two swaps which each have a strictly smaller label. Thus it follows that:

Lemma 3.6. *A label-p cascade contains at most 2^{p-q} label-q swaps.*

We say that an involuntary loss is *contained* in a cascade if some swap in the cascade causes it. Since each swap causes at most one involuntary loss, the lemma above implies:

Corollary 3.7. *A label-p cascade contains at most $(2^{p-q+1} - 1)$ involuntary losses to nodes with labels at least q.*

Proof. An involuntary loss to a label-r node must be caused by a swap with label at least r. The bound follows by summing the number of swaps with label r, for r between q and p. □

3.3.2 Computing the approximation ratio

Armed with the bound of Corollary 3.7, we now proceed to lower bound $\Delta_{\text{opt}}(G)$. Recall from Section 3.2.3 that it suffices to lower bound the average degree of W_{p^*+1} in T, where W_{p^*} is a μ-sparse level and T is the tree output by the algorithm.

Lemma 3.8. *Let p be an integer greater than p^*. Then $|W_p| > \mu\,|W_{p+1}|$.*

Proof. Each iteration of a phase of the algorithm decreases the size of at most one level p and increases the size of the level $p+1$. Thus the only level that can go from not being μ-sparse to being μ-sparse is level p. Since the algorithm terminates as soon as it finds a μ-sparse level, it terminates with exactly one μ-sparse level. □

Lemma 3.9. *The number of involuntary losses to nodes in W_{p^*+1} is at most*

$$2\,|W_{p^*+1}|\left(\frac{\mu}{\mu-2}\right).$$

Proof. Each involuntary loss to a node in W_{p^*+1} occurs in a cascade, and by Corollary 3.7, the number of involuntary losses to nodes in W_{p^*+1} in a label-p cascade is at most 2^{p-p^*}. The total number of involuntary losses to nodes in W_{p^*+1} during the course of the phase is at most

$$\begin{aligned}
\sum_{p\geq p^*+1} 2^{p-p^*}|W_p| &< \sum_{p\geq p^*+1} 2^{p-p^*}\frac{|W_{p^*+1}|}{\mu^{p-p^*-1}} \\
&= 2\,|W_{p^*+1}|\sum_{p\geq p^*+1}\left(\frac{2}{\mu}\right)^{p-p^*-1} \\
&= 2|W_{p^*+1}|\left(\frac{\mu}{\mu-2}\right)
\end{aligned}$$

□

We are now ready to establish the approximation ratio of the algorithm.

Theorem 3.10. *Given a graph G and a constant $\mu > 2$, the* PR-MDMST *algorithm obtains in polynomial time an MST of degree Δ, where*

$$\Delta \leq \mu\, \Delta_{\text{opt}}(G) + 2 + \frac{2\mu}{\mu - 2}.$$

Proof. The PR-MDMST algorithm, when executed on graph G, terminates with a tree T and a pair $\mathcal{W}^{p^*} = (F, X)$. We now compute the number $|T[W_{p^*+1}]|$ of edges incident on W_{p^*+1} in T. Each node in W_{p^*+1} has degree at least $(\Delta - 1)$ after its last voluntary loss, and it may then suffer some involuntary losses. Using the bound from Lemma 3.9, the sum of degrees of nodes in W_{p^*+1} in T is at least $\left(\Delta - 1 - \frac{2\mu}{\mu-2}\right)|W_{p^*+1}|$. Since there are at most $|W_{p^*+1}| - 1$ edges in T that have both endpoints in W_{p^*+1}, the number of edges in T that are incident on W_{p^*+1} is at least $\left(\Delta - 2 - \frac{2\mu}{\mu-2}\right)|W_{p^*+1}|$.

Thus from Lemma 3.4,

$$\Delta_{\text{opt}}(G) \geq \left(\Delta - 2 - \frac{2\mu}{\mu - 2}\right)\frac{1}{\mu}.$$

Rearranging, we get $\Delta \leq \mu\, \Delta_{\text{opt}}(G) + 2 + \frac{2\mu}{\mu-2}$. □

Setting μ to be $2 + \frac{2}{\sqrt{\Delta_{\text{opt}}(G)}}$, we get

$$\Delta \leq 2\Delta_{\text{opt}}(G) + 4\sqrt{\Delta_{\text{opt}}(G)} + 4.$$

Corollary 3.11. *Given a graph G, there is a polynomial-time algorithm that outputs an MST of degree Δ, where*

$$\Delta \leq 2\Delta_{\text{opt}}(G) + O(\sqrt{\Delta_{\text{opt}}(G)}).$$

3.4 An additive approximation

In this section, we show a different upper bound on the performance of the algorithm that is better than the bound in the previous section if the graph G is *everywhere-sparse*, i.e. for every subset U of nodes, the induced subgraph on U is sparse.

More precisely, for a graph G, let the *local density* $s(G)$ be defined as the density of the densest subgraph of G:

$$s(G) = \max_{U \subset V} \left\{ \frac{|E(U)|}{|U|} \right\}.$$

We show that on input G and $\mu = 1$, the PR-MDMST algorithm outputs an MST of degree at most $\Delta_{\text{opt}}(G) + s(G)$. In Section 3.4.2, we combine this result with a lemma from Goemans [19] to get a $(\Delta_{\text{opt}}(G) + 2)$-algorithm.

3.4.1 A density-based bound

Let T be the tree and $\mathcal{W}^{p^*}$ the witness output by the PR-MDMST algorithm on input G and $\mu = 1$. Let Δ be the degree of T. Since $\mu = 1$, the sets W_{p^*} and W_{p^*+1} must be equal, and hence, level p^* is empty when the algorithm terminates. The following lemma bounds $|T[W_{p^*+1}]|$.

Lemma 3.12. *Let T be the MST, l the labeling, and p^* the empty level when the algorithm* PR-MDMST *terminates. Let $E(W_{p^*+1})$ be the set of edges in G that have both endpoints in W_{p^*+1}. Then the number of edges in T that are incident on W_{p^*+1} is at least*

$$(\Delta - 1)\,|W_{p^*+1}| + 1 - |E(W_{p^*+1})|\,.$$

Proof. For a node $u \in W_{p^*+1}$, let $T[u] \subset T$ be the set of edges in T incident on u, and

let $L[u] \subset E$ be the set of edges that node u loses involuntarily after its last voluntary loss. Since each node has degree at least $\Delta - 1$ after its last voluntary loss, it follows that

$$|T[u]| \geq \Delta - 1 - |L[u] \setminus T[u]| .$$

Moreover, if v is the last node to be relabeled, $|T[v]| \geq \Delta$. Thus the sum of degrees of nodes in W_{p^*+1} in T is at least

$$(\Delta - 1) |W_{p^*+1}| + 1 - \sum_{u \in W_{p^*+1}} |L[u] \setminus T[u]| .$$

An edge (u, v) may be in $L[u]$ or $L[v]$ but not both. It follows that

$$\sum_{u \in W_{p^*+1}} |L[u] \setminus T[u]| \leq \left| \cup_{u \in W_{p^*+1}} (L[u] \setminus T[u]) \right| .$$

Recall that every involuntary loss to a node in W_{p^*+1} comes from an edge in $E(W_{p^*+1})$. Thus each edge in $\cup_{u \in W_{p^*+1}} (L[u] \setminus T[u])$ is in $E(W_{p^*+1}) \setminus T(W_{p^*+1})$.

An edge (u, v) in $T(W_{p^*+1})$ contributes to both $T[u]$ and $T[v]$. Thus the number of edges in T incident on W_{p^*+1} is

$$\begin{aligned}
&\sum_{u \in W_{p^*+1}} |T[u]| - T(W_{p^*+1}) \\
&\geq (\Delta - 1) |W_{p^*+1}| + 1 - \left| \cup_{u \in W_{p^*+1}} (L[u] \setminus T[u]) \right| - T(W_{p^*+1}) \\
&\geq (\Delta - 1) |W_{p^*+1}| + 1 - |E(W_{p^*+1}) \setminus T(W_{p^*+1})| - T(W_{p^*+1}) \\
&\geq (\Delta - 1) |W_{p^*+1}| + 1 - |E(W_{p^*+1})|
\end{aligned}$$

□

We now use the bound on $|T[W_{p^*+1}]|$ in Lemma 3.12 to prove a lower bound on $\Delta_{\text{opt}}(G)$.

Theorem 3.13. *Let T and $\mathcal{W}^{p^*}$ be the output of the* PR-MDMST *algorithm given a graph G and $\mu = 1$. Then the degree Δ of T is bounded by $\Delta_{\mathrm{opt}}(G) + \lceil\, s(G) \,\rceil$.*

Proof. From Lemma 3.12, the number of edges in T incident on W_{p^*+1} is at least

$$(\Delta - 1)\,|W_{p^*+1}| + 1 - |E(W_{p^*+1})|\,.$$

By the definition of local density,

$$\frac{|E(W_{p^*+1})|}{|W_{p^*+1}|} \leq s(G).$$

Lemma 3.4 then implies that

$$\Delta_{\mathrm{opt}}(G) \geq \Delta - s(G) - 1 + \frac{1}{|W_{p^*+1}|}.$$

Since $\Delta_{\mathrm{opt}}(G)$ is an integer, we conclude that $\Delta_{\mathrm{opt}}(G) \geq \Delta - \lceil\, s(G) \,\rceil$. □

3.4.2 An additive error of 2

Goemans [19] shows that the support of the natural linear program from the MDMST problem is sparse. Given a graph G, he considers the following linear program:

$$\min\ c(x) = \sum_e c_e x_e$$

subject to:

$$\begin{aligned}
&\sum_e c_e x_e = C^* && \\
&x(E(S)) \leq |S| - 1 && S \subset V \\
&x(E(V)) = |V| - 1 && \\
&x(\delta(v)) \leq k && v \in V \\
&x_e \geq 0 && e \in E
\end{aligned}$$

where C^* is the cost of an MST in G. One can do a binary search on the value of k to find the smallest k for which a solution exists, i.e. to find a lower bound on the value of $\Delta_{\text{opt}}(G)$. Let x^* be an optimal extreme-point solution to the above linear program for a graph G and for $k \leq \Delta_{\text{opt}}(G)$. Let E^* denote the support of x^*, i.e. $E^* = \{e \in E : x^*_e > 0\}$. Theorem 5 in Goemans [19] can be paraphrased as:

Theorem 3.14 (Goemans [19, Theorem 5])**.** *The local density of the graph $G^* = (V, E^*)$ is less than 2.*

We first argue that $\Delta_{\text{opt}}(G) = \Delta_{\text{opt}}(G^*)$. Since $x^*_e = 0$ for all $e \notin E^*$, it follows that x^* is a feasible solution to the linear program for $G^* = (V, E^*)$, and so $\Delta_{\text{opt}}(G^*) \leq \Delta_{\text{opt}}(G)$. Since G^* is a subgraph of G, $\Delta_{\text{opt}}(G^*) \geq \Delta_{\text{opt}}(G)$. We conclude that $\Delta_{\text{opt}}(G) = \Delta_{\text{opt}}(G^*)$.

Since the linear program can be solved efficiently, we can compute the graph G^*

in polynomial time. Our $(\Delta_{\text{opt}}(G)+2)$-algorithm computes the graph G^* and then runs the algorithm PR-MDMST with input G^* and $\mu = 1$. By Theorem 3.13, the tree T output by the algorithm has degree most $\Delta_{\text{opt}}(G)+2$. Theorem 3.15 summarizes.

Theorem 3.15. *Given a graph G, there is a polynomial-time algorithm that computes an MST of G with degree at most $\Delta_{\text{opt}}(G)+2$.*

3.5 An MBDST algorithm

From Theorem 2.4 and Theorem 3.11, we derive the following theorem.

Theorem 3.16. *For any $\beta > 0$, there is a polynomial-time algorithm that, given a graph G and degree bound B, computes a spanning tree T with maximum degree at most $2(1+\beta)B + O(\sqrt{(1+\beta)B})$ and cost at most $\left(1+\frac{1}{\beta}\right) c_{\text{opt}_B}(G)$.*

For example, for $\beta = 1$, we produce a spanning tree of cost at most twice the optimal and of degree at most $4B + 4\sqrt{2B} + O(1)$.

3.6 The MDMCB problem

In this section, we consider a generalization of the MDMST problem.

3.6.1 Definitions

Recall that a *matroid* is defined to be a pair $M = (E, \mathcal{I})$, where E is a ground set of elements and $\mathcal{I}$ is a family of *independent sets* such that (i) $\emptyset \in \mathcal{I}$, (ii) $A \in \mathcal{I}$, $B \subseteq A$ imply that $B \in \mathcal{I}$, and (iii) $A, B \in \mathcal{I}$, $|A| > |B|$ imply that there exists $e \in A \setminus B$ with $B \cup \{e\} \in \mathcal{I}$. A maximum-cardinality independent set of M is called a *base* of M.

Let $c : E \to \mathbb{R}^+$ be a non-negative cost function on the ground set of M. A base T of M that minimizes the cost $c(T) = \sum_{e \in T} c(e)$ is referred to as a *minimum-cost base* (MCB). Let $x^T \in \{0,1\}^{|E|}$ be the incidence vector of an MCB T. We say a vector $x \in [0,1]^{|E|}$ is a fractional MCB if it is the convex combination of incidence vectors of MCB's.

Recall that a hypergraph $G = (V, E)$ consists of a set of nodes V and hyperedges $E \subseteq 2^{2^V}$, i.e. each hyperedge is a subset of vertices. We say G is a k-ary hypergraph if the cardinality of each edge in E is at most k. If $u \in e$, we say that node u is an *endpoint* of e and that e is *incident* on u. For a subset $F \subseteq E$ of edges and a node $u \in V$, we define the *degree of* u in F to be $|\{e \in F : u \in e\}|$. The *degree of* F is then defined as the maximum over $u \in V$ of the degree of u in F, i.e. $\max_u |\{e \in F : u \in e\}|$. Further, let $F(U)$ and $F[U]$ denote the sets $\{e \in F : e \subset U\}$ and $\{e \in F : e \cap U \neq \emptyset\}$ respectively.

We can now formally define the minimum-degree minimum-cost base (MDMCB) problem. The input to the problem is a k-ary hypergraph $G = (V, E)$ and a matroid M with E as its ground set.

MDMCB Problem: Given a k-ary hypergraph $G = (V, E)$ and a weighted matroid $M = (E, \mathcal{I}, c)$, find an MCB of M that has minimum degree in G.

Let $\Delta_{\text{opt}}(G, M)$ denote the optimal degree for an instance of MDMCB.

3.6.2 The MDMCB algorithm

Our algorithm for MDMCB is a generalization of the PR-MDMST algorithm. To define it formally, we first define some analogous concepts. Given an MCB T of a matroid M, a pair (e, e') of elements in E is a *swap* in T if $e \in T$, $e' \notin T$, and $T \setminus \{e\} \cup \{e'\}$ is also an MCB of M. The *label* and *excess* of a node in V can be defined exactly as in Section 3.2. A labeling $l : V \to \mathbf{N}$ is extended to a labeling on the ground set E

as follows: for $e \in E$, $l(e) = \max_{u \in e} l(u)$.

For a node $u \in V$, we let S_u^T denote the set of swaps (e, e') in T such that e is incident on u but e' in not incident on u. We call a swap (e, e') in S_u^T *useful* for u because it can be used to decrease the degree of u. We say a swap (e, e') in T is *feasible* for a labeling l if $l(e) \leq l(e') + 1$. As in Section 3.2, a labeling $l : V \to \mathbf{N}$ is defined to be *feasible* for an MCB T if for all nodes $u \in V$, for all swaps $(e, e') \in S_u^T$, $l(e) \leq l(e') + 1$. Given a labeling l and an MCB T, a swap $(e, e') \in S_u^T$ is called *permissible* for u if $l(u) \geq l(e') + 1$.

Our PR-MDMCB algorithm is defined exactly as the push-relabel algorithm described in Figure 3.1, except that it takes as input a k-ary hypergraph $G = (V, E)$ and a weighted matroid M on ground set E, in addition to the parameter μ. To prove that the PR-MDMCB algorithm works correctly, we first show that feasibility is maintained in any iteration of the algorithm.

3.6.3 Feasibility

Lemma 3.17. *The* PR-MDMCB *algorithm always maintains a feasible labeling.*

Proof. As in Lemma 3.1, we show this by induction on the number of iterations in a phase. In the beginning of any phase of the algorithm, all nodes and hence all edges e have label 0, which is a feasible labeling. In one step of the algorithm, we either update a label or execute a permissible swap (e, e'). Since we increment the label of a node only when it has no permissible swaps, feasibility is maintained in the first case.

To prove that feasibility is maintained when a permissible swap is executed, we make use of the *rank* function $r : 2^E \to \mathbf{N}$ of the matroid M. For a subset $E' \subseteq \mathbf{E}$, the rank $r(E')$ is defined to be $\max_{F \subseteq E' : F \in \mathcal{I}} |F|$. For any subsets $A, B \subseteq E$, the rank function r satisfies (i) $r(A) \leq |A|$, (ii) $r(A) \leq r(B)$ if $A \subseteq B$, and (iii)

$r(A \cup B) + r(A \cap B) \leq r(A) + r(B)$. Note that A is a base of M if and only if $r(A) = r(M)$.

Let T be the current MCB after executing a sequence of swaps, so that the labeling l is feasible for T. Let (e, e') be the permissible swap executed in the current iteration, and let (f, f') be a swap in $T' = T \setminus \{e\} \cup \{e'\}$. If (f, f') is a swap in T, then by the inductive condition, $l(f) \leq l(f') + 1$.

Thus it remains to consider a pair (f, f') that is a swap in T' but not in T. There are three cases:

- *$f' \in T$ and hence not available for the swap:* In this case, $e = f'$. We observe that $T \setminus \{f\} \cup \{e'\} = (T \setminus \{e\} \cup \{e'\}) \setminus \{f\} \cup \{f'\} = T' \setminus \{f\} \cup \{f'\}$, which is an MCB of M since (f, f') is a swap in T'. Thus the pair (f, e') is a swap in T. By feasibility of (f, e') and permissibility of (e, e'), we conclude that $l(f) \leq l(e') + 1 = l(e) = l(f')$.

- *$f \notin T$ and hence not available for the swap:* In this case $e' = f$. We observe that $T \setminus \{e\} \cup \{f'\} = (T \setminus \{e\} \cup \{e'\}) \setminus \{f\} \cup \{f'\} = T' \setminus \{f\} \cup \{f'\}$, which is an MCB of M since (f, f') is a swap in T'. Thus the pair (e, f') is a swap in T. By permissibility of (e, e') and feasibility of (e, f'), we conclude that $l(f) = l(e') = l(e) - 1 \leq l(f')$.

- *$f \in T$ and $f' \notin T$:* Note that since (f, f') is not a swap in T but is a swap in T', $r(T \setminus \{f\} \cup \{f'\}) = r(M) - 1$.

  We first claim that $T \setminus \{e\} \cup \{f'\}$ is a base of M. Suppose not. Then $r(T \setminus \{e\} \cup \{f'\}) = r(M) - 1$. By submodularity of the rank function, $r(T \setminus \{e, f\} \cup \{f'\}) + r(T \cup \{f'\}) \leq r(T \setminus \{e\} \cup \{f'\}) + r(T \setminus \{f\} \cup \{f'\})$. Thus $r(T \setminus \{e, f\} \cup \{f'\}) = r(M) - 2$ so that $r(T \setminus \{e, f\} \cup \{e', f'\}) \leq r(M) - 1$, which contradicts that fact that (f, f') is a swap in T'.

  We now argue that $T \setminus \{f\} \cup \{e'\}$ is also a base of M. Suppose not. Then

$r(T \setminus \{f\} \cup \{e'\}) = r(M) - 1$. Once again submodularity tells us that $r(T \setminus \{f\} \cup \{e', f'\}) + r(T \setminus \{f\}) \leq r(T \setminus \{f\} \cup \{e'\}) + r(T \setminus \{f\} \cup \{f'\}) \leq 2r(M) - 2$. Thus $r(T \setminus \{f\} \cup \{e', f'\}) \leq r(M) - 1$. This then implies that $r(M) = r(T' \setminus \{f\} \cup \{f'\}) = r(T \setminus \{e, f\} \cup \{e', f'\}) \leq r(M) - 1$, which again contradicts the fact that (f, f') is a swap in T'.

Since T, $T \setminus \{e\} \cup \{f'\}$, and $T \setminus \{f\} \cup \{e'\}$ are all bases of M, and T is an MCB, it follows that $c(f') \geq c(e)$ and $c(e') \geq c(f)$. Since $c(e) = c(e')$ and $c(f) = c(f')$, we conclude that $c(e) = c(f)$. Thus (e, f') and (f, e') are both swaps in T. By feasibility of these swaps and permissiblity of (e, e'), we conclude that $l(f) \leq l(e') + 1 = l(e) \leq l(f') + 1$.

We have shown that, in all cases, the swap (f, f') is feasible. Hence the induction holds. □

3.6.4 The witness

The concept of the witness generalizes to the MDMCB problem. We define a *witness* to be a pair (F, X) where $F \subseteq E$ is a subset of some MCB and $X \subseteq V$ has the property that in any MCB T of M containing F, each edge in $T \setminus F$ is incident on X in G. The following lemma is analogous to Lemma 3.2 and follows from essentially the same arguments.

Lemma 3.18. *Let $\mathcal{W} = (F, X)$ be a witness as defined above for a hypergraph G and a matroid M. Then any (fractional) MCB of M has maximum degree at least $\frac{r(M)-|F|}{|X|}$ in G.*

3.6.5 A constant-factor approximation

We now give generalizations of other results from Sections 3.2 and 3.3. Generalizations of Lemmas 3.3, 3.4, 3.5, and 3.8 are immediate. The observation that each swap is a

parent of at most k swaps leads to Lemma 3.19, which is an analogue of Lemma 3.6. Corollary 3.20 is analogous to Corollary 3.7.

Lemma 3.19. *A label-p cascade contains at most k^{p-q} label-q swaps.*

Corollary 3.20. *A label-p cascade contains at most $(k^{p-q+1}-1)$ involuntary losses to nodes with labels at least q.*

Proof. From Lemma 3.19, the total number of swaps with label at least q in a label-p cascade is at most $\frac{k^{p-q+1}-1}{k-1}$. The corollary follows from the fact that each swap causes at most $(k-1)$ involuntary losses. □

Lemma 3.21. *The number of involuntary losses to nodes in W_{p^*+1} is at most*

$$k|W_{p^*+1}|(\frac{\mu}{\mu-k}).$$

The proof of Lemma 3.21 is analogous to the proof of Lemma 3.9.

We conclude with an approximation guarantee for the matroid variant of the PR-MDMST algorithm:

Theorem 3.22. *Given a k-ary hypergraph $G=(V,E)$, a weighted matroid $M=(E,\mathcal{I},c)$, and a real-valued parameter $\mu > k$, the* PR-MDMCB *algorithm obtains in polynomial time an MCB of degree Δ, where $\Delta \leq k\mu\,\Delta_{\text{opt}}(G,M)+1+\frac{k\mu}{\mu-k}$.*

Proof. On input(G,M,μ), the PR-MDMST algorithm terminates with a maximum independent subset T and a pair $\mathcal{W}^{p^*}=(F,X)$. We now compute the number $|T[W_{p^*+1}]|$ of edges in T incident on W_{p^*+1}. Each node in W_{p^*+1} has degree at least $(\Delta-1)$ after its last voluntary loss, and it may then suffer some involuntary losses. Using the bound from Lemma 3.21, the total loss in degree to W_{p^*+1} from involuntary losses is $\frac{k\mu}{\mu-k}\,|W_{p^*+1}|$. The sum of degrees in T of nodes in W_{p^*+1} is therefore at least $(\Delta-1-\frac{k\mu}{\mu-k})\,|W_{p^*+1}|$. Since each edge is incident on at most k nodes, the number

$|T[W_{p^*+1}]|$ of edges in T incident on W_{p^*+1} is at least $(\frac{1}{k})$ times this sum of degrees. Thus from the generalization of Lemma 3.4,

$$\Delta_{\text{opt}}(G, M) \geq \left(\Delta - 1 - \frac{k\mu}{\mu - k}\right) \frac{1}{k\mu}.$$

Rearranging, we get

$$\Delta \leq k\mu\, \Delta_{\text{opt}}(G, M) + 1 + \frac{k\mu}{\mu - k}$$

□

Remark: The bound above does not strictly generalize the bound in Theorem 3.10 since we do not have the structure necessary the bound the number of edges internal to W_{p^*+1} as we did in the proof of Theorem 3.10. Instead we use a cruder argument to lower bound the cardinality of $T[W_{p^*+1}]$.

Choosing $\mu = k + \sqrt{\frac{k}{\Delta_{\text{opt}}(G,M)}}$, we get the following bound.

Corollary 3.23. *Given a hypergraph G and a matroid M, there is a polynomial-time algorithm that outputs an MCB of degree Δ, where*

$$\Delta \leq k^2 \Delta_{\text{opt}}(G, M) + 2k^{\frac{3}{2}} \sqrt{\Delta_{\text{opt}}(G, M)} + k + 1.$$

3.6.6 A density-based bound

In this section, we generalize the result of Section 3.4.1. We first define a notion of density for a hypergraph. Given a hypergraph $G = (V, E)$ and a subset $U \subseteq V$, we define the density of U to be the ratio $\frac{\sum_{e \in E[U]} (|e \cap U| - 1)}{|U|}$. In other words, each edge e incident on U contributes $|e \cap U| - 1$ to the numerator. The *local density* $s(G)$ of a hypergraph G is the defined to be the maximum density of any subset of V, i.e. $s(G) = \max_{U \subseteq V} \frac{\sum_{e \in E[U]} (|e \cap U| - 1)}{|U|}$.

Let T be the tree and $\mathcal{W}^{p^*}$ the witness output by the PR-MDMCB algorithm on input (G, M) and $\mu = 1$. Let Δ be the degree of T. Since $\mu = 1$, the sets W_{p^*} and W_{p^*+1} must be equal, and hence, level p^* is empty when the algorithm terminates. The following lemma is an analogue of Lemma 3.12.

Lemma 3.24. *Let T be the MCB, l the labeling, and p^* the empty level when the algorithm* PR-MDMCB *terminates. Then the number of edges in T that are incident on W_{p^*+1} is at least $(\Delta - 1)\,|W_{p^*+1}| + 1 - \sum_{u \in E[W_{p^*+1}]}(|e \cap W_{p^*+1}| - 1)$.*

Proof. For a node $u \in W_{p^*+1}$, let $T[u] \subset T$ be the set of edges in T incident on u, and let $L[u] \subset E$ be the set of edges that node u loses involuntarily after its last voluntary loss. Since each node has degree at least $\Delta - 1$ after its last voluntary loss, it follows that $|T[u]| \geq \Delta - 1 - |L[u] \setminus T[u]|$. Moreover, if v is the last node to be relabeled, $|T[v]| \geq \Delta$. Thus the sum of degrees of nodes in W_{p^*+1} in T is at least $(\Delta - 1)\,|W_{p^*+1}| + 1 - \sum_{u \in W_{p^*+1}} |L[u] \setminus T[u]|$.

An edge e may be in $L[u]$ for at most $|e \cap W_{p^*+1}| - 1$ nodes $u \in e$. It follows that

$$\sum_{u \in W_{p^*+1}} |L[u] \setminus T[u]| \leq \sum_{e \in \cup_{u \in W_{p^*+1}} (L[u] \setminus T[u])} |e \cap W_{p^*+1}| - 1.$$

Moreover, $\cup_{u \in W_{p^*+1}} (L[u] \setminus T[u])$ is a subset of $E[W_{p^*+1}] \setminus T[W_{p^*+1}]$.

An edge e in $T[W_{p^*+1}]$ contributes to $T[u]$ for each endpoint $u \in e \cap W_{p^*+1}$, and thus is counted $|e \cap W_{p^*+1}|$ times in the sum of degrees of nodes in W_{p^*+1} in T. Thus

the number $|T[W_{p^*+1}]|$ of edges in T incident on W_{p^*+1} is

$$\begin{aligned}
&\sum_{u \in W_{p^*+1}} |T[u]| - \sum_{e \in T(W_{p^*+1})} (|e \cap W_{p^*+1}| - 1) \\
&\geq (\Delta - 1)\,|W_{p^*+1}| + 1 - \sum_{u \in W_{p^*+1}} |L[u] \setminus T[u]| - \sum_{e \in T(W_{p^*+1})} (|e \cap W_{p^*+1}| - 1) \\
&\geq (\Delta - 1)\,|W_{p^*+1}| + 1 - \sum_{u \in E[W_{p^*+1}] \setminus T[W_{p^*+1}]} (|e \cap W_{p^*+1}| - 1) - \sum_{e \in T(W_{p^*+1})} (|e \cap W_{p^*+1}| - 1) \\
&= (\Delta - 1)\,|W_{p^*+1}| + 1 - \sum_{u \in E[W_{p^*+1}]} (|e \cap W_{p^*+1}| - 1)
\end{aligned}$$

□

We now use the above bound on $|T[W_{p^*+1}]|$ to prove a lower bound on $\Delta_{\text{opt}}(G, M)$.

Theorem 3.25. *Let T and $\mathcal{W}^{p^*}$ be the output of the* PR-MDMCB *algorithm given a hypergraph G, a matroid M, and $\mu = 1$. Then the degree Δ of T is at most $\Delta_{\text{opt}}(G, M) + \lceil\, s(G) \,\rceil$.*

Proof. From Lemma 3.24, the number of edges in T incident on W_{p^*+1} is at least $(\Delta - 1)\,|W_{p^*+1}| + 1 - \sum_{u \in E[W_{p^*+1}]}(|e \cap W_{p^*+1}| - 1)$. By the definition of local density, the ratio $\frac{\sum_{u \in E[W_{p^*+1}]}(|e \cap W_{p^*+1}| - 1)}{|W_{p^*+1}|}$ is at most $s(G)$. An analogue of Lemma 3.4 then implies that $\Delta_{\text{opt}}(G, M) \geq \Delta - s(G) - 1 + \frac{1}{|W_{p^*+1}|}$. Since $\Delta_{\text{opt}}(G, M)$ is an integer, we conclude that $\Delta_{\text{opt}}(G, M) \geq \Delta - \lceil\, s(G) \,\rceil$. □

We note that we are not aware of an analogue of Theorem 3.14 for the MDMCB problem. Such an analogue would imply an additive approximation via Theorem 3.25.

Chapter 4

MSTs with degree bounds

In Chapter 2, we show that the problem of finding an MBDST can be reduced to the problem of finding an actual MST, under a different cost function, that satisfies upper and lower bounds on the degrees of nodes. This secondary problem is the main subject of this chapter.

We begin by giving a formal definition of the Minimum Spanning Tree with Degree Bounds (MSTDB) problem, as well as some necessary notation. In Section 4.3, we describe polynomial-time algorithms that enforce upper bounds and lower bounds separately; the algorithms are then carefully combined to obtain a polynomial-time algorithm for the MSTDB problem. Section 4.4 presents the corresponding result for the MBDST problem. In Sections 4.5, we show how to replace the sub-routines of Section 4.3 with algorithms based on augmenting paths that achieve much better approximations in degree but may take quasi-polynomial time. The corresponding result for the MBDST problem appears in Section 4.6.

4.1 The MSTDB problem

In Chapter 2, Section 2.6, we show that a true approximation algorithm for the MBDST problem can be obtained from an algorithm that finds an MST that meets upper and lower bounds on degree. Hence, we formally define the Minimum Spanning Tree with Degree Bounds (MSTDB) problem as follows:

MSTDB Problem: Given a weighted graph $G = (V, E, c)$, a degree upper bound B_H, a subset of vertices $L \subseteq V$, and a degree lower bound B_L, find, if one exists, a minimum spanning tree of G such that no vertex has degree more than B_H and all vertices in L have degree at least B_L.

Note that an MST with these degree bounds may not exist in the graph. In this case, we would like an algorithmic solution to provide a proof of non-existence.

4.2 Additional notation

For a tree T, let $d^L_{\min}(T) = \min_{v \in L} \deg_T(v)$. For symmetry of notation, we let $d_{\max}(T) = \Delta(T) = \max_{v \in V} \deg_T(v)$. For an integer $d > 0$ and a tree T, let $S_{\geq d}$ be the set of nodes with degree at least d in T, and let $S^L_{\leq d}$ be the subset of nodes in L that have degree at most d.

For a tree T and a cluster $C \subseteq V$ of nodes, we say C is *internally connected* in T if the induced subgraph (C, T_C) is connected, where $T_C = \{(u, v) \in T : u, v \in C\}$.

4.3 MSTDB: A Polynomial-time algorithm

Our polynomial-time algorithm MstdbP for the MSTDB problem is based on Fischer's algorithm [14] for finding a minimum-degree MST of a graph. Our algorithm finds an

MST such that (a) every vertex has degree at most $bB_H + \log_b n$ and (b) all vertices in L have degree at least $B_L/b - \log_b n$. If it fails, it finds a combinatorial witness to show that there exists no MST of the graph in which all vertices have degree at most B_H and the vertices in L have degree at least B_L.

Starting with an arbitrary MST, our MstdbP algorithm proceeds in phases of improvement. Each phase is essentially a combination of the two algorithms MaxdmstP and MindmstP, which are described in Sections 4.3.1 and 4.3.2. The MaxdmstP algorithm is essentially Fischer's algorithm, and MindmstP is a somewhat symmetric version of Fischer's algorithm that finds an MST in which a given set of nodes have maximum minimum degree.

To measure the progress made by our algorithm, we define a potential function $\phi(T)$ on the set of all MSTs. The potential function has two components: $\phi_H(T)$ and $\phi_L(T)$. The potential function $\phi_H(T)$ decreases as we move towards a tree in which more vertices satisfy a degree upper bound of B_H, and the potential function $\phi_L(T)$ decreases as we move towards a tree in which more vertices in L satisfy the degree lower bound of B_L. The potential functions are defined as follows:

$$\begin{aligned} \phi_H(T) &= \sum_v 5^{deg(v)-B_H} \\ \phi_L(T) &= \sum_{v \in L} 5^{B_L - deg(v)} \\ \phi(T) &= \phi_H(T) + \phi_L(T) \end{aligned}$$

Note that for $d > B_H$, performing a swap that decreases the number of vertices with degree at least d in a tree T decreases the potential $\phi_H(T)$, since the swap does not increase the number of vertices with degree at most $d-1$ by more than 2. Similarly, for $d < B_L$, the potential $\phi_L(T)$ decreases if a swap is performed that decreases the number of vertices in L with degree at most d, since the swap does not

Algorithm MaxdmstP(G, T, X, d)

Let $\mathcal{T} \subseteq T$ be a Steiner tree on $S_{\geq(d-1)}$.
Freeze the edges of $\mathcal{T}$ incident on $S_{\geq(d-1)}$.
if there is an (X, d)-deflating swap (e, e') where e is not frozen
then output $T \setminus \{e\} \cup \{e'\}$.

Figure 4.1: Pseudo-code for MaxdmstP.

increase the number of vertices in L with degree at least $d + 1$ by more than 2.

4.3.1 The MaxdmstP algorithm

Given an MST T of G, let $\mathcal{T} \subseteq T$ be a Steiner tree on the nodes of $S_{\geq(d-1)}$ (the Steiner nodes are the nodes in $V \setminus S_{\geq(d-1)}$). We say that we *freeze* the edges of $\mathcal{T}$ incident on $S_{\geq(d-1)}$, meaning that the edges of $\mathcal{T}$ that are incident on $S_{\geq(d-1)}$ are not allowed to be removed by any swap. For a subset $X \subseteq V$, an integer $d > 0$, and a tree T on G, a swap (e, e') is called (X, d)-*deflating* if e is incident on $S_{\geq d}$ but not on X, and e' is not incident on $S_{\geq(d-1)}$. In other words, the swap (e, e') decreases the degree of a node in $S_{\geq d}$ without increasing the degree of a node in $S_{\geq(d-1)}$ or decreasing the degree of a node in X. We look for (X, d)-deflating swaps (e, e') where e is not frozen.

Given an MST T, a set X of nodes, and an integer d, the MaxdmstP algorithm is very simple: first freeze the edges of $\mathcal{T}$ incident on $S_{\geq(d-1)}$ as above. Find and execute an (X, d)-deflating swap (e, e') where e is not frozen, if it exists. See Figure 4.1 for a formal description.

When the MaxdmstP and MindmstP algorithms are combined later, X corresponds to the set of low-degree nodes. Since we do not want the low-degree nodes to lose degree, we disallow deletion of edges incident on them. The Steiner tree $\mathcal{T}$ is frozen to help create the witness when the algorithm fails to find a swap.

To show that the MaxdmstP algorithm works as promised, we need to use a some-

what more complicated witness than the one described in Chapter 2, Section 2.5. A witness $\mathcal{W}$ consists of a partition of the nodes into a center set U and clusters $C_1, C_2, \ldots, C_k$, a set $W \supseteq U$ of nodes, and a set F of frozen edges. The witness $\mathcal{W}$ must have the property that any MST that includes the edges in F must contain at least one edge from each C_i to W. Thus this witness is similar to the witness described in Chapter 2 except for the set F of frozen edges. See Figure 4.2.

Intuitively, W is a set of high-degree vertices, and U is a set of "incurably" high-degree nodes; i.e. their degree cannot be decreased without increasing the degree of W. The set F of frozen edges is useful technically in the proofs. If the size of F is small and k is large, then any MST must use a large number of edges incident on W.

Lemma 4.1 (Witness to high optimal degree)**.** *A* high-degree witness $\mathcal{W} = (\{U, C_1, C_2, \ldots, C_k\}, W, F)$ *certifies that any fractional MST* τ *of* G *has maximum degree at least*

$$\left(\frac{k - 2\,|F|}{|W|}\right)$$

Proof. First we restrict ourselves to MSTs that contain all the edges in F. The proof in this case is identical to that of Lemma 2.1 except for the following difference: Unlike the witness in Lemma 2.1, W may not be disjoint from the C_i's. However any edge that simultaneously connects C_i and C_j to W (i.e. an edge from $C_i \cap W$ to $C_j \cap W$) contributes two units of degree to W. The bound in Lemma 2.1 therefore continues to hold.

In any MST that does not contain all the edges of F, the total degree of W can be smaller by at most $2\,|F|$. The average degree of W is therefore at least $(k - 2\,|F|)/\,|W|$ in any MST of G. Since a fractional MST is a convex combination of MSTs, the average degree bound also holds for any fractional MST. □

If the MaxdmstP algorithm fails to find a swap, we can create a witness as follows.

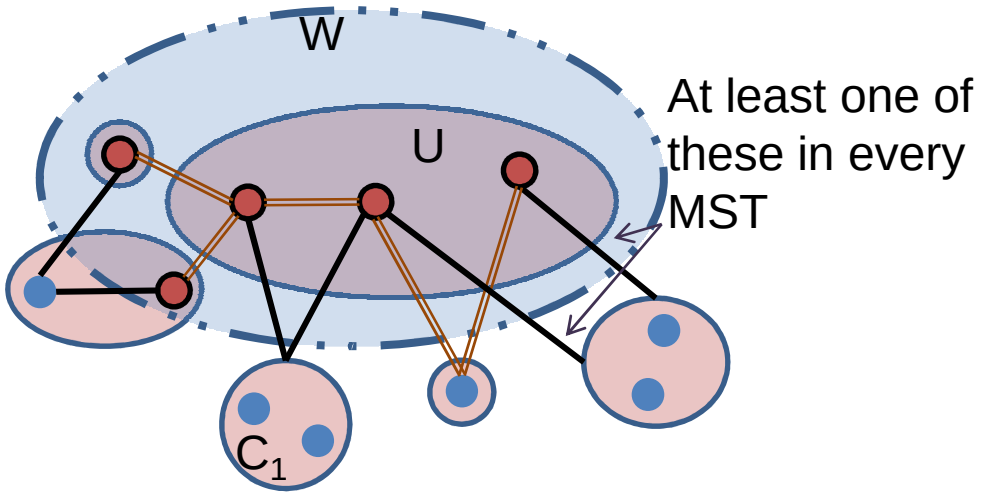

Figure 4.2: Example of a high-degree witness, as used in Lemma 4.1. The double edges are frozen.

Let $U = S_{\geq d}$ be the center set. Let $C_1, C_2, \ldots, C_k$ be the connected components obtained by deleting the nodes in U from the current MST T. We set F to be the union of the set of edges of $\mathcal{T}$ incident on $S_{\geq(d-1)}$ and the set of edges in T belonging to $X \times S_{\geq d}$. Finally W is the set $S_{\geq(d-1)}$. Theorem 4.2 calculates the guarantees offered by this witness. With some foresight, we assume that X is chosen appropriately.

Theorem 4.2 ([14])**.** *Suppose we are given as input an MST T, an integer d and a set of nodes X such that $X = S^L_{\leq(B_L+B_H-d+1)}$. When the algorithm* MaxdmstP *is called on this input, it either outputs a tree with potential function value at most $\phi(T) - 5^{d-B_H-1}$, or finds a witness $\mathcal{W}$ that certifies that for any fractional MST τ of G,*

$$d_{\max}(\tau) \geq (d-3)\frac{|S_{\geq d}|}{|S_{\geq(d-1)}|} - \frac{2\,|X|}{|S_{\geq(d-1)}|} - 4.$$

Proof. The first part of the theorem holds if there is an (X, d)-deflating swap in T. Executing the swap reduces the degree of at least one node in $S_{\geq d}$ and does not increase the degree of any node in $S_{\geq(d-1)}$. Hence $\phi_H(T)$ decreases by at least $3(5^{d-B_H-1})$. Since a swap decreases the degree of at most two nodes, both outside $X = S^L_{\leq(B_L+B_H-d+1)}$, $\phi_L(T)$ increases by at most $2(5^{B_L-(B_L+B_H-d+1)}) = 2(5^{d-B_H-1})$.

The claim follows.

Otherwise let $\mathcal{W} = (\{U, C_1, \ldots, C_k\}, W, F)$ be defined as above. We first argue that $\mathcal{W}$ is a witness. Suppose otherwise, i.e. that there is a tree T' containing F that has no edge going from C_i to W. The proof will show that the existence of such a T' contradicts the failure to find an (X, d)-deflating swap.

Let T_i be the set of edges in T from U to C_i. We first argue that $|T_i \setminus F| \leq 1$. By definition of C_i, it is a connected component of $T \setminus T_i$ and is therefore internally connected in $T \setminus (T_i \setminus F)$. Moreover, $T \setminus (T_i \setminus F)$ contains $\mathcal{T}$ so that U is connected in $T \setminus (T_i \setminus F)$. Hence if there were two edges in $(T_i \setminus F)$ connecting C_i to U, T would have a cycle.

Thus $|T_i \setminus F| \leq 1$. Since T' contains F but has no edges from $U \subseteq W$ to C_i, the set $T_i \cap F$ is empty and T_i contains a single edge, say e_i. The discussion above implies that C_i is a component of $T \setminus \{e_i\}$. By the exchange property, $T \setminus \{e_i\}$ can be completed from T'; that is, there is an edge $e_i' \in T'$ such that (e_i, e_i') is a swap with respect to T. Since e_i' must be incident on C_i, it is not incident on W and thus (e_i, e_i') must be an (X, d)-deflating swap. Since no such swaps exist for T, we derive a contradiction.

Finally, note that

$$|F| \leq 2\left|S_{\geq(d-1)}\right| + (|S_{\geq d}| + |X| - 1),$$

since the average degree of $S_{\geq(d-1)}$ in the Steiner tree $\mathcal{T}$ is at most two, and the second constituent of F is a forest on $S_{\geq d} \cup X$. Moreover, $k \geq (d-2)\,|S_{\geq d}|$, since each edge from $S_{\geq d}$ to $V \setminus S_{\geq d}$ deleted during the construction of $\mathcal{W}$ contributes an additional component, and the average external degree of $S_{\geq d}$ is at least $(d-2)$. The claim follows from Lemma 4.1. □

Algorithm MindmstP(G, T, L, Y, d)

if there is a (Y, d)-inflating swap (e, e')
then output $T \setminus \{e\} \cup \{e'\}$
else Create Witness.

Figure 4.3: Pseudo-code for MindmstP.

4.3.2 The MindmstP algorithm

Here we give an algorithm for the nearly symmetric problem of finding an MST that respects degree lower bounds for a subset L of the vertices.

For an MST T, subsets $L, Y \subseteq V$ of vertices, and an integer $d > 0$, a swap (e, e') is called (Y, d)-*inflating* if e' is incident on $S^L_{\leq d}$ but not on Y and e is not incident on $S^L_{\leq (d+1)}$. That is, a (Y, d)-inflating swap is one that increases the degree of a node in $S^L_{\leq d}$ without increasing the degree of a node in Y or decreasing the degree of a node in $S^L_{\leq (d+1)}$.

The MindmstP algorithm is very natural: given $Y \subseteq V$, $d > 0$, and an MST T, execute a (Y, d)-inflating swap, if one exists. See Figure 4.3 for a formal description.

We define the witness $\mathcal{W}_{L,P}$ provided by the MindmstP algorithm as follows. Let V be partitioned into sets $U, C_1, C_2, \ldots, C_k$. Let W be a subset of U, and let $Y \subset V$ be disjoint from U. Let $R \subseteq (Y \times W)$ be a subset of the edges between Y and W. Finally, let H be another set of edges incident on W. A witness $\mathcal{W}_{L,P} = (\{U, C_1, \ldots, C_k\}, W, Y, R, H)$ is a decomposition of the graph such that in any MST T that contains the edges in H and excludes the edges in R, for each i, $1 \leq i \leq k$, there is at most one edge in T between C_i and W that does not belong to H. See Figure 4.4 for an illustration.

The witness is in spirit very similar to the high-degree witness. Intuitively, U is a set of low degree vertices, and W is a set of "incurably" low-degree nodes in L, i.e. their degree cannot be increased without taking degree away from U. The set Y

can be thought of as a set of high-degree nodes and R as a set of edges from high-degree nodes to low-degree nodes; these constituents of the witness are needed later when the MindmstP algorithm is combined with the MaxdmstP algorithm to prevent the MindmstP algorithm from raising the degree of high-degree nodes. The set H of edges is useful technically in the proofs to ensure certain connectivity properties within W.

Lemma 4.3. *Given a graph $G = (V, E)$, a low-degree witness*

$$\mathcal{W}_{L,P} = (\{U, C_1, \ldots, C_k\}, W, Y, R, H)$$

certifies that in every fractional MST τ of G,

$$d^L_{\min}(\tau) \leq \frac{k + |U| + 2\,|W| + |Y| + |H| - 2}{|W|}.$$

Proof. First let us consider an integral MST T such that T contains H and excludes R. The existence of the witness $\mathcal{W}_{L,P}$ guarantees that for each i, $1 \leq i \leq k$, T has at most one edge between C_i and W that is not in H. This implies that W cannot have total degree in T more than $k + |U| + |W| - 2$, since of the $(k + |U| - 1)$ edges incident on W, at most $|W| - 1$ can have both endpoints in W. In an arbitrary integral MST, the total degree of W cannot be improved to more than

$$k + |U| + 2\,|W| + |Y| + |H| - 2$$

since one could replace the $|H|$ edges in H, and add at most $|Y| + |W| - 1$ of the edges from R. Since a fractional MST is just a convex combination of integral MSTs, the upper bound on the total degree of W remains the same, and the claim follows. □

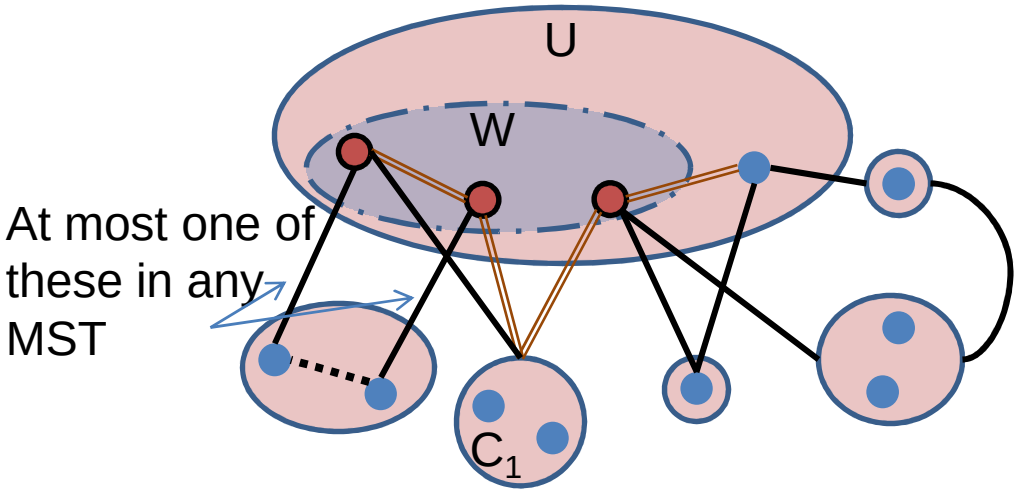

Figure 4.4: Example of a low-degree witness, as used in Lemma 4.3. The double-edges are in H. (R and Y are not shown.)

If the MindmstP algorithm fails to make a swap, we create a witness as follows. Let $U = S^L_{\leq(d+1)}$, let $W = S^L_{\leq d}$, and let $C_1, \ldots, C_k$ be the components that remain when U is removed from T. Choose a Steiner tree $\mathcal{T}$ on U such that $\mathcal{T} \subseteq T$, and let H be the set of edges in $\mathcal{T}$ that are incident on U. Let R be the set of edges of G between Y and W that are *not* in T, that is, $R = (Y \times W) \setminus T$. Let $\mathcal{W}_{L,P} = (\{U, C_1, \ldots, C_k\}, W, Y, R, H)$. The following theorem quantifies the quality of the witness, when the algorithm produces one. With some foresight, we assume the set Y is chosen appropriately.

Theorem 4.4. *Suppose we are given as input a graph $G = (V, E)$, an MST T, subsets $L, Y \subseteq V$ and an integer $d > 0$. Then the algorithm* MindmstP *when called with $Y = S_{\geq(B_L+B_H-d-1)}$, either produces a tree T with potential function value at most $\phi(T) - 5^{B_L-d-1}$, or finds a witness $\mathcal{W}_{L,P}$ that certifies that for any fractional MST τ of G*

$$d^L_{\min}(\tau) \leq (d+3)\frac{\left|S^L_{\leq(d+1)}\right|}{\left|S^L_{\leq d}\right|} + \frac{|Y|}{\left|S^L_{\leq d}\right|} + 2.$$

Proof. In the case that the algorithm finds a (Y, d)-inflating swap, we increase the degree of some node in $S^L_{\leq d}$, and affect the degree of at most three other nodes by one. Moreover, we do not decrease the degree of any node in $S^L_{\leq (d+1)}$. It follows that the potential function decreases by the claimed amount.

Suppose then that the algorithm fails to find a swap. Let T be the tree for which the algorithm creates a witness. We show first that

$$\mathcal{W}_{L,P} = (\{U, C_1, \ldots, C_k\}, W, Y, R, H)$$

defined as above is a witness. Assume otherwise: that is, there is some MST T' containing H and excluding R such that for some i, $1 \leq i \leq k$, there are two edges e'_1, e'_2 between C_i and W in T' that do not belong to H. (Note that C_i need not be a connected component in T'.) At least one of these two edges, say e'_2, is not in T. By the construction of the clusters, T has at most one edge, say e_i, that is not in H and that is between W and C_i.

The proof argues that the existence of e'_1 and e'_2 in T' contradicts the failure to find a (Y, d)-inflating swap. We consider two cases:

Case 1: $e'_1 \in T$, i.e. $e_i = e'_1$. Since T and T' are both MSTs and $e'_2 \notin T$, there is an edge $e_2 \in T$ different from e'_1 such that (e_2, e'_2) is a swap with respect to T. Since (e_2, e'_2) is not (Y, d)-inflating, e_2 must be incident on $U = S^L_{\leq (d+1)}$. Consider $T_2 = T \setminus \{e_2\} \cup \{e'_2\}$. Since e_2 is not internal to C_i, C_i is still internally connected in T_2. Let u'_1 and u'_2 be the endpoints of e'_1 and e'_2, respectively, in W. There is a simple path in T_2 from u'_1 to u'_2 that contains the edges e'_1 and e'_2. Since $u'_1, u'_2 \in W \subseteq U$, there is a path in $\mathcal{T}$ from u'_1 to u'_2. This path also exists in T_2, since $e_2 \notin H$ is incident on U, while all Steiner tree edges incident on U are in H. Thus there are two distinct simple paths from u'_1 to u'_2 in T_2, contradicting the acyclicity of T_2.

Case 2: $e'_1 \notin T$. This case is similar, except that we have to do a little more work

to get to T_2. Since e'_1 exists in T' but not in T, there is an edge $e_1 \in T$ such that (e_1, e'_1) is a swap with respect to T. The lack of (Y, d)-inflating swaps for T implies that e_1 must be incident on U. Let $T_1 = T \setminus \{e_1\} \cup \{e'_1\}$. The edge e'_2 can be swapped in for some edge $e_2 \neq e'_1$ in T_1. We next argue that e_2 is incident on U. Assume the contrary. Then (e_2, e'_2) is a (Y, d)-inflating swap with respect to T_1. The algorithm's failure to find a (Y, d)-inflating swap implies that (e_2, e'_2) is not a swap with respect to T. However, a simple structural argument then shows that (e_2, e'_1) is a swap with respect to T. Moreover, it is (Y, d)-inflating, which contradicts the failure to find a (Y, d)-inflating swap. Thus, both e_1 and e_2 are incident on U and not internal to C_i. We conclude that C_i is internally connected in T_2, which leads to a contradiction by an argument analogous to Case 1.

The only thing that remains is to prove the quality of the witness $\mathcal{W}_{L,P}$. Since the components $C_1, \ldots, C_k$ are created by removing $S^L_{\leq(d+1)}$ from T, we have

$$k \leq (d+1) \left| S^L_{\leq(d+1)} \right|.$$

Since the edges in H are in a Steiner tree and incident on $S^L_{\leq(d+1)}$, there are at most $\left| S^L_{\leq(d+1)} \right| - 1$ of them. Lemma 4.3 then implies the claim. □

4.3.3 The MstdbP algorithm

Now we describe our MstdbP algorithm more completely. Recall that our MstdbP algorithm works in phases. Each phase employs algorithms MaxdmstP and MindmstP to improve either a high-degree vertex or a low-degree vertex in L; when both improvements fail, their failure is justified by two combinatorial witnesses. See Figure 4.5 for a formal description.

Let us now look into a phase of the algorithm in more detail. Each phase of MstdbP begins by picking a δ such that

$$\left|S^L_{\leq(B_L-\delta+1)}\right| \leq b\left|S^L_{\leq(B_L-\delta)}\right| \qquad \text{and} \qquad \left|S_{\geq(B_H+\delta-1)}\right| \leq b\left|S_{\geq(B_H+\delta)}\right|$$

where $b > 1$ is an input parameter. It is easy to show that one can always find such a δ in any range of size at least $2\log_b n$, and hence in particular, between

$$\max\{d_{\max}(T) - B_H, B_L - d^L_{\min}(T)\} - 2\log_b n$$

and

$$\max\{d_{\max}(T) - B_H, B_L - d^L_{\min}(T)\}.$$

If $\delta < 0$, we are done; we assume $\delta > 0$ for the rest of this section.

For the rest of the phase, vertices with degree at least $B_H + \delta$ are considered "high-degree" vertices and those in L with degree at most $B_L - \delta$ are considered "low-degree". We employ MaxdmstP and MindmstP to reduce the degree of a high-degree vertex and increase the degree of a low-degree vertex respectively. As alluded to earlier, we need to ensure that the improvements they perform do not interfere. For this purpose, we disallow the MaxdmstP algorithm from removing edges incident on $X = S^L_{\leq(B_L-\delta+1)}$, and disallow the MindmstP algorithm from adding edges to $Y = S_{\geq(B_H+\delta-1)}$. Each subroutine either improves the potential significantly or returns a combinatorial witness. Each phase runs these two subroutines. The algorithm terminates if one of two things happen: the algorithm finds an MST with the required degree guarantees, or both subroutines output combinatorial witnesses on a particular tree T.

Lemma 4.5 guarantees that the algorithm terminates quickly. Theorem 4.6 shows that when both MaxdmstP and MindmstP fail with two combinatorial witnesses, at least one of the witnesses is good.

Lemma 4.5. *Algorithm* MstdbP *terminates in polynomial time.*

Algorithm MstdbP(G, B_L, B_H, L, b)

Start with arbitrary minimum spanning tree T.
repeat
  Compute δ so that
    $\left|S^L_{\leq(B_L-\delta+1)}\right| \leq b\left|S^L_{\leq(B_L-\delta)}\right|$ and $\left|S_{\geq(B_H+\delta-1)}\right| \leq b\left|S_{\geq(B_H+\delta)}\right|$.
  Call MaxdmstP with $d = B_H + \delta$ and $X = S^L_{\leq(B_L-\delta+1)}$.
  Call MindmstP with $d = B_L - \delta$ and $Y = S_{\geq(B_H+\delta-1)}$.
until both calls fail.

Figure 4.5: Pseudo-code for MstdbP.

Proof. Let T be the tree at the beginning of a particular phase and T' be the tree at the end of the phase. At the end of this phase, at least one of the following is true:

(i) T' has a potential function value at most $\phi(T) - 5^{\delta-1}$.

(ii) MaxdmstP and MindmstP both output combinatorial witnesses and the algorithm terminates.

The first step happens when one or both of MaxdmstP and MindmstP succeed; in this case Theorems 4.2 and 4.4 imply the claimed decrease in potential. Moreover, note that

$$\delta \geq \max\{d_{\max}(T) - B_H, B_L - d^L_{\min}(T)\} - 2\log_b n$$

so that $d_{\max}(T) - B_H \leq \delta + 2\log_b n$ and $B_L - d^L_{\min}(T) \leq \delta + 2\log_b n$. The total potential is then

$$\phi(T) \leq n \cdot 5^{1+2\log_b n} 5^{\delta-1}.$$

Thus, the decrease in potential is at least $\phi(T)/(n \cdot 5^{1+2\log_b n})$ in each phase, and the potential function decreases by half after $(n \cdot 5^{1+2\log_b n})$ phases. Since each node contributes at most $2 \cdot 5^n$ to the initial potential, the initial potential is at most exponential in n. The algorithm therefore terminates in polynomial time. □

4.4 MBDST: A polynomial-time algorithm

Theorem 4.6. *Given a gragh G, a degree bound B, and $b \in (1,2)$, there is a polynomial time algorithm that computes a spanning tree T with cost at most $c_{\mathrm{opt}_B}(G)$ and degree at most $\frac{b}{2-b}B + O(\log_b n)$.*

Proof. For a fixed B', suppose that we compute $c^{\lambda^{B'}}$ and set L to be the set of nodes with a positive λ_u. Thus we have a fractional spanning tree τ that has maximum degree $d_{\max}(\tau)$ at most B' and $d^L_{\min}(\tau)$ at least B'. Executing $\mathsf{MstdbP}(G, B', B', L, b)$ on this cost function produces a spanning tree T and witnesses that certify that

$$B' \geq d_{\max}(\tau) \geq \frac{B' + \delta - 3}{b} - 2\frac{\left|S^L_{\leq(B'-\delta+1)}\right|}{\left|S_{\geq(B'+\delta-1)}\right|} - 4$$

and

$$B' \leq d^L_{\min}(\tau) \leq b(B' - \delta + 3) + \frac{\left|S_{\geq(B'+\delta-1)}\right|}{\left|S^L_{\leq(B'-\delta)}\right|} + 2.$$

By our choice of δ, at least one of the following inequalities holds:

$$2b\frac{\left|S^L_{\leq(B'-\delta+1)}\right|}{\left|S_{\geq(B'+\delta-1)}\right|} < 2b$$

and

$$\frac{\left|S_{\geq(B'+\delta-1)}\right|}{\left|S^L_{\leq(B'-\delta)}\right|} < 2b.$$

This is because the product of the terms on the left of each inequality is at most $2b^2 < 4b^2$. In either case, this implies that

$$\delta \leq (b-1)B' + 5 + 6b.$$

However,

$$\delta \geq \max\{d_{\max}(T) - B', B' - d^L_{\min}(T)\} - 2\log_b n,$$

so that

$$d_{\max}(T) \leq bB' + 5 + 6b + 2\log_b n$$

and

$$d^L_{\min}(T) \geq (2-b)B' - 5 - 6b - 2\log_b n.$$

For $1 < b < 2$ and

$$B = (2-b)B' - 5 - 6b - 2\log_b n,$$

which means that we have computed a tree T such that $d^L_{\min}(T) \geq B$ and

$$d_{\max}(T) \leq \frac{b}{2-b}B + \frac{2}{2-b}(5 + 6b + 2\log_b n).$$

Theorem 2.3 implies T has cost at most $c_{\mathrm{opt}_B}(G)$ and degree at most

$$\frac{b}{2-b}B + O(\log_b n).$$

□

4.5 MSTDB: A quasi-polynomial-time algorithm

In this section, we describe our quasi-polynomial time algorithm MstdbQ for the MSTDB problem. Given a graph $G = (V, E)$, a set L of nodes, a degree upper bound B_H and a degree lower bound B_L, the MstdbQ algorithm finds an MST T such that (a) every vertex has degree at most $B_H + O(\frac{\log n}{\log\log n})$ and (b) every vertex in L has degree at least $B_L - O(\frac{\log n}{\log\log n})$. If it fails, it produces a combinatorial witness to show that there exists no MST in the graph with degree at most B_H in which the vertices

in L have degree at least B_L.

This algorithm uses the same framework as the algorithm in Section 4.3. Starting with an arbitrary MST, it proceeds in phases of improvement. Each phase is essentially a combination of the two algorithms MaxdmstQ and MindmstQ, which are described in Sections 4.5.1 and 4.5.2. The MaxdmstQ algorithm attempts to decrease the degree of a node with high degree, and MindmstQ symmetrically attempts to increase the degree of a node in L which has low degree. If they both fail, the combinatorial witnesses they provide can be combined to produce a witness to the fact that there exists no MST in the graph with degree at most B_H in which the vertices in L have degree at least B_L.

To measure the progress made by our algorithm, we define a potential function $\phi_Q(T)$ on the set of all MSTs as follows:

$$\phi_Q(T) = \sum_v (2n)^{deg(v)-B_H} + \sum_{v \in L} (2n)^{B_L - deg(v)}.$$

While our algorithms MaxdmstQ and MindmstQ execute several swaps in order to improve the degree of one node, we nevertheless ensure that the potential decreases in each phase. Note that we have replaced the 5 in the definition of ϕ by $2n$; this is necessary to ensure the decrease in potential.

4.5.1 The MaxdmstQ algorithm

Before describing our algorithm formally, we introduce some more definitions and notation. Let T be the MST of G at the beginning of the current phase of the algorithm. Recall that $d_{\max}(T)$ denotes the maximum degree over all vertices in the current MST T.

Given an MST T, a set X of nodes, and an integer d, the aim of the MaxdmstQ algorithm is to reduce the degree of some vertex with degree at least d without

reducing the degree of any vertex in X or increasing the degree of any vertex with degree at least $d-1$. This is achieved via a series of edge swaps. We show that for appropriately chosen X, the algorithm outputs a tree T' that has lower potential $\phi_Q(T')$ than T. If it fails to do this, it outputs a combinatorial witness proving a lower bound on maximum degree of any MST.

Throughout the MaxdmstQ algorithm, we maintain a center set $W \subset V$, and a partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ of $V \setminus W$ into clusters. The initial center set W_0 is $S_{\geq(d-1)}$, and the initial clusters are the connected components created by deleting W_0 from T. In general, though, a cluster defined during the course of the MaxdmstQ algorithm is not necessarily internally connected. We also maintain a set R of restricted edges, that will not be allowed to be added to the tree; R is initially empty.

We use W to refer generally to the center set, and W_i to denote the center set at some specific iteration i of this phase. For a node $u \in W$, the clusters connected to u by tree edges are called the *children clusters* of u.

Let $\mathcal{T} \subseteq T$ be a Steiner tree on the nodes of W_0 (the Steiner nodes are the nodes in $V - W_0$). We say that we *freeze* the edges of $\mathcal{T}$ incident on W_0, meaning that the edges of $\mathcal{T}$ that are incident on W_0 are not allowed to be removed by any swap. As we show later, freezing these edges ensures that executing the edge swaps at the end of a phase of the algorithm results in a tree.

Unlike the MaxdmstP algorithm of Section 4.3.1, the MaxdmstQ algorithm does not restrict itself to making only (X, d)-deflating swaps. It instead finds a *sequence of swaps* that has a similar effect. More precisely, given a set X of nodes, a sequence of swaps $(e_1, e'_1), (e_2, e'_2), \ldots, (e_k, e'_k)$ is called *an (X, d)-deflating sequence* if (a) e_1 has exactly one endpoint in $S_{\geq d}$, (b) for all i, $1 \leq i < k$, e'_i and e_{i+1} have a common endpoint in $S_{\geq(d-1)}$, (c) for all i, $1 \leq i \leq k$, e_i is not incident on X, and (d) after all swaps in the sequence are performed, no node in $S_{\geq(d-1)}$ has larger degree than it does in T.

The aim of the MaxdmstQ algorithm is to construct an (X, d)-deflating sequence of swaps from swaps of a particular type. Given a set X of nodes, and a partition of all the nodes into a center set W and a set $\mathcal{C}$ of clusters, a swap (e, e') is called (X, W)-*deflating* if (a) e' is an inter-cluster edge, and (b) e has exactly one endpoint in W and is not incident on X. Recalling the definition of an (X, d)-deflating swap from Section 4.3.1, we note that an (X, W)-deflating swap is not necessarily (X, d)-deflating, since e' may be incident on $S_{\geq(d-1)}$, or e itself may be incident on $W \setminus S_{\geq d}$.

The algorithm repeatedly finds (but does not perform) an (X, W)-deflating swap (e, e') such that e is not frozen and e' is not in R. After finding such a swap, it then removes the endpoint u of e that is in W from W and performs a *merge* step in which a new cluster C_u is formed by merging u with some other clusters. This swap (e, e') is called *the u-deflating swap*. The *restrict* step (see Figure 4.6) of the algorithm prevents the swapping in of an edge between two nodes that initially have degree $(d-1)$ in T. The process is repeated until we either run out of (X, W)-deflating swaps that don't use frozen or restricted edges, or we remove a vertex u with degree at least d from W. In the former case, we construct a witness. In the latter case, we execute a particular (X, d)-deflating sequence of swaps, specified below, that decreases the degree of u. Figure 4.7 illustrates a run of the MaxdmstQ algorithm.

In order to specify the sequence of swaps executed at the end of the run of the MaxdmstQ algorithm, we first define, for any node u that is removed from W, *the u-deflating sequence* of swaps. It is defined inductively, as follows. Let (e, e') be the u-deflating swap, defined during the course of the MaxdmstQ algorithm. Note that because of the restrict step of the algorithm, at most one endpoint of e' is in the initial center set W_0. If neither of the endpoints of e' is in W_0, the u-deflating sequence is defined to contain just the u-deflating swap (e, e'). If one of the endpoints, say u', of e' is in W_0, the u-deflating sequence is defined inductively by adding the swap (e, e') to the beginning of the u'-deflating sequence. Since u' must be removed from the

Algorithm MaxdmstQ(G, T, X, d)

Initialize $W = S_{\geq(d-1)}$, $R = \emptyset$.
Let $\mathcal{C}$ be the components formed upon deleting W from T.
Let $\mathcal{T} \subseteq T$ be a Steiner tree on $S_{\geq(d-1)}$.
Freeze the edges of $\mathcal{T}$ incident on $S_{\geq(d-1)}$.
repeat
  Find an (X, W)-deflating swap $(e = (u, v), e' = (u', v'))$
    such that e is not frozen and $e' \notin R$.
  if no such swap exists
    break out of loop.
  Let u be the endpoint of e in W.
  Remove u from W. Call (e, e') *the u-deflating swap.*
  Merge: Form a new cluster C_u by merging u
    with the cluster containing u',
    the cluster containing v',
    and all the children clusters of u.
  Restrict: For each edge (u, w) such that $w \in S_{\geq(d-1)} \setminus W$,
    add (u, w) to R.
until some node $u \in S_{\geq d}$ is removed from W.
if removed a node $u \in S_{\geq d}$ from W
  then execute (X, d)-deflating sequence of swaps,
    starting with the u-deflating swap.

Figure 4.6: Pseudo-code for MaxdmstQ.

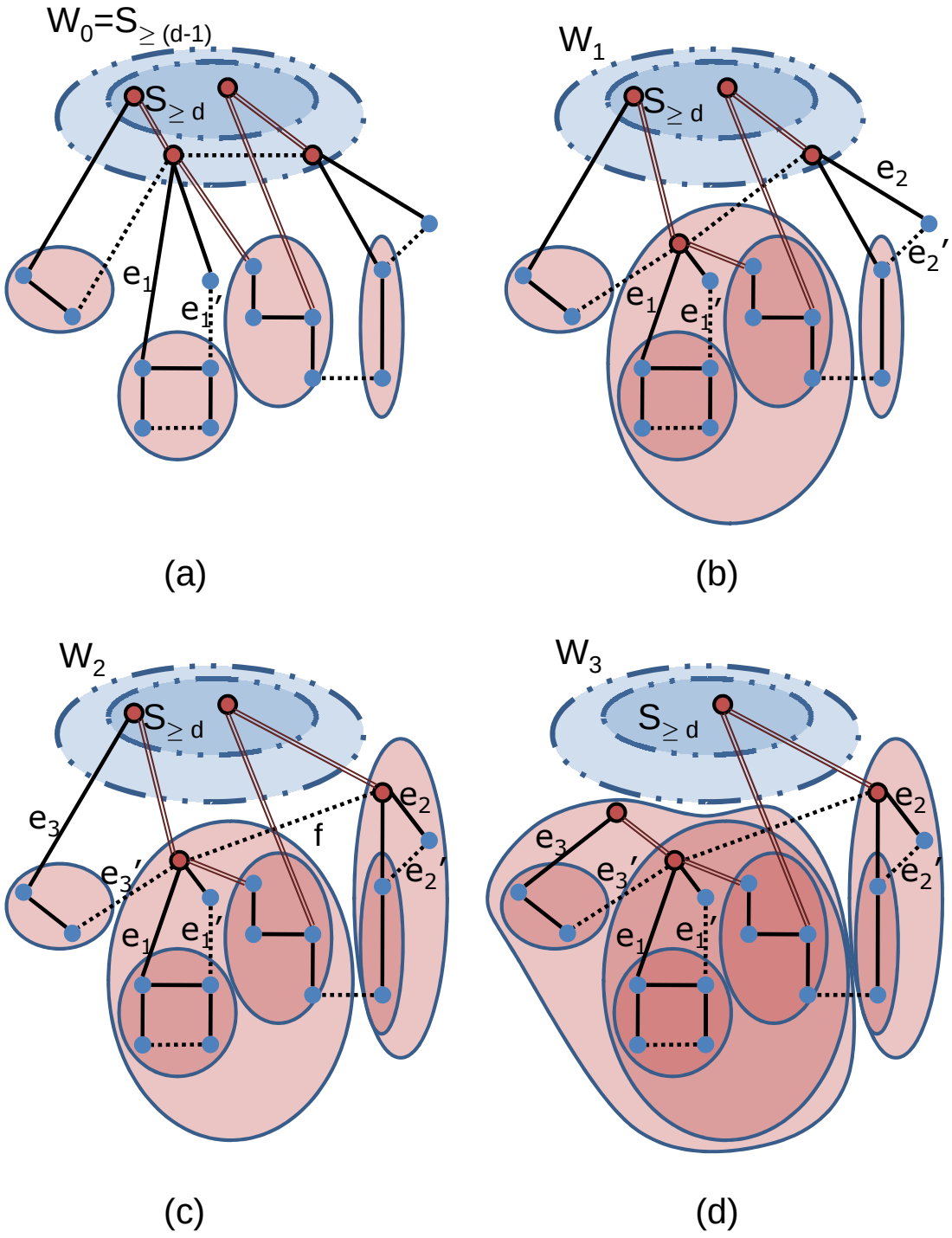

Figure 4.7: A possible partial run of the MaxdmstQ algorithm. The frozen edges are shown by double lines. In each step, a swap is discovered that leads to the removal of a vertex from the center set W and the merging of some clusters. In step (c), the edge labeled f is added to R. Finally, in step (d), a vertex in $S_{\geq d}$ is removed from W. The execution (not shown) of the swaps (e_3, e_3') and (e_1, e_1') end this run.

center set before u is, this sequence is well-defined.

If u is the vertex of degree at least d removed from W, then the sequence of swaps executed in the last step of the algorithm is the u-deflating sequence. Lemmas 4.7 and 4.8 show that this sequence is in fact an (X, d)-deflating sequence. If the algorithm terminates instead by running out of (X, W)-deflating swaps, Lemma 4.9 and Theorem 4.11 show that we can interpret the remaining structure as a witness to the fact that the maximum degree of T is close to optimal.

For any node u removed from the center set, recall that C_u denotes the cluster formed by the merge step after u is removed (see Figure 4.6 for a precise definition); note that since new clusters are formed by merging old clusters, a cluster C_u formed during one iteration lies wholly within some cluster in all subsequent iterations.

Lemma 4.7. *Suppose we remove a vertex u from the center set W_i in some iteration i. If we execute the u-deflating sequence, then (a) the degree of u decreases by one, (b) no node in $S_{\geq(d-1)}$ has larger degree than it does in T, and (c) the degree of no node changes by more than one.*

Proof. We prove this by induction on the length of the u-deflating sequence. We strengthen the claim to add the condition that (d) the degree of every node outside C_u is unchanged.

In the base case, when the u-deflating sequence contains only one swap (e, e') with the endpoints of e' outside $S_{\geq(d-1)}$, the conditions (a),(b), and (c) follow immediately, and (d) follows from the merge step.

Now suppose that the claim holds for the u'-deflating sequence, which is augmented with the swap (e, e') to form the u-deflating sequence. The swap $(e = (u, v), e' = (u', v'))$ increases the degree of u' by one and the u'-deflating sequence decreases it by one; thus the degree of u' is unchanged. By construction v' is outside $S_{\geq(d-1)}$, and thus we have proved (a) and (b). The edge (u, v) does not lie on the

path from u to u' in T, since if it did, it would be in $\mathcal{T}$ and hence frozen. Also, v is in the same cluster as v'; if it were not, the path in T from u to v' would contain another node from W_0, and thus (u, v) would be in $\mathcal{T}$ and hence frozen. Since the edge (u', v') is an inter-cluster edge, v' lies outside $C_{u'}$, and thus so does v. Part (d) of the inductive hypothesis then implies that the degrees of v and v' are unaffected by the u'-deflating sequence. Thus (c) follows. Finally the merge step implies that u, v, v', and $C_{u'}$ all lie within C_u, and thus (d) is true as well. □

We next show that executing the u-deflating sequence of swaps results in a tree. If executed in isolation, any single swap in the sequence obviously does not introduce a cycle. However, it is not as clear that doing them simultaneously does not disrupt the tree structure. In fact, the frozen edges of the Steiner tree play a crucial role here.

Lemma 4.8. *For any $u \in W_0$, the graph produced after performing the u-deflating sequence of swaps is a tree.*

Proof. Let $(e_1, e_1'), (e_2, e_2'), \ldots, (e_k, e_k')$ be the u-deflating sequence of swaps, so that e_1 is incident on u, and e_i' and e_{i+1} share an endpoint. Let T_j be the graph resulting after we execute the first j of these swaps, i.e. $T_0 = T$ and $T_j = T_{j-1} \setminus \{e_j\} \cup \{e_j'\}$. We show inductively that T_j is a tree.

The base case is immediate. Suppose that T_{j-1} is a tree for some j. It suffices to show that if $e_j = (u_j, v_j)$ could be replaced by $e_j' = (u_j', v_j')$ in T, it is a valid swap in T_{j-1} as well. In other words, if e_j lies on the tree path from v_j' to u_j' in T, then e_j lies on the tree path from v_j' to u_j' in T_{j-1} as well. Note that by construction e_j and e_{j-1}' share an endpoint u_j and that e_{j-1}' is the first edge in the sequence incident on C_{u_j}.

Consider the tree path in T from v_j' to u_j' (see Figure 4.8). Note that by construction, all tree edges leaving C_{u_j} are incident on W_0. If the path lies wholly within C_{u_j}, then it must still exist in T_{j-1}, since all deleted edges $e_1, \ldots, e_{j-1}$ lie outside C_{u_j}. Otherwise this path can be decomposed into three segments P_1, P_2, and P_3, such that

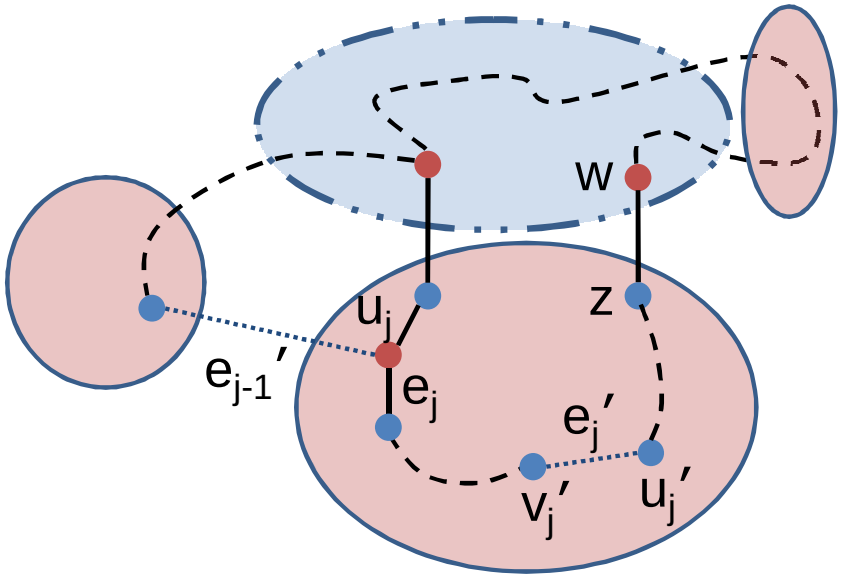

Figure 4.8: Proof of Lemma 4.8. The dotted edges are not in T; the dashed lines indicate paths.

$v'_j \in P_1$, $u'_j \in P_3$, P_1 and P_3 are maximal paths contained in C_{u_j}, and P_2 is a path connecting P_1 and P_2. Without loss of generality, u_j (and thus e_j) lies in the segment P_1 containing v'_j. Let (w, z) be the last edge on P_2; i.e. $w \in W_0$ and z is connected to u'_j by P_3.

The paths P_1 and P_3 are preserved in T_{j-1}, since deleted edges $e_1, \ldots, e_{j-1}$ lie outside C_{u_j}. The path $P_2 \setminus (w, z)$ is in the Steiner tree $\mathcal{T}$, since it connects u_j to w in T; thus the freezing step ensures that it is preserved in T_{j-1}. Finally, the edge (w, z) must also exist in T_{j-1}: If $(w, z) \notin T_{j-1}$, then $T_{j-1} \setminus T$ must contain an edge e'_i, $1 \leq i \leq j-1$, incident on the component of $T \setminus \{(w, z)\}$ containing u'_j. Note that by construction, this component does not contain u_j, and therefore $e'_i \neq e'_{j-1}$. Since this component is contained in C_{u_j}, then e'_i is incident on C_{u_j}, which contradicts the observation that e'_{j-1} is the only edge incident on C_{u_j} in $T_{j-1} \setminus T$. Thus we have shown that the entire v'_j-u'_j path in T is preserved in T_{j-1}. The claim follows. □

The MaxdmstQ witness

Now we show how to interpret the structure produced by running the MaxdmstQ algorithm as a high-degree witness, described in Section 4.3.1. Let W be the center set, $C_1, C_2, \ldots, C_k$ be the clusters, R the restricted edges, and T the tree when the algorithm terminates. Let T_X be the set of all edges in T in $X \times W$, and $F_{\mathcal{T}}$ be the set of Steiner tree edges that are frozen by the algorithm. We let $F = T_X \cup F_{\mathcal{T}}$. Let $\mathcal{W}_Q = (\{W, C_1, \ldots, C_k\}, W, F)$. We first argue that $\mathcal{W}_Q$ is a witness for the graph with the edges in R deleted.

Lemma 4.9. *$\mathcal{W}_Q = (\{W, C_1, \ldots, C_k\}, W, F)$ is a high-degree witness for $G' = (V, E \setminus R)$.*

Proof. Suppose that there is some MST T' of G' containing F that does not contain an edge from C_i to W.

Let T_i be the set of edges in T connecting C_i to W. By construction, there are no inter-cluster tree edges, so T_i is non-empty. By the exchange property, there is a set $T_i' \subseteq T'$ such that $T_1 = T \setminus T_i \cup T_i'$ is an MST. Since C_i and $V \setminus C_i$ are disconnected in $T \setminus T_i$, there must be an edge $e' \in T_i'$ connecting them. By the exchange property, there is an edge $e \in T_i$ such that (e, e') is a swap with respect to T. Since $e' \in T'$, e' is not incident on W, and thus (e, e') is an (X, W)-deflating swap, contradicting our assumption. Hence the claim. □

Lemma 4.10. *At most $2|W_0|$ edges are frozen by the algorithm, where W_0 is our initial witness.*

Proof. The average degree of a tree is at most two, and the degree of a Steiner vertex is at least two. Thus the average degree of the terminals in any Steiner tree is at most two. □

Finally, we show that when given an appropriate set X of vertices, the algorithm either computes a tree with lower potential, or produces a good witness.

Theorem 4.11. *Suppose we are given as input an MST T, an integer d, and a set X of nodes. Then the algorithm* MaxdmstQ*, when called with $X = S^L_{\leq(B_L+B_H-d+1)}$ runs in polynomial time, and either outputs a tree with potential function value at most $\phi(T) - (2n)^{d-B_H-1}$, or finds a witness that certifies that for any fractional MST τ of G,*

$$d_{\max}(\tau) \geq (d-2) - 5\frac{\left|S_{\geq(d-1)}\right|}{|S_{\geq d}|} - \frac{|X|}{\left|S_{\geq(d-1)}\right|}.$$

Proof. Since there are at most $|V| \cdot |E|$ swaps, each iteration runs in polynomial time. In each iteration, we remove a vertex from W, and thus the overall algorithm runs in polynomial time.

From Lemmas 4.7 and 4.8, we see that if the MaxdmstQ algorithm removes a vertex of degree d from the center set, it produces an MST T'. In this case, there is at least one vertex $u \in S_{\geq d}$ whose degree decreases by one, and no vertex has its degree raised above $d-1$. In addition, the degree of vertices in $S^L_{\leq(B_L+B_H-d+1)}$ does not decrease, and no vertex has its degree changed by more than one. Thus the decrease in potential due to u is at least $(2n-1)(2n)^{d-B_H-1}$, while the total increase due to the other vertices is at most $n(2n)^{d-B_H-1}$. The claimed decrease in potential follows.

If the algorithm is unable to improve any vertex of degree d, it instead halts after t iterations with another MST of degree d. Let $W_0 = S_{\geq(d-1)}$ be the initial center set, and let $W_t \supseteq S_{\geq d}$ be the final center set. The number of components obtained by deleting the edges incident to W_t is at least $(d-2)|W_t|$. The number of merges performed by the algorithm is at most $|W_0 \setminus W_t|$: To see this, note that every time we do a merge in the merge step, we remove a node from W. Therefore the number of clusters at the end of the algorithm is at least $(d-2)|W_t| - |W_0 \setminus W_t|$. According to Lemma 4.9, the witness $\mathcal{W}_Q$ is a high-degree witness for $G' = (V, E \setminus R)$. We can make use of the lower bound, however, by observing that all edges in R are incident

on $W_0 \setminus W_t$, so that no MST of G can use more than $|W_0 \setminus W_t|$ of these edges. Using Lemma 4.9, Lemma 4.10, and the fact that the number of edges in T in $W_0 \times X$ is at most $|W_0| + |X|$, we conclude that for any fractional MST τ of G,

$$d_{\max}(\tau) \geq (d-2) - 5\frac{\left|S_{\geq(d-1)}\right|}{|S_{\geq d}|} - \frac{|X|}{\left|S_{\geq(d-1)}\right|}.$$

□

4.5.2 The MindmstQ algorithm

Before describing our algorithm formally, we introduce some definitions and notation. Let T be the MST of G at the beginning of the algorithm. Recall that $d^L_{\min}(T) = \min_{v \in L} \deg_T(v)$ is the minimum degree over all nodes in L in tree T. Recall also that for a tree T, we denote by $S^L_{\leq d}$ the subset of nodes in L which have degree at most d in T.

The algorithm takes as input the graph G, a tree T, sets L and Y of vertices, and a positive integer d. The aim of the MindmstQ algorithm is to increase the degree of some node in $S^L_{\leq d}$ without increasing the degree of any vertex in Y or decreasing the degree of any vertex in $S^L_{\leq(d+1)}$. This is achieved via a series of edge swaps. We show that for appropriately chosen Y, the algorithm outputs a tree T' that has lower potential $\phi_Q(T')$ than that of T. If it fails to do this, it outputs a combinatorial witness proving an upper bound on the degree of L in any MST.

Throughout the MindmstQ algorithm, we maintain a center set W, and a partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ of $V \setminus W$ into clusters. The initial center set W_0 is $S^L_{\leq(d+1)}$, and the initial clusters are the connected components created by deleting W_0 from T. In general, during the course of the algorithm, a component may be split into several clusters, though each cluster remains internally connected in T. We use the term *intra-cluster edge* to refer to an edge whose endpoints are in the same cluster.

We also maintain a set N of special nodes and a set F of frozen edges. N is initially set to $S^L_{\leq(d+1)}$ and will be augmented during the algorithm. Let $\mathcal{T}$ be a Steiner tree on N such that $\mathcal{T} \subseteq T$ (with $V \setminus N$ as the Steiner nodes). We *freeze* the edges of $\mathcal{T}$ adjacent to N— that is, we disallow the algorithm from swapping out these edges. The set F is initialized to be the set of frozen edges of $\mathcal{T}$. As we show later, freezing these edges ensures that executing the sequence of edge swaps at the end of the algorithm results in a tree.

Given a set Y of nodes, a sequence of swaps $(e_1, e'_1), (e_2, e'_2), \ldots, (e_k, e'_k)$ is called a (Y, d)-*inflating sequence* if (a) e'_1 has exactly one endpoint in $S^L_{\leq d}$, (b) for all $i, 1 \leq i < k$, e_i and e'_{i+1} have a common endpoint in $S^L_{\leq(d+1)}$, (c) for all $i, 1 \leq i \leq k$, e'_i is not incident on Y, and (d) after all swaps in the sequence are performed, no node in $S^L_{\leq(d+1)}$ has smaller degree than it does in T.

The aim of the MindmstQ algorithm is to construct a (Y, d)-inflating sequence of swaps from swaps of a particular kind. Given a set Y of nodes, and a partition of all nodes into a center set W and a set $\mathcal{C}$ of clusters, a swap (e, e') is called (Y, W)-*inflating* if (a) e is an intra-cluster edge, and (b) e' has exactly one endpoint in W and is not incident on Y. Recalling the definition of (Y, d)-inflating swaps in Section 4.3.2, we note that a (Y, W)-inflating swap (e, e') may not be (Y, d)-inflating, since e may be incident on $S^L_{\leq(d+1)}$ or e' may be incident on $W \setminus S^L_{\leq d}$.

The algorithm repeatedly finds (but does not execute) an (Y, W)-inflating swap (e, e') such that the edge e is not frozen. It then removes the endpoint $u' \in W$ of e' from W and performs a *merge* step to create a cluster $C_{u'}$ by merging u' with some other clusters (see Figure 4.9). The swap (e, e') is called the u'-*inflating swap*. The following *split* step breaks the cluster containing e. Let u be the endpoint of e closer to u' in T. If u is not in W_0, we call such a swap a *basic swap*. If (e, e') is a basic swap, the *freeze* step adds u and v' to N and augments F appropriately (see Figure 4.9). Note that for a non-basic swap, u is already in N. In either case, we call

Algorithm MindmstQ(G, T, L, Y, d)

Initialize $W = N = S^L_{\leq(d+1)}$.
Let $\mathcal{C}$ be the components formed upon deleting W from T.
Let $\mathcal{T} \subseteq T$ be the Steiner tree on N.
Let F be the edges of $\mathcal{T}$ incident on N.
repeat
  Find a (Y, W)-inflating swap $(e = (u, v), e' = (u', v'))$ such that $e \notin F$.
  if no such swap exists
    break out of loop.
  Let u' be the endpoint of e' in W.
  Remove u' from W. Call (e, e') the *u'-inflating swap.*
  Merge: Form a new cluster $C_{u'}$ by merging u'
      along with all the children clusters of u',
      except for those which would involve a merge
      along an edge between u' and $S^L_{\leq(d+1)}$.
  Split the cluster containing (u, v) along e into two clusters C_u and C_v.
  Freeze: Let u be the endpoint of e closer to u' in T.
    If $u \notin W_0$, add u, v' to N.
    Augment $\mathcal{T}$ to a Steiner tree on N
    and add the new edges of $\mathcal{T}$ incident on N to F.
    Call the u-v' path in T the *tail* of this swap.
until some node u' in $S^L_{\leq d}$ is removed from W.
if removed a node $u' \in S^L_{\leq d}$ from W
  then execute the u'-inflating sequence of swaps.

Figure 4.9: Pseudo-code for MindmstQ.

the u-v' path in T the *tail* of the swap. The process is repeated until we either run out of (Y, W)-inflating swaps that don't involve frozen edges or we remove a vertex $u' \in S^L_{\leq d}$ from W. In the former case, we construct a witness. In the latter case, we execute a particular (Y, d)-inflating sequence of swaps, specified below, that increases the degree of u' by one. As we see in Lemma 4.12, the merge, split, and freeze steps ensure that we never swap out more than one edge incident on any node, and that this sequence of swaps results in a tree. Figure 4.10 illustrates a run of the MindmstQ algorithm.

In order to specify the sequence of swaps executed at the end of the run of the

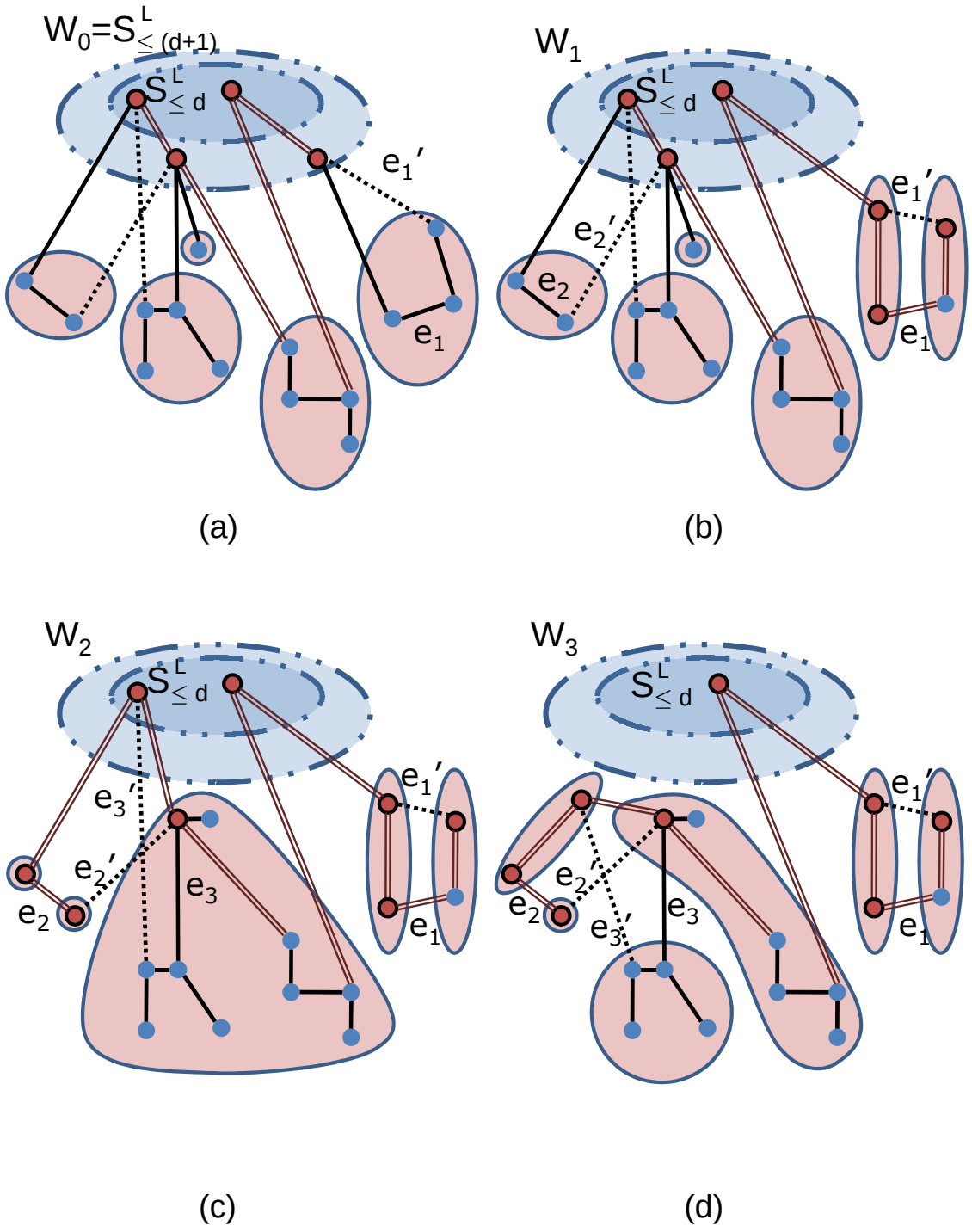

Figure 4.10: A possible partial run of the MindmstQ algorithm. The frozen edges are shown by double lines, and the vertices in N are circled. In each step, a swap is discovered that leads to the removal of a vertex from the center set W and the merging and splitting of some clusters. In steps (b) and (c), new edges are also frozen. Finally, in step (d), a vertex in $S^L_{\leq d}$ is removed from W. The execution (not shown) of the swaps (e_3, e_3') and (e_2, e_2') end this run.

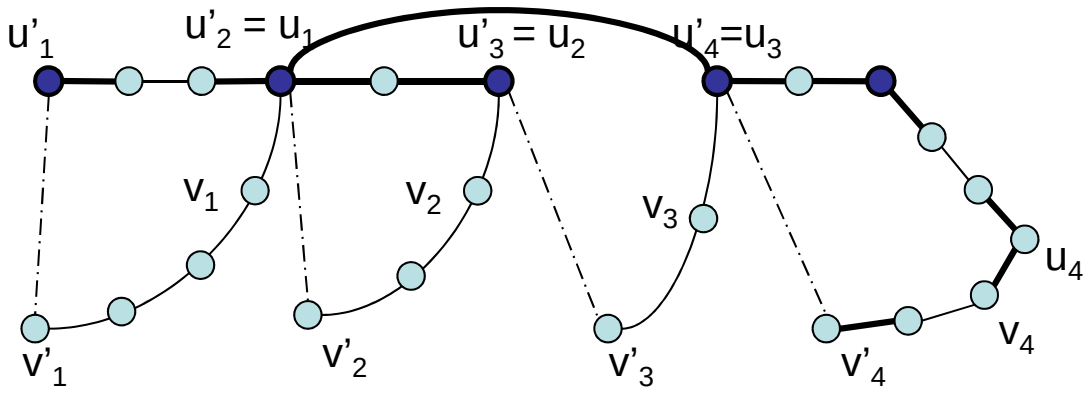

Figure 4.11: The sequence of swaps. The bold edges are frozen. The dark-colored nodes are in W_0. The paths from u_i to v'_i are the tails.

MindmstQ algorithm, we first define, for any node u' that is removed from W, the *u'-inflating sequence* of swaps inductively as follows. Let (e, e') be the u'-inflating swap. We shall argue shortly that at most one endpoint of e is in the initial center set W_0. If neither of the endpoints of e is in W_0, then the u'-inflating sequence is defined to contain just the u'-inflating swap (e, e'). If one of the endpoints, say u, of e is in W_0, the u'-inflating sequence is defined inductively by adding the swap (e, e') to the beginning of the u-inflating sequence.

If u' is the vertex of degree at most d removed from W, then the sequence of swaps executed in the last step of the algorithm is the u'-inflating sequence. Lemma 4.12 shows that this sequence is in fact a (Y, d)-inflating sequence. Theorem 4.15 shows that if we run out of (Y, W)-inflating swaps, we can turn the resulting structure into a low-degree witness certifying an upper bound on the degree of L in any MST.

Lemma 4.12. *Suppose that we remove a vertex u' from W. Then if we execute the u'-inflating sequence of swaps defined above, (a) the degree of u' increases by 1, (b) no node in $S^L_{\leq(d+1)}$ has smaller degree than it does in T, (c) the degree of no node changes by more than one, and (d) the resulting graph is a tree.*

Proof. Let $(e_1, e'_1), (e_2, e'_2), \ldots, (e_k, e'_k)$ be the u'-inflating sequence, where $(e_i, e'_i) = ((u_i, v_i), (u'_i, v'_i))$, $u' = u'_1$, and $u'_{i+1} \in S^L_{\leq(d+1)}$ is the endpoint shared by e_i and e'_{i+1}.

We first observe that for any swap $((u_i, v_i), (u'_i, v'_i))$, u_i and v_i are not both in W_0. This is because at the time of discovery of this swap, (u_i, v_i) is an intra-cluster edge in T. The merge step of the algorithm ensures that we never merge clusters along an edge between two nodes in W_0. Thus an edge between two nodes in W_0 can never be an intra-cluster edge.

Recall that for swap (e_i, e'_i), we refer to the endpoint of e_i closer to u'_i as u_i. Thus e_i is on the u'_i-v_i path in T. If v_i were in N, then e_i would be frozen and we wouldn't have chosen the swap (e_i, e'_i). Thus for $i < k$, v_i must be different from u'_{i+1} (which is in $S^L_{\leq(d+1)}$ and hence in N). Since e_i and e'_{i+1} share the endpoint u'_{i+1}, we conclude that $u_i = u'_{i+1}$ for $i < k$. See Figure 4.11.

For $i < k$, since $u_i = u'_{i+1}$ is in W_0, v_i is not in W_0. Moreover, (e_k, e'_k) is a basic swap, so u_k and v_k are not in W_0. It follows that if u_k or v_i, for any i, is in L, then it has degree at least $d + 2$.

Let T_l be the tree resulting from executing the first j swaps, i.e. $T_0 = T$ and $T_l = T_{l-1} \setminus \{e_l\} \cup \{e'_l\}$. We show inductively that T_l is a tree. Let $N = N_t$ be the final set of special nodes. We strengthen the inductive claim to add the condition that the Steiner tree on N is undisturbed during the swaps.

Recall that the tail of swap (e_i, e'_i) is defined to be the u_i-v'_i path in T. We make the following claims.

Claim 4.13. *The tail for swap i for $i < k$ intersects with the Steiner tree $\mathcal{T}$ on W_0 only in u_i.*

Proof. Recall that $u'_{i+1} = u_i$ is in W_0. If the tail of the swap i contained a node $w \in \mathcal{T}$ different from u_i, the edge (u_i, v_i) would be on the u_i-w path and thus frozen. □

Claim 4.14. *The tail for swap i is vertex-disjoint from the tail for swap j for $1 \leq i < j \leq k$.*

Proof. First, let $i < j < k$. We observe that $(u_i, v_i) \neq (u_j, v_j)$. This holds because the splitting step of the algorithm ensures that (u_j, v_j) is an inter-cluster edge after the swap $((u_j, v_j), (u'_j, v'_j))$ is discovered.

There is a Steiner tree path from u_i to u_j. Claim 4.13 implies that e_i and e_j are not on this path. If the tails of swaps i and j intersected in a node w, there is a simple path from u_i to u_j via w, that contains e_i and e_j and is therefore distinct from the Steiner tree path. This however contradicts the acyclicity of T.

Now let us look at the tail of swap k. Since this swap is basic, u_k and v_k are in N when swap i, $i \neq k$, is discovered. Suppose that the tail of swap k shares a vertex w with the tail of swap i, $i < k$. Then w has a path to u_i with e_i as its last edge. Since w lies on the tail of swap k, the tail consists of the v'_k-w path and the u_k-w path. Either the u_i-u_k path or the u_i-v'_k path must go through w and thus contains e_i. Since v'_k, u_k and u_i are all in N, e_i must be frozen at the time of discovery of the swap (e_i, e'_i), which contradicts the fact that the swap (e_i, e'_i) was chosen. □

Proof of Lemma 4.12, continued. Let T_j be the tree resulting from executing the first j swaps, i.e. $T_0 = T$ and $T_j = T_{j-1} \setminus \{e_j\} \cup \{e'_j\}$. Let $N = N_t$ be the final set of special nodes.

We show inductively that T_j is a tree. The base case is immediate.

Consider the path from u'_j to v'_j in T; this path contains e_j by construction. It is enough to show that this path exists in T_{j-1}. This path consists of the u'_j-u_j path, along with the tail of swap j, which is the u_j-v'_j path. Since both u_j and u'_j lie in N, the path between them is not affected by the first $(j-1)$ swaps; indeed, each of the first $(j-1)$ swaps is a non-basic swap which deletes an edge incident on N, and the edges from $\mathcal{T}$ incident on N are frozen. Since the deleted edges $e_1, \ldots, e_{j-1}$ lie on the first $(j-1)$ tails, which by Claim 4.14 are disjoint from the tail of swap j, the u_j-v'_j path in T is preserved in T_{j-1}. Thus the entire u'_j-v'_j path in T is preserved in T_{j-1}.

It follows inductively that T_j is a tree, for $1 \leq j \leq k$, proving part (d) of Lemma 4.12.

We next show parts (a), (b), and (c) of Lemma 4.12. The tail of swap i contains u_i, v_i, and v'_i, and the tails of the swaps are vertex-disjoint according to Claim 4.14. This implies that except for the equalities $u_i = u'_{i+1}$, the nodes involved in the swaps are all distinct. The node u'_1 gains an edge because of the swap (e_1, e'_1) and is not involved in any other swap, which proves part (a). The nodes u'_i, $i > 1$, are the only nodes involved in the swaps that lie in $S^L_{\leq(d+1)}$. Each of these nodes gains and loses one unit of degree, implying part (b). Since the remaining involved nodes are all distinct from each other, part (c) follows. □

Theorem 4.15. *Suppose we are given as input an MST T, an integer d and a set Y of nodes. Then, the algorithm* MindmstQ *when called with $Y = S^L_{\leq(B_L+B_H-d-1)}$ either outputs a tree with potential function value at most $\phi(T) - (2n)^{d-B_H-1}$, or finds a witness $\mathcal{W}$ that certifies that for any fractional MST τ of G,*

$$d^L_{\min}(\tau) \leq (d+1) + 10\frac{\left|S^L_{\leq(d+1)}\right|}{\left|S^L_{\leq d}\right|} + \frac{|Y|}{\left|S^L_{\leq d}\right|} - 6.$$

Proof. Lemma 4.12 implies that if the algorithm removes a vertex of degree d from W, it can find a tree in which the degree of this node is increased by one. The calculation showing that the potential decreases is identical to that in Theorem 4.11.

If not, we show how to convert the structure produced by the MindmstQ algorithm into a low-degree witness of the type in Lemma 4.3. Suppose the algorithm begins with an MST T, a set Y, a degree d, and a set L, but fails to improve the degree of any vertex in L of degree at most d. It terminates after $t \geq 0$ iterations with W_t as the center set where $S^L_{\leq d} \subseteq W_t \subseteq W_0 = S^L_{\leq(d+1)}$, and $C_1, C_2, \ldots, C_{k'}$, as the clusters.

Let $W = W_t$ and $R = (W \times Y) \setminus T$. Let I be the set of inter-cluster edges in T;

we set H to $F \cup I$. We argue that

$$\mathcal{W}_{L,Q} = (\{W, C_1, \ldots, C_{k'}\}, W, Y, R, H)$$

is a low-degree witness.

Assume the contrary, i.e. $\mathcal{W}_{L,Q}$ is not a low-degree witness. Then there is an MST T' of G that contains H, excludes R and contains two edges $e'_1, e'_2 \notin H$ from cluster C_i to W.

The proof is similar to that of Theorem 4.4. Let (e_1, e'_1) be a swap that adds e'_1 to T, such that e_1 is not in H, and let T_1 be the tree $T \setminus \{e_1\} \cup \{e'_1\}$. Note that C_i is internally connected in T. If e_1 were not incident on W_0, (e_1, e'_1) would be (Y, W)-inflating, contradicting our assumption that the algorithm runs out of such swaps. Thus e_1 must be incident on W_0, and therefore C_i is internally connected in T_1 as well.

Now let (e_2, e'_2) be a swap that adds e'_2 to T_1 such that $e_2 \notin H$ and $e_2 \neq e'_1$. Let $T_2 = T_1 \setminus \{e_2\} \cup \{e'_2\}$. Suppose that e_2 is not incident on W_0. Then the swap (e_2, e'_2) is (Y, W)-inflating with respect to T_1. Since the algorithm runs out of (Y, W)-inflating swaps with respect to T, this swap does not exist in T. This implies that (e_2, e'_1) must be a swap in T. However, this swap is also (Y, W)-inflating, which is a contradiction. Thus e_2 is incident on W_0, and therefore C_i is internally connected in T_2.

Let u'_1 and u'_2 be the endpoints of e'_1 and e'_2, respectively, in W. Because C_i is internally connected in T_2, there is a path from u'_1 to u'_2 in T_2 containing the edges e'_1 and e'_2. There is a Steiner tree path in T from u'_1 to u'_2. This path is also present in T_2 since e_1 and e_2 are incident on W_0 and not frozen, while all Steiner tree edges incident on W_0 are frozen. Thus there are two distinct paths from u'_1 to u'_2 in T_2, which contradicts the acyclicity of T_2. We conclude that $\mathcal{W}_{L,Q}$ is a low-degree witness.

To finish the proof of Theorem 4.15, we compute the bound implied by Lemma 4.3.

First we count the number of clusters created by the algorithm. The number of clusters is the number of components formed by deleting W_t from T, of which there are at most $(d+1)|W_t|$, plus the number of splits. There are a total of $|W_0 \setminus W_t|$ splits in the split step of the algorithm, each of which creates one additional cluster by splitting along the intra-cluster edge involved in the swap. There are also some clusters split (or rather, not merged) by tree edges adjacent to two $(d+1)$-degree nodes, but there can be at most $|W_0 \setminus W_t|$ of these. Thus, the number of clusters at the end of the algorithm is at most $(d+1)|W_t| + 2|W_0 \setminus W_t|$.

The set N has at most $W_0 + 2\,|W_0 \setminus W_t|$ nodes in it, and thus there are at most $2\,|W_0| + 4\,|W_0 \setminus W_t|$ edges in F (see proof of Lemma 4.10). Since each split (and each non-merge) contributes at most one inter-cluster edge, there are at most $2\,|W_0 \setminus W_t|$ edges in I. Therefore $|H| \leq 2\,|W_0| + 6\,|W_0 \setminus W_t|$.

Recall that Lemma 4.3 proves that for a witness $(\{W, C_1, \ldots, C_{k'}\}, W, Y, R, H)$ and any fractional tree τ,

$$d^L_{\min}(\tau) \leq \frac{k' + 2\,|W| + |U| + |Y| + |H| - 2}{|W|}.$$

Plugging in the values gives

$$\begin{aligned} &d^L_{\min}(\tau) \\ &\leq \frac{(d+1)|W_t| + 2\,|W_0 \setminus W_t| + 2\,|W_t| + |W_t| + |Y| + 2\,|W_0| + 6\,|W_0 \setminus W_t| - 2}{|W_t|}. \end{aligned}$$

Finally, noting that $W_0 = S^L_{\leq(d+1)}$ and $W_t \supseteq S^L_{\leq d}$, we obtain

$$d^L_{\min}(\tau) \leq (d+1) + 10\frac{\left|S^L_{\leq(d+1)}\right|}{\left|S^L_{\leq d}\right|} + \frac{|Y|}{\left|S^L_{\leq d}\right|} - 6.$$

□

4.5.3 The MstdbQ algorithm

Our quasipolynomial MstdbQ algorithm is similar to the polynomial one, except that we use the better subroutines described above, and consequently use different parameters. Each phase employs algorithms MaxdmstQ and MindmstQ to improve either a high-degree vertex or a low-degree vertex in L; when both improvements fail, their failure is justified by two combinatorial witnesses.

Each phase of MstdbQ begins by picking a δ such that

$$\left|S^L_{\leq(B_L-\delta+1)}\right| \leq \frac{\log n}{\log\log n}\left|S^L_{\leq(B_L-\delta)}\right| \quad \text{and} \quad \left|S_{\geq(B_H+\delta-1)}\right| \leq \frac{\log n}{\log\log n}\left|S_{\geq(B_H+\delta)}\right|.$$

It is easy to show that one can always find such a δ in any range of length at least $2\frac{\log n}{\log\log n}$, and hence in particular, between

$$\max\{d_{\max}(T) - B_H, B_L - d^L_{\min}(T)\} - 2\frac{\log n}{\log\log n}$$

and

$$\max\{d_{\max}(T) - B_H, B_L - d^L_{\min}(T)\}.$$

Without loss of generality, $\delta > 0$.

For the rest of the phase, vertices with degree at least $B_H + \delta$ are considered "high-degree" vertices and those in L with degree at most $B_L - \delta$ are considered "low-degree". We employ MaxdmstQ and MindmstQ to reduce the degree of a high-degree vertex and increase the degree of a low-degree vertex respectively. As alluded to earlier, we need to ensure that the improvements they perform do not interfere with each other. For this purpose, we disallow the MaxdmstQ algorithm from removing edges incident on $X = S^L_{\leq(B_L-\delta+1)}$, and disallow the MindmstQ algorithm from adding edges to $Y = S_{\geq(B_H+\delta-1)}$. Each subroutine either improves the potential significantly

Algorithm MstdbQ(G, L, B_L, B_H)

Start with arbitrary minimum spanning tree T.
repeat
Compute δ so that $\left|S^L_{\leq(B_L-\delta+1)}\right| \leq \frac{\log n}{\log\log n}\left|S^L_{\leq(B_L-\delta)}\right|$
and $\left|S_{\geq(B_H+\delta-1)}\right| \leq \frac{\log n}{\log\log n}\left|S_{\geq(B_H+\delta)}\right|$.
Call MaxdmstP with $d = B_H + \delta$ and $X = S^L_{\leq(B_L-\delta+1)}$.
Call MindmstP with $d = B_L - \delta$ and $Y = S_{\geq(B_H+\delta-1)}$.
until both calls fail.

Figure 4.12: Pseudo-code for MstdbQ.

or returns a combinatorial witness. The algorithm terminates if one of the following happen: the algorithm finds an MST with the required degree guarantees, or both subroutines output combinatorial witnesses on a particular tree T. See Figure 4.12 for a formal description of the algorithm.

Lemma 4.16 guarantees that this is enough to make the algorithm terminate. Theorem 4.17 shows that when both MaxdmstQ and MindmstQ fail with two combinatorial witnesses, at least one of the witnesses is good.

Lemma 4.16. *Algorithm* MstdbQ *terminates in quasi-polynomial time.*

Proof. Let T be the tree at the beginning of a particular phase and T' be the tree at the end of the phase. At the end of this phase, at least one of the following is true:

(i) T' has a potential function value at most $\phi_Q(T) - (2n)^{\delta-1}$.

(ii) MaxdmstQ and MindmstQ both output combinatorial witnesses and the algorithm terminates.

The first step happens when one or both of MaxdmstQ and MindmstQ succeed; in this case Theorems 4.11 and 4.15 imply the claimed decrease in potential. Moreover,

note that

$$\delta \geq \max\{d_{\max}(T) - B_H, B_L - d^L_{\min}(T)\} - 2\frac{\log n}{\log\log n}$$

so that

$$\phi(T) \leq n \cdot (2n)^{1+2\frac{\log n}{\log\log n}}(2n)^{\delta-1}.$$

Thus the decrease in potential is at least $\phi_Q(T)/(n^{O(\frac{\log n}{\log\log n})})$ in each phase, and the potential function decreases by half after $(n^{O(\frac{\log n}{\log\log n})})$ phases. Since the initial potential is at most exponential in n, the algorithm terminates in quasi-polynomial time. □

4.6 MBDST: A quasi-polynomial-time algorithm

Theorem 4.17. *Given a gragh G, and a degree bound B, there is a quasi-polynomial time algorithm that computes a spanning tree T with cost at most $c_{\text{opt}_B}(G)$ and degree at most $B + O(\frac{\log n}{\log\log n})$.*

Proof. For a fixed B', suppose that we compute $c^{\lambda^{B'}}$ and set L to be set of nodes with a positive λ_u value. Thus we have a fractional spanning tree τ that has maximum degree $d_{\max}(\tau)$ at most B' and $d^L_{\min}(\tau)$ at least B'. Executing $\mathsf{MstdbQ}(G, L, B', B')$ on this cost function produces a spanning tree T and witnesses that certify that

$$B' \geq d_{\max}(\tau) \geq B' + \delta - O\left(\frac{\log n}{\log\log n}\right) - \frac{\left|S^L_{\leq(B'-\delta+1)}\right|}{\left|S_{\geq(B'+\delta-1)}\right|}$$

and

$$B' \leq d^L_{\min}(\tau) \leq B' - \delta + O\left(\frac{\log n}{\log\log n}\right) + \frac{\left|S_{\geq(B'+\delta-1)}\right|}{\left|S^L_{\leq(B'-\delta)}\right|}.$$

By our choice of δ, at least one of the following inequalities holds:

$$2\frac{\left|S^L_{\leq(B'-\delta+1)}\right|}{\left|S_{\geq(B'+\delta-1)}\right|} < \frac{\log n}{\log\log n}$$

$$\frac{\left|S_{\geq(B'+\delta-1)}\right|}{\left|S^L_{\leq(B'-\delta)}\right|} < \frac{\log n}{\log\log n}.$$

This implies that $\delta \leq O(\frac{\log n}{\log\log n})$. However,

$$\delta \geq \max\{d_{\max}(T) - B', B' - d^L_{\min}(T)\} - 2\frac{\log n}{\log\log n},$$

so that $d_{\max}(T) \leq B' + O(\frac{\log n}{\log\log n})$ and $d^L_{\min}(T) \geq B' - O(\frac{\log n}{\log\log n})$.

Choosing $B' = B + O(\frac{\log n}{\log\log n})$, this means that we have computed a tree with $d^L_{\min}(T) \geq B$ and $d_{\max}(T) \leq B + O(\frac{\log n}{\log\log n})$. Using Theorem 2.3, we get a tree of cost at most $\text{opt}_{LD(B)}$ and degree at most $B + O(\frac{\log n}{\log\log n})$. □

Part II

Sorting and selection in posets

Chapter 5

Sorting in posets

In this chapter, we present our results for the problem of sorting a partially ordered set.

We begin by reviewing basic terminology needed for this chapter and Chapter 6. In Section 5.3, we briefly discuss an efficient representation of a poset. In Section 5.5 we give an algorithm of optimal query complexity. An efficient algorithm based on Mergesort is described in Section 5.6. In Section 5.7, we give a randomized algorithm, based on a generalization of Quicksort, for computing a linear extension of a poset. We also give a randomized algorithm for computing the heights of all elements in a poset. Finally, in Section 5.8, we show that the results on sorting posets generalize to the case when an upper bound on the width is not known and to the case of transitive relations.

5.1 Preliminary definitions

To precisely describe the problems considered in this chapter and our results, we require some formal definitions. A *partially ordered set*, or *poset*, is a pair $\mathcal{P} = (P, \succ)$, where P is a set of elements and $\succ \subset P \times P$ is an irreflexive, transitive binary relation.

For elements $a, b \in P$, if $(a, b) \in \succ$, we write $a \succ b$ and we say that a *dominates* b, or that b is *smaller than* a. If $a \not\succ b$ and $b \not\succ a$, we say that a and b are *incomparable* and write $a \not\sim b$.

A *chain* $C \subseteq P$ is a subset of mutually comparable elements, that is, a subset such that for any elements $c_i, c_j \in C$, $i \neq j$, either $c_i \succ c_j$ or $c_j \succ c_i$. An *ideal* $I \subseteq P$ is a subset of elements such that if $x \in I$ and $x \succ y$, then $y \in I$. The *height* of an element a is the maximum cardinality of a chain whose elements are all dominated by a. We call the set $\{a : \forall b,\ b \succ a \text{ or } b \not\sim a\}$ of elements of height 0 the *minimal* elements. An *anti-chain* $A \subseteq P$ is a subset of mutually incomparable elements. The *width* $w(\mathcal{P})$ of poset $\mathcal{P}$ is defined to be the maximum cardinality of an anti-chain of $\mathcal{P}$.

A *decomposition* $\mathcal{C}$ of $\mathcal{P}$ into chains is a family $\mathcal{C} = \{C_1, C_2, \ldots, C_q\}$ of disjoint chains such that their union is P. The *size* of a decomposition is the number of chains in it. The width $w(\mathcal{P})$ is clearly a lower bound on the size of any decomposition of $\mathcal{P}$. We make frequent use of *Dilworth's Theorem*, which states that there is a decomposition of $\mathcal{P}$ of size $w(\mathcal{P})$. A decomposition of size $w(\mathcal{P})$ is called a *minimum chain decomposition*.

5.2 The sorting problem

The central computational problem of this chapter is to produce a representation of a poset $\mathcal{P} = (P, \succ)$, given the set P of n elements, an upper bound of w on the width of $\mathcal{P}$, and access to an oracle for $\mathcal{P}$.

In the absence of a bound on the width, the query complexity of the sorting problem is exactly $\binom{n}{2}$, in view of the worst-case example in which all pairs of elements are incomparable. In the classical sorting problems, $w = 1$. Our interest is mainly in the case where $w \ll n$, since this assumption is natural in many of the applications.

Furthermore, if w is of the same order as n, then it is easy to see that the complexity of sorting is of order n^2, as in the case where no restrictions are imposed on the poset.

We discuss the case when an upper bound on the width is not known a priori in Section 5.8.1.

5.2.1 Related work

Faigle and Turán [13] have described two algorithms for sorting posets, both of which have query complexity $O\left(wn \log \frac{n}{w}\right)$. (In fact the second algorithm is shown to have query complexity $O(n \log N_{\mathcal{P}})$, where $N_{\mathcal{P}}$ is the number of ideals in input poset $\mathcal{P}$. It is easy to see that $N_{\mathcal{P}} = O(n^w)$ if $\mathcal{P}$ has width w, and that $N_{\mathcal{P}} = (n/w)^w$ if $\mathcal{P}$ consists of w incomparable chains, each of size n/w.) The total complexity of sorting posets has not been considered. However, the total complexity of the first algorithm given by Faigle and Turán depends on the subroutine for computing a chain decomposition (the complexity of which is not analyzed in [13]). It is not clear if there exists a polynomial-time implementation of the second algorithm.

5.2.2 Our techniques

It is natural to approach the problem of sorting in posets by considering generalizations of the well-known algorithms for the case of total orders, whose running times are closely matched by proven lower bounds. Somewhat surprisingly, natural generalizations of the classic algorithms *do not* provide optimal poset algorithms in terms of total and query complexity.

The generalization of Mergesort considered in Section 5.6 loses a factor of w in its total complexity compared to the information-theoretic lower bound. Interestingly, one can achieve the information-theoretic lower bound on query complexity (up to a constant factor) by carefully exploiting the structure of the poset. In Section 5.5, we

describe at length the techniques used obtain this result. We do not know whether it is possible to achieve the information-theoretic bound on total complexity.

5.3 Representing a poset

Once the relation between every pair of elements in a poset has been determined, some representation of this information is required, both for output and for use in our algorithms. The simple ChainMerge data structure that we describe here supports constant-time look-ups of the relation between any pair of elements. It is built from a chain decomposition of the poset.

Let $\mathcal{C} = \{C_1, \ldots C_q\}$ be a chain decomposition of a poset $\mathcal{P} = (P, \succ)$. The data structure ChainMerge$(\mathcal{P}, \mathcal{C})$ stores, for each element $x \in P$, q indices as follows: Let C_i be the chain of $\mathcal{C}$ containing x. The data structure stores the index of x in C_i and, for all j, $1 \leq j \leq q$, $j \neq i$, the index of the largest element of chain C_j that is dominated by x. The performance of the data structure is characterized by the following lemma.

Claim 5.1. *Given a query oracle for a poset $\mathcal{P} = (P, \succ)$ and a decomposition $\mathcal{C}$ of $\mathcal{P}$ into q chains, building the* ChainMerge *data structure has query complexity at most $2qn$ and total complexity $O(qn)$, where $n = |P|$. Given* ChainMerge$(\mathcal{P}, \mathcal{C})$*, the relation in $\mathcal{P}$ of any pair of elements can be found in constant time.*

Proof. The indices corresponding to chain C_j that must be stored for the elements in chain C_i can be found in $O(|C_i|+|C_j|)$ time, using $|C_i|+|C_j|$ queries, by simultaneously scanning C_i and C_j. Since each chain is scanned $2q - 1$ times, it follows that the query complexity of ChainMerge$(\mathcal{P}, \mathcal{C})$ is at most $2qn$, and the total complexity is $O(q \cdot \sum_{i=1}^{q} |C_i|) = O(qn)$.

Let $x, y \in P$, with $x \in C_i$ and $y \in C_j$. The look-up operation works as follows: If $i = j$, we simply do a comparison on the indices of x and y in C_i, as in the case of a

total order. If $i \neq j$, then we look up the index of the largest element of C_j that is dominated by x; this index is greater than (or equal to) the index of y in C_j if and only if $x \succ y$. If $x \not\succ y$, then we look up the index of the largest element of C_i that is dominated y; this index is greater than (or equal to) the index of x in C_i if and only if $y \succ x$. If neither $x \succ y$ nor $y \succ x$, then $x \not\sim y$. □

5.4 A lower bound

An information-theoretic lower bound on the query complexity of sorting is implied by the following theorem of Brightwell and Goodall [2], which provides a lower bound on the number $N_w(n)$ of posets of width at most w on n elements.

Theorem 5.2. *The number $N_w(n)$ of partially ordered sets of n elements and width at most w satisfies*

$$\frac{n!}{w!}\ 4^{n(w-1)}\ n^{-24w(w-1)} \leq N_w(n) \leq n!\ 4^{n(w-1)}\ n^{-(w-2)(w-1)/2} w^{w(w-1)/2}.$$

It follows that, for $w = o\left(\frac{n}{\log n}\right)$,

$$\log N_w(n) = \Theta(n \log n + wn).$$

5.5 On optimal query complexity

In this section, we describe a sorting algorithm that has optimal query complexity, i.e. it sorts a poset of width at most w on n elements using $\Theta(n \log n + wn)$ oracle queries. Our algorithm is not necessarily computationally efficient, so in Section 5.6, we consider efficient solutions to the problem.

5.5.1 Other approaches

Before presenting our algorithm, it is worth discussing an intuitive approach that is different from the one we take. For any set of oracle queries and responses, there is a corresponding set of posets, which we call *candidates*, that are the posets consistent with the responses to these queries. A natural sorting algorithm is to find a sequence of oracle queries such that, for each query (or for a positive fraction of the queries), the possible responses to it partition the space of posets that are candidates after the previous queries into three parts, at least two of which are relatively large. Such an algorithm would achieve the information-theoretic lower bound (up to a constant).

For example, the effectiveness of Quicksort for sorting total orders relies on the fact that most of the queries made by the algorithm partition the space of candidate total orders into two parts, each of relative size of at least 1/4. Indeed, in the case of total orders, much more is known: for any subset of queries, there is a query that partitions the space of candidate total orders, i.e. linear extensions, into two parts, each of relative size of at least 3/11 [29].

In the case of width-w posets, however, it could potentially be the case that most queries partition the space into three parts, one of which is much larger than the other two. For example, if the set consists of w incomparable chains, each of size n/w, then a random query has a response of incomparability with probability about $1 - 1/w$. (On an intuitive level, this explains the extra factor of w in the query complexity of our version of Mergesort, given in Section 5.6.) Hence, we resort to more elaborate sorting strategies.

5.5.2 A building block

Our optimal algorithm builds upon Poset–BinInsertionSort, a basic algorithm that is identical to "Algorithm A" of Faigle and Turán [13]. The algorithm is inspired

by the binary insertion-sort algorithm for total orders. Pseudocode for Poset–BinInsertionSort is presented in Figure 5.1. The natural idea behind Poset–BinInsertionSort is to sequentially insert elements into a subset of the poset, while maintaining a chain decomposition of the latter into a number of chains equal to the width w of the poset to be constructed. A straightforward implementation of this idea is to perform a binary search on every chain of the decomposition in order to figure out the relationship of the element being inserted with every element of that chain and, ultimately, with all the elements of the current poset. It turns out that this simple algorithm is not optimal; it is off by a factor of w from the optimum. In the rest of this section, we show how to adapt Poset–BinInsertionSort to achieve the information-theoretic lower bound.

We begin by analyzing Poset–BinInsertionSort.

Lemma 5.3 (Faigle & Turán [13])**.** Poset–BinInsertionSort *sorts any partial order $\mathcal{P}$ of width at most w on n elements using at most $O(wn \log n)$ oracle queries.*

Proof. The correctness of Poset–BinInsertionSort should be clear from its description. (The simple argument showing that Step 4e can be executed based on the information obtained in Step 4d is similar to the proof for the ChainMerge data structure in Section 5.3.) It is not hard to see that the number of oracle queries incurred by Poset–BinInsertionSort for inserting each element is $O(w \log n)$ and, therefore, the total number of queries is $O(wn \log n)$. □

It follows that, as n scales, the number of queries incurred by the algorithm is more by a factor of w than the lower bound. The Achilles' heel of the Poset-BinInsertionSort algorithm is in the method of insertion of an element—specifically, in the way the binary searches of Step 4d are performed. In these sequences of queries, no structural properties of $\mathcal{P}'$ are used for deciding which queries to the oracle are more useful than others; in some sense, the binary searches give the same "attention"

Algorithm POSET–BININSERTIONSORT($\mathcal{P}$)
input: a set P, a query oracle for a poset $\mathcal{P} = (P, \succ)$,
and upper bound w on the width of $\mathcal{P}$
output: a CHAINMERGE data structure for $\mathcal{P}$

1. $\mathcal{P}' := (\{e\}, \{\})$, where $e \in P$ is some arbitrary element;
/* $\mathcal{P}'$ is the current poset*/
2. $P' := \{e\}$; $\mathcal{R}' := \{\}$;
3. $U := P \setminus \{e\}$;
/* U is the set of elements that have not been inserted */
4. **while** $U \neq \emptyset$
a. pick an arbitrary element $e \in U$;
/* e is the element that will be inserted in $\mathcal{P}'$*/
b. $U := U \setminus \{e\}$;
c. find a chain decomposition $\mathcal{C} = \{C_1, C_2, \ldots, C_q\}$ of $\mathcal{P}'$, for $q \leq w$;
d. **for** $i = 1, \ldots, q$
i. let $C_i = \{e_{i1}, \ldots, e_{i\ell_i}\}$, where $e_{i\ell_i} \succ \ldots \succ e_{i2} \succ e_{i1}$;
ii. do binary search on C_i to find
the smallest element (if any) that dominates e;
iii. do binary search on C_i to find
the largest element (if any) that is dominated by e;
e. based on results of binary searches,
infer all relations of e with elements of P';
f. add into $\mathcal{R}'$ all the relations of e with the elements of P';
$P' := P' \cup \{e\}$;
g. $\mathcal{P}' = (P', \mathcal{R}')$;
5. find a decomposition $\mathcal{C}$ of $\mathcal{P}'$; build CHAINMERGE($\mathcal{P}', \mathcal{C}$)
(no additional queries needed);
6. **return** CHAINMERGE($\mathcal{P}', \mathcal{C}$);

Figure 5.1: Pseudo-code for POSET–BININSERTIONSORT.

to queries whose answer would greatly decrease the number of remaining possibilities and those whose answer is not very informative. However, as we discuss earlier in this section, a sorting algorithm that always makes the most informative query is not guaranteed to be optimal.

Our algorithm tries to resolve this dilemma. We suggest a scheme that has the same structure as the Poset–BinInsertionSort algorithm but exploits the structure of the already constructed poset $\mathcal{P}'$ in order to amortize the cost of the queries over the insertions. The amortized query cost matches the information-theoretic bound.

5.5.3 The EntropySort algorithm

The new algorithm, named EntropySort, modifies the binary searches of Step 4d into weighted binary searches. The weights assigned to the elements satisfy the following property: the number of queries it takes to insert an element into a chain is proportional to the number of candidate posets that will be eliminated after the insertion of the element. In other words, we spend fewer queries for insertions that are not informative and more queries for insertions that are informative. In some sense, this corresponds to an *entropy-weighted binary search*. To make this notion precise, we use the following definition.

Definition. Suppose that $\mathcal{P}' = (P', \mathcal{R}')$ is a poset of width at most w, U a set of elements such that $U \cap P' = \emptyset$, $u \in U$ and $\mathcal{ER}, \mathcal{PR} \subseteq (\{u\} \times P') \cup (P' \times \{u\})$. We say that $\mathcal{P} = (P' \cup U, \mathcal{R})$ is *a width w extension of $\mathcal{P}'$ on U conditioned on* $(\mathcal{ER}, \mathcal{PR})$, if $\mathcal{P}$ is a poset of width w, $\mathcal{R} \cap (P' \times P') = \mathcal{R}'$ and, moreover, $\mathcal{ER} \subseteq \mathcal{R}$, $\mathcal{R} \cap \mathcal{PR} = \emptyset$. In other words, $\mathcal{P}$ is an extension of $\mathcal{P}'$ on the elements of U which is consistent with $\mathcal{P}'$, it contains the relations of u to P' given by $\mathcal{ER}$ and does not contain the relations of u to P' given by $\mathcal{PR}$. The set $\mathcal{ER}$ is then called the *set of enforced relations* and

the set $\mathcal{PR}$ the *set of prohibited relations*.

We give in Figure 5.2 the pseudocode of Step 4d′ of EntropySort, which replaces Step 4d of Poset–BinInsertionSort.

The correctness of the EntropySort algorithm follows trivially from the correctness of Poset–BinInsertionSort. We prove next that its query complexity is optimal. Recall that $N_w(n)$ denotes the number of partial orders of width w on n elements.

Theorem 5.4. EntropySort *sorts any partial order $\mathcal{P}$ of width at most w on n elements using at most*

$$2 \log N_w(n) + 4wn = \Theta(n \log n + wn)$$

oracle queries. In particular, the query complexity of the algorithm is at most

$$2n \log n + 8wn + 2w \log w.$$

Proof. We first characterize the number of oracle calls required by the weighted binary searches.

Lemma 5.5 (Weighted Binary Search)**.** *For every $j \in \{1, 2, \ldots, \ell_i + 1\}$, if e_{ij} is the smallest element of chain C_i which dominates element e ($j = \ell_i + 1$ corresponds to the case where no element of chain C_i dominates e), then j is found after at most $2 \cdot (1 + \log \frac{\mathcal{D}_i}{\mathcal{D}_{ij}})$ oracle queries in Step* v. *of the algorithm described above.*

Proof. Let $\lambda = \frac{\mathcal{D}_{ij}}{\mathcal{D}_i}$ be the length of the interval that corresponds to e_{ij}. We wish to prove that the number of queries needed to find e_{ij} is at most $2(1 + \lfloor \log \frac{1}{\lambda} \rfloor)$. From the definition of the weighted binary search, we see that if the interval corresponding to e_{ij} contains a point of the form $2^{-r} \cdot m$ in its interior, where r, m are integers, then

Step 4d′ for Algorithm EntropySort($\mathcal{P}$)

4d′. $\mathcal{ER} = \emptyset$; $\mathcal{PR} = \emptyset$;
for $i = 1, \ldots, q$
i. let $C_i = \{e_{i1}, \ldots, e_{i\ell_i}\}$, where $e_{i\ell_i} \succ \ldots \succ e_{i2} \succ e_{i1}$;
ii. for $j = 1, \ldots, \ell_i + 1$
• set $\mathcal{ER}_j = \{(e_{ik}, e) | j \leq k \leq \ell_i\}$; set $\mathcal{PR}_j = \{(e_{ik}, e) | 1 \leq k < j\}$;
• compute $\mathcal{D}_{ij}$ = # w-extensions of $\mathcal{P}'$ on U, conditioned on $(\mathcal{ER} \cup \mathcal{ER}_j, \mathcal{PR} \cup \mathcal{PR}_j)$;
/* $\mathcal{D}_{ij}$ is # posets on P consistent with $\mathcal{P}'$, ($\mathcal{ER}$, $\mathcal{PR}$), in which e_{ij} is smallest element dominating e in C_i; $j = \ell_i + 1$ is case that no element of C_i dominates e;*/
iii. set $\mathcal{D}_i = \sum_{j=1}^{\ell_i+1} \mathcal{D}_{ij}$;
/* $\mathcal{D}_i$ is # of w-extensions of $\mathcal{P}'$ on U conditioned on $(\mathcal{ER}, \mathcal{PR})$*/
iv. partition unit interval $[0, 1)$ into $\ell_i + 1$ intervals $([b_j, t_j))_{j=1}^{\ell_i+1}$, for $b_1 = 0$, $b_j = t_{j-1}$, $\forall j \geq 2$, and $t_j = (\sum_{j' \leq j} \mathcal{D}_{ij'})/\mathcal{D}_i$, $\forall j \geq 1$.
/* an interval $\leftrightarrow$ an element of C_i, or a "dummy" element $e_{i\ell_i+1}$ */
v. binary search on $[0, 1)$ for smallest element dominating e in C_i :
/* weighted version of binary search in Step 4dii of Figure 5.1 */
set $x = 1/2$; $t = 1/4$; $j^* = 0$;
repeat: find j such that $x \in [b_j, t_j)$;
if ($j = \ell_i + 1$ **and** $e_{i,j-1} \not\succ e$) **OR** ($e_{ij} \succ e$ **and** $j = 0$) **OR** ($e_{ij} \succ e$ **and** $e_{i,j-1} \not\succ e$)
set $j^* = j$; break; /* found it */
else if ($j = \ell_i + 1$) **OR** ($e_{ij} \succ e$)
set $x = x - t$; $t = t * 1/2$; /* look below */
else
set $x = x + t$; $t = t * 1/2$; /* look above */
vi. e_{ij^*} is smallest element dominating e in C_i;
set $\mathcal{ER} := \mathcal{ER} \cup \mathcal{ER}_{j^*}$ and $\mathcal{PR} := \mathcal{PR} \cup \mathcal{PR}_{j^*}$;
vii. find largest element (if any) dominated by e in C_i:
for $j = 0, 1, \ldots, \ell_i$,
compute $\mathcal{D}'_{ij}$ = # posets on P consistent with $\mathcal{P}'$, ($\mathcal{ER}$, $\mathcal{PR}$), in which e_{ij} is largest element dominated by e in C_i;
/* $j = 0$ corresponds to case no element of C_i dominated by e; */
let $\mathcal{D}'_i = \sum_{j=0}^{\ell_i} \mathcal{D}'_{ij}$;
do weighted binary search analogous to that of Step v;
viii. update accordingly the sets $\mathcal{ER}$ and $\mathcal{PR}$;

Figure 5.2: To obtain Algorithm EntropySort, substitute Step 4d′ above for Step 4d in Figure 5.1.

the search reaches e_{ij} after at most r steps. Now, an interval of length λ must include a point of the form $2^{-r} \cdot m$, where $r = 1 + \lfloor \log \frac{1}{\lambda} \rfloor$, which concludes the proof. $\square$

It is important to note that the number of queries spent by the weighted binary search is small for uninformative insertions, which correspond to large $\mathcal{D}_{ij}$'s, and large for informative ones, which correspond to small $\mathcal{D}_{ij}$'s. Hence, our use of the term entropy-weighted binary search. A parallel of Lemma 5.5 holds, of course, for finding the largest element of chain C_i dominated by element e.

Suppose now that $P = \{e_1, \ldots, e_n\}$, where $e_1, e_2, \ldots, e_n$ is the order in which the elements of P are inserted into poset $\mathcal{P}'$. Also, denote by $\mathcal{P}_k$ the restriction of poset $\mathcal{P}$ onto the set of elements $\{e_1, e_2, ..., e_k\}$ and by Z_k the number of width w extensions of poset $\mathcal{P}_k$ on $P \setminus \{e_1, \ldots, e_k\}$ conditioned on $(\emptyset, \emptyset)$. Clearly, $Z_0 \equiv N_w(n)$ and $Z_n = 1$. The following lemma is sufficient to establish the optimality of EntropySort.

Lemma 5.6. EntropySort *needs at most* $4w + 2\log \frac{Z_k}{Z_{k+1}}$ *oracle queries to insert element* e_{k+1} *into poset* $\mathcal{P}_k$ *in order to obtain* $\mathcal{P}_{k+1}$.

Proof. Let $\mathcal{C} = \{C_1, \ldots, C_q\}$ be the chain decomposition of the poset $\mathcal{P}_k$ constructed at Step 4c of EntropySort in the iteration of the algorithm in which element e_{k+1} needs to be inserted into poset $\mathcal{P}_k$. Suppose also that, for all $i \in \{1, \ldots, q\}$, $\pi_i \in \{1, \ldots, \ell_i + 1\}$ and $\kappa_i \in \{0, 1, \ldots, \ell_i\}$ are the indices computed by the binary searches of Steps v. and vii. of the algorithm. Also, let $\mathcal{D}_i$, $\mathcal{D}_{ij}$, $j \in \{1, \ldots, \ell_i + 1\}$, and $\mathcal{D}'_i$, $\mathcal{D}'_{ij}$, $j \in \{0, \ldots, \ell_i\}$, be the quantities computed at Steps ii., iii. and vii. It is not hard to see that the following are satisfied

$$Z_k = \mathcal{D}_1 \qquad \mathcal{D}'_{q\kappa_q} = Z_{k+1}$$
$$\mathcal{D}_{i\pi_i} = \mathcal{D}'_i, \forall i = 1, \ldots, q \qquad \mathcal{D}'_{i\kappa_i} = \mathcal{D}_{i+1}, \forall i = 1, \ldots, q-1$$

Now, using Lemma 5.5, it follows that the total number of queries required to con-

struct $\mathcal{P}_{k+1}$ from $\mathcal{P}_k$ is at most

$$\sum_{i=1}^{q} \left(2 + 2\log \frac{\mathcal{D}_i}{\mathcal{D}_{i\pi_i}} + 2 + 2\log \frac{\mathcal{D}'_i}{\mathcal{D}'_{i\kappa_i}} \right) \leq 4w + 2\log \frac{Z_k}{Z_{k+1}}.$$

□

Using Lemma 5.6, the query complexity of EntropySort is

$$\begin{aligned} &\sum_{k=0}^{n-1} (\#\text{ queries needed to insert element } e_{k+1}) \\ &\quad = \sum_{k=0}^{n-1} \left(4w + 2\log \frac{Z_k}{Z_{k+1}} \right) \\ &\quad = 4wn + 2\log \frac{Z_0}{Z_n} = 4wn + 2\log N_w(n). \end{aligned}$$

Taking the logarithm of the upper bound in Theorem 5.2, it follows that the number of queries required by the algorithm is $2n\log n + 8wn + 2w\log w$.

□

5.6 An efficient sorting algorithm

In this section, we turn to the problem of efficient sorting. Superficially, the Poset-Mergesort algorithm that we present has a recursive structure that is similar to the classical Mergesort algorithm. The merge step is quite different, however; it makes crucial use of the technical Peeling algorithm in order to efficiently maintain a small chain decomposition of the poset throughout the recursion. The Peeling algorithm, described formally in Section 5.6.2, is a specialization of the classic flow-based bipartite-matching algorithm [37] that is efficient in the comparison model.

5.6.1 Algorithm Poset-Mergesort

Given a set P, a query oracle for a poset $\mathcal{P} = (P, \succ)$, and an upper bound w on the width of $\mathcal{P}$, the Poset-Mergesort algorithm produces a decomposition of $\mathcal{P}$ into w chains and concludes by building a ChainMerge data structure. To get the chain decomposition, the algorithm partitions the elements of P arbitrarily into two subsets of (as close as possible to) equal size; it then finds a chain decomposition of each subset recursively. The recursive call returns a decomposition of each subset into at most w chains, which constitutes a decomposition of the whole set P into at most $2w$ chains. Then the Peeling algorithm of Section 5.6.2 is applied to reduce the decomposition to a decomposition of w chains. Given a decomposition of $P' \subseteq P$, where $m = |P'|$, into at most $2w$ chains, the Peeling algorithm returns a decomposition of P' into w chains using $2wm$ queries and $O(w^2 m)$ time. Figure 5.3 shows pseudo-code for Poset-Mergesort.

Theorem 5.7. Poset-Mergesort *sorts any poset $\mathcal{P}$ of width at most w on n elements using at most*

$$2wn \log(n/w))$$

queries, with total complexity

$$O(w^2 n \log(n/w)).$$

Proof. The correctness of Poset-Mergesort is immediate. Let $T(m)$ and $Q(m)$ be the worst-case total and query complexity, respectively, of the procedure Poset-Mergesort-Recurse on a poset of width w containing m elements. When $m \leq w$, $T(m) = O(w)$ and $Q(m) = 0$. When $m > w$, $T(m) = 2T(m/2) + O(w^2 m)$ and $Q(m) \leq 2Q(m/2) + 2wm$. Therefore, $T(n) = O(w^2 n \log(n/w))$ and $Q(n) \leq 2wn \log(n/w)$. The cost incurred by the last step of the algorithm, i.e. that of

Algorithm Poset-Mergesort($\mathcal{P}$)
input: a set P, a query oracle for a poset $\mathcal{P} = (P, \succ)$,
and upper bound w on the width of $\mathcal{P}$
output: a ChainMerge data structure for $\mathcal{P}$

run Poset-Mergesort-Recurse(P),
producing a decomposition $\mathcal{C}$ of $\mathcal{P}$ into w chains;
build and **return** ChainMerge($\mathcal{P}, \mathcal{C}$);

Procedure Poset-Mergesort-Recurse(P')
input: a subset $P' \subseteq P$, a query oracle for $\mathcal{P} = (P, \succ)$,
an upper bound w on the width of $\mathcal{P}$
output: a decomposition into at most w chains
of the poset $\mathcal{P}'$ induced by $\succ$ on P'

if $|P'| \leq w$
then return the trivial decomposition of $\mathcal{P}'$ into chains of length 1
else
1. partition P' into two parts of equal size, P'_1 and P'_2;
2. run Poset-Mergesort-Recurse(P'_1)
and Poset-Mergesort-Recurse(P'_2);
3. collect the outputs to get a decomposition $\mathcal{C}$ of $\mathcal{P}'$
into $q \leq 2w$ chains;
4. if $q > w$, run Peeling($\mathcal{P}, \mathcal{C}$),
to get a decomposition $\mathcal{C}'$ of $\mathcal{P}'$ into w chains;
return $\mathcal{C}'$;

Figure 5.3: Pseudo-code for Poset-Mergesort.

building the ChainMerge, is negligible. □

5.6.2 The Peeling algorithm

In this section we present an algorithm that efficiently reduces the size of a given decomposition of a poset. It can be seen as an adaptation of the classic flow-based bipartite-matching algorithm [37] that is designed to be efficient in the oracle model. The Peeling algorithm is given an oracle for poset $\mathcal{P} = (P, \succ)$, where $n = |P|$, and a decomposition of P into at most $2w$ chains. It first builds a ChainMerge data structure using at most $2qn$ queries and time $O(qn)$. Every query the algorithm makes after that is actually a look-up in the data structure and therefore takes constant time and no oracle call.

The Peeling algorithm proceeds in a number of *peeling iterations*. Each iteration produces a decomposition of $\mathcal{P}$ with one less chain, until after at most w peeling iterations, a decomposition of $\mathcal{P}$ into w chains is obtained. A detailed formal description of the algorithm is given in Figure 5.4.

Theorem 5.8. *Given an oracle for $\mathcal{P} = (P, \succ)$, where $n = |P|$, and a decomposition of $\mathcal{P}$ into at most $2w$ chains, the* Peeling *algorithm returns a decomposition of $\mathcal{P}$ into w chains. It has query complexity at most $2wn$ and total complexity $O(w^2 n)$.*

Proof. To prove the correctness of one peeling iteration, we first observe that it is always possible to find a pair (x, y) of top elements such that $y \succ x$, as specified in Step 1a, since the size of any anti-chain is at most the width of $\mathcal{P}$, which is less than the number of chains in the decomposition. We now argue that it is possible to find a subsequence of dislodgements as specified by Step 2a. Let y_t be the element defined in step 3 of the algorithm. Since y_t was dislodged by x_t, x_t was the top element of some list when that happened. In order for x_t to be a top element, it was either top

Algorithm PEELING($\mathcal{P}, \mathcal{C}$)
input: a query oracle for poset $\mathcal{P} = (P, \succ)$,
an upper bound w on the width of $\mathcal{P}$,
and a decomposition $C = \{C_1, \ldots, C_q\}$ of $\mathcal{P}$, where $q \leq 2w$
output: a decomposition of $\mathcal{P}$ into w chains

build CHAINMERGE($\mathcal{P}, \mathcal{C}$); /* All further queries are look-ups. */
for $i = 1, \ldots, q$
construct a linked list for each chain $C_i = e_{i\ell_i} \rightarrow \cdots \rightarrow e_{i2} \rightarrow e_{i1}$,
where $e_{i\ell_i} \succ \cdots \succ e_{i2} \succ e_{i1}$;
while $q > w$, perform a peeling iteration:
1. **for** $i = 1, \ldots, q$, set $C'_i = C_i$;
2. **while** every C'_i is nonempty
/* the largest element of each C'_i is a *top element* */
a. find a pair (x, y), $x \in C'_i$, $y \in C'_j$, of top elements
such that $y \succ x$;
b. delete y from C'_j; /* x *dislodges* y */
3. in the sequence of dislodgements,
find a subsequence $(x_1, y_1), \ldots, (x_t, y_t)$ such that:
- deletion of y_t (in step 2b) created an empty chain;
- for $i = 2, \ldots, t$, y_{i-1} is the parent of x_i in its original chain;
- x_1 is the top element of one of the original chains;

4. modify the original chains $C_1, \ldots, C_q$:
a. **for** $i = 2, \ldots, t$
i. delete the pointer going from y_{i-1} to x_i;
ii. replace it with a pointer going from y_i to x_i;
b. add a pointer going from y_1 to x_1;
5. set $q = q - 1$; re-index the modified original chains from 1 to $q - 1$;
return the current chain decomposition, containing w chains

Figure 5.4: Pseudo-code for the PEELING Algorithm.

from the beginning, or its parent y_{t-1} must have been dislodged by some element x_{t-1}, and so on.

We claim that, given a decomposition into q chains, one peeling iteration produces a decomposition of $\mathcal{P}$ into $q-1$ chains. Recall that $y_1 \succ x_1$ and, moreover, for every i, $2 \leq i \leq t$, $y_i \succ x_i$, and $y_{i-1} \succ x_i$. Observe that after Step 4 of the peeling iteration, the total number of pointers has increased by 1. Therefore, if the link structure remains a union of disconnected chains, the number of chains must have decreased by 1, since 1 extra pointer implies 1 less chain. It can be seen that the switches performed by Step 4 of the algorithm maintain the invariant that the in-degree and out-degree of every vertex is bounded by 1. Moreover, no cycles are introduced since every pointer that is added corresponds to a valid relation. Therefore, the link structure is indeed a union of disconnected chains.

The query complexity of the PEELING algorithm is exactly the query complexity of CHAINMERGE, which is $2wn$. We show next that one peeling iteration can be implemented in time $O(qn)$, which implies the claim.

In order to implement one peeling iteration in time $O(qn)$, a little book-keeping is needed, in particular, for Step 2a. We maintain during the peeling iteration a list L of potentially-comparable pairs of elements. At any time, if a pair (x, y) is in L, then x and y are top elements. At the beginning of the iteration, L consists of all pairs (x, y) where x and y are top elements. Any time an element x that was not a top element becomes a top element, we add to L the set of all pairs (x, y) such that y is currently a top element. Whenever a top element x is dislodged, we remove from L all pairs that contain x. When Step 2a requires us to find a pair of comparable top elements, we take an arbitrary pair (x, y) out of L and check if x and y are comparable. If they are not comparable, we remove (x, y) from L, and try the next pair. Thus, we never compare a pair of top elements more than once. Since each element of P is responsible for inserting at most q pairs to L (when it becomes a top element), it

follows that a peeling iteration can be implemented in time $O(qn)$.

□

5.7 Computing linear extensions and heights

In this section we consider two problems that are closely related to the problem of determining a partial order: given a poset, compute a linear extension, and compute the heights of all elements.

More formally, a total order $(P, >)$ is a *linear extension* of a partial order $(P, \succ)$ if, for any two elements $x, y \in P$, $x \succ y$ implies $x > y$. We give a randomized algorithm that, given a set P of n elements, access to an oracle for a poset $\mathcal{P} = (P, \succ)$, and an upper bound w on the width of $\mathcal{P}$, computes a linear extension of $\mathcal{P}$ with expected total complexity $O(n \log n + wn)$. We give another randomized algorithm that, on the same input, determines the height of every element of $\mathcal{P}$ with expected total complexity $O(wn \log n)$.

The algorithms are analogous to Quicksort, and are based on a *ternary* search tree, an extension of the well-known binary search tree for maintaining elements of a linear order. A ternary search tree for a poset $\mathcal{P} = (P, \succ)$, consists of a root, a left subtree, a middle subtree and a right subtree. The root contains an element $x \in P$ and the left, middle, and right subtrees are ternary search trees for the restrictions of $\mathcal{P}$ to the sets $\{y \,|\, x \succ y\}$, $\{y \,|\, x \not\succ y\}$ and $\{y \,|\, y \succ x\}$, respectively. The ternary search tree for the empty poset consists of a single empty node.

We give a simple randomized algorithm to construct a ternary search tree for $\mathcal{P}$ as follows: The algorithm assigns a random element of P to the root, compares each of the $n-1$ other elements to the element at the root to determine the sets associated with the three children of the root, and then, recursively, constructs a ternary search tree for each of these three sets.

Define the weight of an internal node x of a ternary search tree as the total number of internal nodes in its three subtrees, and the weight of a ternary search tree as the sum of the weights of all internal nodes. Then the number of queries required to construct a ternary search tree is exactly the weight of the tree.

Theorem 5.9. *The expected weight of a ternary search tree for any poset of size n and width w is $O(n \log n + wn)$.*

Proof. Consider the path from the root to a given element x. The number of edges in this path from a parent to a middle subtree is at most w. The expected number of edges from a parent to a left or right subtree is $O(\log n)$ since, at every step along the path, the probability is at least $1/2$ that the sizes of the left and right subtrees differ by at most a factor of 3. It follows that the expected contribution of any element to the weight of the ternary search tree is $w + O(\log n)$. □

Once a ternary search tree for a poset has been constructed, a linear extension can be constructed by a single depth-first traversal of the tree. If x is the element at the root, then the linear extension is the concatenation of the linear extensions of the following four subsets, corresponding to the node and its three subtrees: $\{y \mid x \succ y\}$, $\{x\}$, $\{y \mid x \not\sim y\}$ and $\{y \mid y \succ x\}$. The corollary below follows.

Corollary 5.10. *There is a randomized algorithm that, given a poset of size n and width at most w, computes a linear extension of the poset and has expected total complexity*

$$O(n \log n + wn).$$

The problem of computing a linear extension has many applications. In particular, we show the following:

Lemma 5.11. *There is a deterministic algorithm that, given a linear extension of a poset of size n and width at most w, computes the heights of all elements and has*

total complexity $O(wn \log n)$.

Proof. Let $h(x) = h$ be the height of element x in $(P, \succ)$. Given a linear extension $x_n > \cdots > x_2 > x_1$, it is easy to compute $h(x)$ for each element x by binary search, using the following observation: Let

$$S(i, h) = \{x_j \mid j \leq i, h(x_j) = h\}$$

be the set of elements of index at most i in the linear extension and of height h in $(P, \succ)$. Then $|S(i, h)| \leq w$ (as the elements of $S(i, h)$ are pairwise incomparable), and $h(x_{i+1}) > h$ if and only if there exists $x \in S(i, h)$ such that $x_{i+1} \succ x$. Thus, given the sets $S(i, h)$, for all h, we can determine $h(x_{i+1})$ and the sets $S(i + 1, h)$, for all h, in time $O(w \log i)$ using binary search. Summing over i yields the claim. □

Combining the algorithms of Corollary 5.10 and Lemma 5.11 gives our final result:

Corollary 5.12. *There is a randomized algorithm that, given a poset of size* n *and width at most* w*, determines the heights of all elements and has expected total complexity* $O(wn \log n)$.

5.8 Variants of the poset model

In this section, we discuss sorting in two variants of the poset model that occur when different restrictions are relaxed. First, we consider posets for which a bound on the width is not known in advance. Second, we allow the irreflexivity condition to be relaxed, which leads to transitive relations. We show that with relatively little overhead in complexity, sorting in either case reduces to the problem of sorting posets.

5.8.1 Unknown width

Recall from Section 5.4 that $N_w(n)$ is the number of posets of width at most w on n elements.

Claim 5.13. *Given a set P of n elements and access to an oracle for poset $\mathcal{P} = (P, \succ)$ of unknown width w, there is an algorithm that sorts P using at most*

$$2 \log w \, (\log N_{2w}(n) + 4wn) = \Theta(n \log w \, (\log n + w))$$

queries. There is an efficient algorithm, of total complexity $O(nw^2 \log w \log(n/w))$, that sorts P using at most $8nw \log w \log(n/(2w))$ queries.

Proof. We use an alternate version of EntropySort that returns fail if it cannot insert an element (while maintaining a decomposition of the given width) and an alternate version of Poset-Mergesort that returns fail if the Peeling algorithm cannot reduce the size of the decomposition to the given width. The first algorithm of the claim is, for $i = 1, 2, \ldots$, to run the alternate version of algorithm EntropySort on input set P, the oracle, and width upper bound 2^i, until the algorithm returns without failing. The second algorithm is analogous but uses the alternate version of Poset-Mergesort. The claim follows from Theorems 5.4 and 5.7, and from the fact that we reach an upper bound of at most $2w$ on the width of $\mathcal{P}$ in $\log w$ rounds. □

5.8.2 Transitive relations and directed graphs

A partial order is a particular kind of transitive relation. In fact, our results generalize to the case of arbitrary transitive relations (which are not necessarily irreflexive) and are therefore relevant to a broader set of applications. Formally, a transitive relation is a pair $(P, \trianglerighteq)$, where P is a set of elements and $\trianglerighteq \subseteq P \times P$ is transitive.

For a directed graph $G = (V, A)$, $A \subseteq \{(u, v) \in V\}$, and vertices $x, y \in V$, x is said to be *reachable* from y if there is a directed path $(y, u_1), \ldots, (u_i, u_{i+1}), \ldots, (u_k, x)$ from y to x in G. The *reachability* relation defined by G is the set of pairs $(x, y) \in V \times V$ such that x is reachable from y in G.

Every transitive relation can be thought of as the reachability relation in a graph, and likewise, the reachability relation of any graph is a transitive relation. Hence, the problem of sorting a transitive relation from queries to an oracle is equivalent to the problem of reconstructing a directed graph from queries to an oracle for its reachability relation.

We show here that the problem of sorting a transitive relation reduces with little overhead in complexity to the problem of sorting a poset.

The *width* of a transitive relation is defined to be the maximum size of a set of mutually incomparable elements. We say that a poset $(P, \succ)$ is *induced* by a transitive relation $(P, \trianglerighteq)$ if $\succ \subseteq \trianglerighteq$. A poset $(P, \succ)$ is *minimally induced* by $(P, \trianglerighteq)$ if for any relation $(x, y) \in \trianglerighteq \setminus \succ$, the pair $(P, \succ \cup (x, y))$ is not a valid partial order, i.e. its corresponding graph contains a directed cycle.

We require the following lemma, bounding the width of a minimally induced poset.

Lemma 5.14. *Let $(P, \succ)$ be a poset minimally induced by the transitive relation $(P, \trianglerighteq)$. Then the width of $(P, \succ)$ is equal to the width of $(P, \trianglerighteq)$.*

Proof. Suppose otherwise, that is, suppose that there is a pair of distinct elements $x, y \in P$ such that $x \not\succ y$ with respect to the partial order $(P, \succ)$, but x and y have some relation in $(P, \trianglerighteq)$. Without loss of generality, suppose that $x \trianglerighteq y$; it may be simultaneously true that $y \trianglerighteq x$. First, we note that $(P, \succ \cup (x, y))$ is a valid partial order; if it were not, i.e. if the addition of (x, y) introduced a cycle, then it would be the case that $y \succ x$, which is a contradiction to their incomparability. However, the poset $(P, \succ \cup (x, y))$ is also induced by $(P, \trianglerighteq)$, which contradicts the assumption

that $(P, \succ)$ is minimally induced. $\square$

We denote by $\mathcal{O}_{\succ}$ an oracle for a poset $(P, \succ)$ and by $\mathcal{O}_{\trianglerighteq}$ an oracle for a transitive relation $(P, \trianglerighteq)$. In the following claim, we assume that the poset sorting algorithm outputs a chain decomposition (such as a ChainMerge); if it does not, the total complexity of the algorithm for sorting a transitive relation increases a bit, but not its query complexity.

Claim 5.15. *Suppose there is an algorithm $\mathcal{A}$ that, given a set P of n elements, access to an oracle $\mathcal{O}_{\succ}$ for a poset $\mathcal{P} = (P, \succ)$, and an upper bound of w on the width of $\mathcal{P}$, sorts P using $f(n, w)$ queries and $g(n, w)$ total complexity. Then there is an algorithm $\mathcal{B}$ that, given P, w, and access to an oracle $\mathcal{O}_{\trianglerighteq}$ for a transitive relation $(P, \trianglerighteq)$ of width at most w, sorts P using $f(n, w) + 2nw$ queries and $g(n, w) + O(nw)$ total complexity.*

Proof. Given an oracle $\mathcal{O}_{\trianglerighteq}$ for the transitive relation $(P, \trianglerighteq)$, we define a special poset oracle $\mathcal{O}$ that runs as follows: Given a query $q(x, y)$, the oracle $\mathcal{O}$ first checks if the relation between x and y can be inferred by transitivity and irreflexivity from previous responses. If so, it outputs the appropriate inferred response; otherwise, it forwards the query to the oracle $\mathcal{O}_{\trianglerighteq}$. The oracle $\mathcal{O}$ outputs the response of $\mathcal{O}_{\trianglerighteq}$ except if both $x \trianglerighteq y$ and $y \trianglerighteq x$; in this case, $\mathcal{O}$ outputs whichever relation is consistent with the partial order determined by previous responses (if both relations are consistent, then it arbitrarily outputs one of the two). By definition, the responses of $\mathcal{O}$ are consistent with a partial order induced by $(P, \trianglerighteq)$.

The first step of algorithm $\mathcal{B}$ is to run algorithm $\mathcal{A}$ on input P and w, giving $\mathcal{A}$ access to the special oracle $\mathcal{O}$, which $\mathcal{B}$ simulates using its access to $\mathcal{O}_{\trianglerighteq}$. Since $\mathcal{A}$ completely sorts its input, it reconstructs a poset induced by $(P, \trianglerighteq)$ via $\mathcal{O}$ that has a maximal set of relations. That is, there is a poset $\mathcal{P} = (P, \succ)$ minimally induced by $(P, \trianglerighteq)$ such that the responses of $\mathcal{O}$ to the sequence of queries made by $\mathcal{A}$ are

indistinguishable from the responses of $\mathcal{O}_{\succ}$ to the same sequence of queries. Since $\mathcal{P}$ has the same width as $(P, \trianglerighteq)$, it is valid to give $\mathcal{A}$ the upper bound of w. Hence, $\mathcal{A}$ sorts $\mathcal{P}$ and outputs some chain decomposition $\mathcal{C} = \{C_1, \ldots C_q\}$ of $\mathcal{P}$ such that $q \leq w$.

The second step of algorithm $\mathcal{B}$ is to make a sequence of queries to the oracle $\mathcal{O}_{\trianglerighteq}$ to recover the relations in $\trianglerighteq \setminus \succ$. It is similar to building a ChainMerge data structure: for all i, j, $1 \leq i, j \leq q$, for every element $x \in C_i$, we store the index of x in chain C_i and the index of the largest element $y \in C_j$ such that $x \trianglerighteq y$. An analysis similar to the one for ChainMerge (see Section 5.3) shows that it takes at most $2nq$ queries to the oracle $\mathcal{O}_{\trianglerighteq}$ and $O(nq)$ total complexity to find all the indices. The relation in $(P, \trianglerighteq)$ between any pair of elements can then be looked up in constant time. □

Chapter 6

Selection in posets

In this chapter, we consider the natural problem, closely related to sorting, of determining the minimal elements of a poset. We also study its generalization, the problem of determining the set of elements in the k bottom levels of the partial order.

We begin with formal definitions and brief discussion of results. In Section 6.2, we give deterministic and randomized algorithms for both problems. Section 6.3 covers lower bounds in adversarial and randomized models of computation.

6.1 The k-selection problem

Recall from Chapter 5, Section 5.1, that the *height* of an element a is the maximum cardinality of a chain whose elements are all dominated by a. The k-selection problem is the problem of finding the elements in the bottom k layers, i.e., the elements of height at most $k-1$, of a poset $\mathcal{P} = (P, \succ)$, given the set P of n elements, an upper bound w on the width of $\mathcal{P}$, and a query oracle for $\mathcal{P}$.

To our knowledge, k-selection in posets has not been previously considered. The upper bounds that we prove in Section 6.2 arise from natural generalizations of analogous algorithms for total orders. The lower bounds, however, are achieved quite

differently; a detailed description of the arguments is given in Section 6.3. We conjecture that our deterministic lower bound for the case of $k = 1$ is actually tight, though the upper bound is off by a factor of 2.

6.2 Upper bounds

In this section, we analyze some deterministic and randomized algorithms for the k-selection problem.

6.2.1 Finding the minimal elements

We begin with the 1-selection problem, i.e., the problem of finding the minimal elements.

Theorem 6.1. *The minimal elements can be found deterministically with at most wn queries and $O(wn)$ total complexity.*

Proof. The algorithm updates a set of size w of elements that are candidates for being smallest elements. Initialize $T_0 = \emptyset$. Assume that the elements are $x_1, \ldots, x_n$. At step t:

- Compare x_t to all elements in T_{t-1}.
- If there exists some $a \in T_{t-1}$ such that $x_t \succ a$, do nothing.
- Otherwise, remove from T_{t-1} all elements a such that $a \succ x_t$ and put x_t into T_t.

At the termination of the algorithm, the set T_n contains all elements of height 0. By construction of T_t, for all t, the elements in T_t are mutually incomparable. Therefore, for all t, it holds that $|T_t| \leq w$, and hence the query complexity of the algorithm is at most wn. □

Theorem 6.2. *There exists a randomized algorithm that finds the minimal elements in an expected number of queries that is at most*

$$\frac{w+1}{2}n + \frac{w^2-w}{2}(\log n - \log w).$$

Proof. The algorithm is similar to the algorithm for the proof of Theorem 6.1, with modifications to avoid (in expectation) worst-case behavior. Let σ be a permutation of $[n]$ chosen uniformly at random. Let $T_1 = \{x_{\sigma(1)}\}$. For $1 \leq t < n$, at step t:

- Let i be an index of the candidates in T_{t-1}, i.e. $T_{t-1} = \{x_{i(1)}, \ldots, x_{i(r)}\}$, where $r \leq w$.
- Let $T_t = T_{t-1}$. Let τ be a permutation of $[r]$ chosen uniformly at random.
- For $j = 1, \ldots, r$:
  - If $x_{\sigma(t)} \succ x_{i(\tau(j))}$, exit the loop and move to step $t+1$.
  - If $x_{i(\tau(j))} \succ x_{\sigma(t)}$, remove $x_{i(\tau(j))}$ from T_t.
- Add $x_{\sigma(t)}$ to T_t.

As in the previous algorithm, it is easy to see that at each step t, the set T_t contains all the minimal elements of $A_t = \{x_{\sigma(1)}, \ldots, x_{\sigma(t)}\}$ and that $|T_t| \leq w$. Note furthermore that at step t,

$$\mathbf{P}[x_{\sigma(t)} \text{ is minimal for } A_t] \leq \frac{w}{t}.$$

If $x_{\sigma(t)}$ is not minimal for A_t, then the expected number of comparisons needed until $x_{\sigma(t)}$ is compared to an element $a \in A_t$ that dominates $x_{\sigma(t)}$ is clearly at most $(w+1)/2$.

We thus conclude that the expected running time of the algorithm is bounded by:

$$\begin{aligned}
&\sum_{t=2}^{w}(t-1) + \sum_{t=w+1}^{n}\left(\frac{w}{t}w + \frac{(t-w)}{t}\frac{(w+1)}{2}\right) \\
&= \binom{w}{2} + \sum_{t=w+1}^{n}\frac{1}{2t}\left(w^2 - w + tw + t\right) \\
&\leq \frac{w+1}{2}n + \frac{w^2-w}{2}(\log n - \log w).
\end{aligned}$$

□

6.2.2 General k-selection

We now turn to the k-selection problem for $k > 1$. We first provide deterministic upper bounds on query and total complexity.

Theorem 6.3. *The query complexity of the k-selection problem is at most*

$$16wn + 4n\log(2k) + 6n\log w.$$

Moreover, there exists an efficient k-selection algorithm with query complexity at most

$$8wn\log(2k)$$

and total complexity

$$O(w^2 n\log(2k)).$$

Proof. The basic idea is to use the sorting algorithm presented in previous sections in order to update a set of candidates for the k-selection problem. Denote the elements by $x_1, \ldots, x_n$. Let $C_0 = \emptyset$. The algorithm proceeds as follows, beginning with t=1:

- While $(t-1)wk + 1 \leq n$, let $D_t = C_{t-1} \cup \{x_{(t-1)wk+1}, \ldots, x_{\min(twk,n)}\}$.

- Sort D_t. Let C_t be the solution to the k-selection problem for D_t.

Clearly, at the end of the execution, C_t contains the solution to the k-selection problem. As we have shown, the query complexity of sorting D_t is

$$4wk \log (2wk) + 16w^2k + 2w \log w$$

and, therefore, the query complexity of the algorithm is

$$\begin{aligned} &\frac{n}{wk}(4wk \log (2wk) + 16w^2k + 2w \log w) \\ &= 4n \log (2wk) + 16wn + \frac{2n}{k} \log w. \end{aligned}$$

This proves the first result. Using the computationally efficient sorting algorithm, we get sorting query complexity $8w^2k \log (2k)$, which results in total query complexity $8nw \log (2k)$ and total complexity $O(nw^2 \log(2k))$. □

Next we outline a randomized algorithm with a better coefficient of the main term wn.

Theorem 6.4. *The k-selection problem has a randomized query complexity of at most*

$$wn + 16kw^2 \log n \log(2k)$$

and total complexity

$$O(wn + \text{poly}(k, w) \log n).$$

Proof. We use the following algorithm:

- Choose an ordering $x_1, \ldots, x_n$ of the elements uniformly at random.
- Let $C_{wk} = \{x_1, \ldots, x_{wk}\}$ and $D_{wk} = \emptyset$.
- Sort C_{wk}. Remove any elements from C_{wk} that are of height greater than $k-1$.

- Let $t = wk + 1$. While $t \leq n$ do:
  - Let $C_t = C_{t-1}$ and $D_t = D_{t-1}$.
  - Compare x_t to the maximal elements in C_t in a random order.
    - For each maximal element $a \in C_t$: if $\text{height}(a) = k - 1$ and $a \succ x_t$, or if $\text{height}(a) < k - 1$ and $x_t \succ a$, then add x_t to D_t, and exit this loop.
    - If for all elements $a \in C_t$, $x_t \not\succ a$, then add x_t to D_t and exit this loop;
  - If $|D_t| = wk$ or $t = n$:
    - Sort $C_t \cup D_t$.
    - Set C_t to be the elements of height at most $k - 1$ in $C_t \cup D_t$.
    - Set $D_t = \emptyset$.
- Output the elements of C_n.

It is clear that C_n contains the solution to the k-selection problem. To analyze the query complexity of the algorithm, recall from Theorem 5.7 that

$$s(w, k) = 8w^2 k \log(2k)$$

is an upper bound on the number of queries used by the efficient sorting algorithm to sort $2wk$ elements in a width-w poset.

There are two types of contributions to the number of queries made by the algorithm: (1) comparing elements to the maximal elements of C_t, and (2) sorting the sets C_0 and $C_t \cup D_t$.

To bound the expected number of queries of the first type, we note that for $t \geq kw + 1$, since the elements are in a random order, the probability that x_t ends up in D_t is at most $\min\left(1, \frac{2kw}{t}\right)$. If x_t is not going to be in D_t, then the number

of queries needed to verify this is bounded by w. Overall, the expected number of queries needed for comparisons to maximal elements is bounded by wn.

To calculate the expected number of queries of the second type, we bound the expected number of elements that need to be sorted as follows:

$$\sum_{t=kw+1}^{n} \min\left(1, \frac{2kw}{t}\right) \leq 2kw(\log n - 1).$$

We thus obtain an upper bound on the total query complexity of

$$wn + 2s(w,k)\log n.$$

□

6.3 Lower bounds

We obtain lower bounds for the k-selection problem both for adaptive and non-adaptive adversaries. Some of our proofs use the following lower bound on finding the k-th smallest element of a linear order on n elements:

Theorem 6.5 (Fussenegger-Gabow [17])**.** *The number of queries required to find the kth smallest element of an n-element total order is at least $n - k + \log\binom{n}{k-1}$.*

The proof of Theorem 6.5 shows that every comparison tree that identifies the kth smallest element must have at least $2^{n-k}\binom{n}{k-1}$ leaves, which implies that the theorem also holds for randomized algorithms.

6.3.1 Adversarial lower bounds

We consider adversarial lower bounds for the k-selection problem. In this model, an adversary simulates the oracle and is allowed to choose her response to a query after

she receives it. Any response is legal as long as there is some partial order of width w with which all of her responses are consistent. We begin with the case of $k = 1$, i.e. finding the set of minimal elements.

Theorem 6.6. *In the adversarial model, at least* $\frac{w+1}{2}n - w$ *queries are needed in order to find the minimal elements.*

Proof. Consider the following adversarial algorithm. The algorithm outputs query responses that correspond to a poset $\mathcal{P}$ of w disjoint chains. Given a query $q(a, b)$, the algorithm outputs a response to the query, and in some cases, it may also announce for one or both of a and b to which chain the element belongs. Note that receiving this extra information can only make things easier for the query algorithm. During the course of the algorithm, the adversary maintains a graph $G = (P, E)$. Whenever the adversary responds that $a \not\sim b$, it adds an edge (a, b) to E.

Let $q_t(a)$ be the number of queries that involve element a, out of the first t queries overall. Let $c(a)$ be the chain assignment that the adversary has announced for element a. (We set $c(a)$ to be undefined for all a, initially.) Let $\{x_i\}_{i=1}^n$ be an indexing, chosen by the adversary, of the elements of P. Let $q(a, b)$ be the t'th query. The adversary follows the following protocol:

- If $q_t(a) \leq w - 1$ or $q_t(b) \leq w - 1$, return $a \not\sim b$. In addition:
  - If $q_t(a) = w - 1$, choose a chain $c(a)$ for a that is different from all the chains to which a's neighbors in G belong, and output it.
  - If $q_t(b) = w-1$ choose a chain $c(b)$ for b that is different from all the chains to which b's neighbors in G belong, and output it.
- If $q_t(a) > w - 1$, $q_t(b) > w - 1$, and $c(a) \neq c(b)$, then output $a \not\sim b$.
- Otherwise, let i and j be the indices of a and b, respectively (i.e. $a = x_i$ and $b = x_j$). If $i > j$, then output $a \succ b$; otherwise, output $b \succ a$.

It is easy to see that the output of the algorithm is consistent with a width-w poset consisting of w chains that are pairwise incomparable. We will also require that each of the chains is chosen at least once (this is easily achieved).

We now prove a lower bound on the number of queries to this algorithm required to find a proof that the minimal elements are indeed the minimal elements.

In any proof that a is *not* a smallest element, it must be shown to dominate at least one other element, but to get such a response from the adversary, a must be queried against at least $w-1$ other elements with which it is incomparable. To prove that a minimal element of one chain is indeed minimal, it must be queried at least against the minimal elements of the other chains to rule out the possibility it dominates one of them. Therefore, each element must be compared to at least $w-1$ elements that are incomparable to it. So the total number of queries of type $q(a,b)$, where $a \not\sim b$, is at least $\frac{w-1}{2}n$.

In addition, for each chain c_i of length n_i, the output must provide a proof of minimality for the minimal element of that chain. By Theorem 6.5, this contributes $n_i - 1$ queries for each chain c_i.

Summing over all the bounds proves the claim. □

Theorem 6.7. *Let $r = \frac{n}{2w-1}$. If $k \leq r$ then the number of queries required to solve the k-selection problem is at least*

$$\frac{(w+1)n}{2} - w(k+\log k) - \frac{w^3}{8} + \min\left((w-1)\log\binom{r}{k-1} + \log\binom{rw}{k-1}, \frac{n(r-k)(w-1)}{2r} + \log\binom{n-(w-1)k}{k-1}\right).$$

Proof. The adversarial algorithm outputs query responses exactly as in the proof of Theorem 6.6, except in the case where the tth query is (a,b) and $q_t(a) = w-1$ or $q_t(b) = w-1$. In that case it uses a more specific rule for the assignment of one or both of these elements to chains.

In addition to assigning the elements to chains, the process must also select the k smallest elements in each chain, and Theorem 6.5 gives a lower bound, in terms of the lengths of the chains, on the number of queries required to do so.

We think of the assignment of elements to chains as a coloring of the elements with w colors. The specific color assignment rule is designed to ensure that, if the number of elements of color c is small, then there must have been many queries in which the element being colored could not receive color c because it had already been declared incomparable to an element with color c. It will then follow that there have been a large number of queries in which an element was declared incomparable to an element with color c. Thus, if many of the chains are very short, then the number of pairs declared incomparable must be very large. On the other hand, if few of the chains are very short, then we can employ Theorem 6.5 to show that the number of queries required to select the k smallest elements in each chain must be large. We obtain the overall lower bound by playing off these two observations against each other.

The color assignment rule is based on a function $d_t(c)$, referred to as the *deviation* of color c after query t, and satisfying the initial condition $d_0(c) = 0$ for all c. The rule is: "assign the eligible color with smallest deviation."

More specifically, let the tth query be (a_t, b_t). The adversary processes a_t and then b_t. Recall that $q_t(a)$ is the number of queries involving element a out of the first t queries overall. Element $e \in \{a_t, b_t\}$ is processed exactly as in the proof of Theorem 6.6 except when $q_t(e) = w - 1$. In that case, let $S_t(e)$ be the set of colors that are *not* currently assigned to neighbors of e; i.e., the set of colors eligible to be assigned to element e. Let $c* = \text{argmin}_{c \in S_t(e)} d_{t-1}(c)$. The adversary assigns color $c*$ to e. Then the deviations of all colors are updated as follows:

1. if $c \notin S_t(e)$ then $d_t(c) = d_{t-1}(c)$;

2. $d_t(c^*) \leftarrow d_{t-1}(c^*) + 1 - \frac{1}{|S_t(e)|}$;

3. For $c \in S_t(e) \setminus \{c^*\}$, $d_t(c) \leftarrow d_{t-1}(c) - \frac{1}{|S_t(e)|}$.

The function $d_t(c)$ has the following interpretation: over the history of the color assignment process, certain steps occur where the adversary has the choice of whether to assign color c to some element; $d_t(c)$ represents the number of times that color c was chosen up to step t, minus the expected number of times it would have been chosen if the same choices had been available at all steps and the color had been chosen uniformly at random from the set of eligible colors.

Because the smallest of the deviations of eligible colors is augmented at each step, it is not possible for any deviation to drift far from zero. Specifically, it can be shown by induction on t that at every step t the sum of the deviations is zero and for $m = 1, 2, \ldots, w$, the sum of the m smallest deviations is greater than or equal to

$$\frac{m(m-w)}{2}.$$

Let $\deg_G(a)$ be the degree of a in G at the end of the process. At the end of the process every element of degree greater than or equal to $w-1$ in G has been assigned to a chain. Each element of degree less than $w-1$ has not been assigned to a chain, and is therefore called *unassigned*. An unassigned element is called *eligible* for chain c if it has not been compared (and found incomparable) with any element of chain c. Let $s(c)$ be the length of chain c and define $\mathrm{def}(c)$, the *deficiency* of chain c, as $\max(0, k - s(c))$. Define the *total deficiency* DEF as the sum of the deficiencies of all chains.

Let u be the number of unassigned elements. Upon the termination of the process it must be possible to infer from the results of the queries that every unassigned element is of height at most $k-1$. This implies that, if unassigned element x is

eligible for chain c, then the number of unassigned elements eligible for chain c must be at most $\mathrm{def}(c)$. Thus the number of pairs (a, c) such that unassigned element a is eligible for chain c is DEF. Define the *deficiency* of unassigned element a as $w - 1 - \deg_G(a)$. Then the sum of the deficiencies of the unassigned elements is bounded above by DEF, and therefore the sum of the degrees in G of the unassigned elements is at least $(w-1)u - \mathrm{DEF}$.

By Theorem 6.5, if $s(c) > k$, then at least

$$\left(s(c) - k + \log \binom{s(c)}{k-1}\right)$$

queries are needed to determine the k smallest elements of chain c.

The total number of queries is the number of edges that have been placed in G in the course of the algorithm (i.e., the number of pairs that have been declared incomparable by the adversary), plus the number of queries required to perform k-selection in each chain. The total number of pairs that have been declared incomparable is $\frac{1}{2}\sum_a \deg_G(a)$.

Let $d(c)$ be the deviation of color c at the end of the process. Let $r(c)$ be the number of steps in the course of the process at which the element being colored was eligible to receive color c. If, at each such step, the color had been chosen uniformly from the set of eligible colors, then the chance of choosing color c would have been at least $\frac{1}{w}$. Thus, by the interpretation of the function $d_t(c)$ given above, $s(c) \geq \frac{r(c)}{w} + d(c)$; equivalently, $r(c) \leq w(s(c) - d(c))$. Also,

$$\sum_{a|c(a)=c} \deg_G(a) \geq n - r(c) \geq n - w(s(c) - d(c)).$$

This sum is also at least $(w-1)s(c)$, since every element assigned to c has been declared incomparable with at least $(w-1)$ other elements.

We can now combine these observations to obtain our lower bound. For each chain c define

$$\text{cost}(c) = \frac{1}{2} \sum_{a|c(a)=c} \deg_G(a) + \max\left(0, s(c) - k + \log\binom{s(c)}{k-1}\right).$$

Then $\sum_c \text{cost}(c) + \frac{1}{2}\left((w-1)u - \text{DEF}\right)$ is a lower bound on the total number of queries, and

$$\begin{aligned}\sum_c \text{cost}(c) \geq\ & \frac{1}{2}\sum_c \max\left((w-1)s(c), n - w(s(c) - d(c))\right) \\ & + \sum_{c|s(c)>k} \left(s(c) - k + \log\binom{s(c)}{k-1}\right).\end{aligned}$$

To obtain our lower bound we minimize this function over all choices of nonnegative integers $s(c)$, u and DEF such that

$$\sum_c s(c) + u = n \qquad \text{and} \qquad \text{DEF} = \sum_c \max(0, k - s(c)).$$

Noting that

$$\sum_c \min(d(c), 0) \geq \min_m \frac{m(m-w)}{2} = -\frac{w^2}{8},$$

we obtain the following lower bound on the total number of queries:

$$\begin{aligned}\frac{(w-1)n}{2} - \frac{\text{DEF}}{2} - \frac{w^3}{8} + \frac{1}{2}\sum_c \max\left(0, n - (2w-1)s(c)\right) \\ + \sum_{c|s(c)>k} \left(s(c) - k + \log\binom{s(c)}{k-1}\right)\end{aligned} \tag{6.1}$$

We now restrict attention to the case $k \leq \frac{n}{2w-1}$. Let $r = \frac{n}{2w-1}$. We show that, at any global minimum of (6.1), DEF $= 0$. To see this, consider any choice of $\{s(c)\}$ such that DEF > 0. Let c be a chain such that $\text{def}(c) > 0$. If $s(c)$ is increased by 1,

then DEF decreases by 1, and the net change in the value of quantity (6.1) is $1 - w$, which is negative.

Thus, in minimizing (6.1) we may assume that DEF $= 0$, and hence that $\sum_c s(c) = n$. So (6.1) may be rewritten as

$$\frac{(w-1)n}{2} - \frac{w^3}{8} + \sum_c F(s(c))$$

where

$$F(s) = \begin{cases} \frac{1}{2}\max(0, n-(2w-1)s) & \text{if } s \le k \\ \frac{1}{2}\max(0, n-(2w-1)s) + \left(s - k + \log\binom{s}{k-1}\right) & \text{if } s > k. \end{cases}$$

Thus, we have the following minimization problem:

$$\text{Minimize } \sum_c F(s(c)), \text{ subject to } s(c) \ge 0 \text{ and } \sum_c s(c) = n.$$

First, we note that

$$\sum_{c|k<s(c)} (s(c) - k) = n - wk.$$

To determine the minimum we consider three ranges of values: the low range $s = k$, medium range $k < s(c) \le r$, and high range $r < s(c) \le n$. Observing that $F(s)$ is strictly concave in the medium range, and concave and strictly increasing in the high range, it follows that, at the global minimum of (6.1), $s(c)$ is equal to either k or r except for one value in the high range and possibly one value strictly within the medium range. The value in the high range is at least rw, since the sum of the values in the low and medium ranges does not exceed $r(w-1)$. If

$$\sum_{c|k\le s(c)\le r} s(c) = (w-1)r - D,$$

then the unique value of $s(c)$ in the high range is $rw + D$. Moreover, exploiting the concavity of $F(s)$ in the medium range, we claim that

$$\sum_{c|k\le s(c)\le r} \log \binom{s(c)}{k-1} \ge \left(w - 1 - \frac{D}{r-k}\right) \log \binom{r}{k-1}.$$

This bound is at most $w \log k$ greater than the sum

$$\sum_{c|k< s(c)\le r} \log \binom{s(c)}{k-1}.$$

Finally, a simple calculation shows that

$$\frac{1}{2}\sum_{c} \max(0, n - (2w-1)s(c)) = \frac{nD}{2r}.$$

Thus we get the following lower bound on $\sum_c F(s(c))$:

$$n - w(k + \log k) + \min_{0\le D\le (w-1)(r-k)} \left(\left(w - 1 - \frac{D}{r-k}\right) \log \binom{r}{k-1} + \frac{nD}{2r} + \log \binom{rw+D}{k-1}\right).$$

As a concave function, this is minimized either at $D = 0$ or $D = (w-1)(r-k)$.This yields the following lower bound on the worst-case number of queries required to solve the k-selection problem when $k \le r$:

$$\frac{(w+1)n}{2} - w(k + \log k) - \frac{w^3}{8} + \min\left((w-1)\log\binom{r}{k-1} + \log\binom{rw}{k-1}, \frac{n(r-k)(w-1)}{2r} + \log\binom{n-(w-1)k}{k-1}\right).$$

□

6.3.2 Lower bounds in the randomized query model

We now prove lower bounds on the number of queries used by randomized k-selection algorithms. We conjecture that the randomized algorithm for finding the minimal elements which we give in the proof of Theorem 6.2 essentially achieves the lower bound. However, the lower bound we prove here is a factor 2 different from this upper bound.

We consider a distribution $D(n, w)$ on partial orders of width w over a set $P = \{x_1, \ldots, x_n\}$. The distribution $D(n, w)$ is defined as follows:

- The support of $D(n, w)$ is the set of partial orders consisting of w chains, where any two elements from different chains are incomparable.
- Each element belongs independently to one of the w chains with equal probability.
- The linear order on each chain is chosen uniformly.

Theorem 6.8. *The expected query complexity of any algorithm solving the k-selection problem is at least*

$$\frac{w+3}{4}n - wk + w\left(1 - \exp\left(-\frac{n}{8w}\right)\right)\left(\log\binom{n/(2w)}{k-1}\right).$$

Proof. In order to provide a lower bound on the number of queries, we provide a lower bound on the number of queries of incomparable elements and then use the classical bound to bound the number of queries of comparable elements.

First we note that for each element a, the algorithm must make either at least one query where a is comparable to some other element b, or at least $w-1$ queries where a is incomparable to all elements queried. (The latter may suffice in cases where a is the unique element of a chain and it is compared to all minimal elements of all other chains.)

We let $Y_t(i)$ denote the number of queries involving x_i *before* the first query for which the response is that x_i is comparable to an element. Also for each of the chains $C_1, \ldots, C_w$ we denote by Z_α the number of queries involving two elements from the same chain.

Letting T denote the total number of queries before the algorithm terminates, we obtain:

$$\mathbf{E}[T] \geq \sum_{i=1}^{n} \frac{1}{2} \mathbf{E}(Y_T(i)) + \sum_{\alpha=1}^{w} \mathbf{E}[Z_\alpha].$$

We claim that for all $1 \leq i \leq n$ we have $\mathbf{E}[Y_T(i)] \geq \frac{w-1}{2}$. This follows by conditioning on the chains that all other elements but x_i belong to. With probability $1/w$, the first query will give a comparison; with probability $1/w$, the second query, etc.

On the other hand, by the classical lower bounds we have for each $1 \leq \alpha \leq w$ that

$$Z_\alpha \geq |C_\alpha| - k + \log \binom{|C_\alpha|}{k-1}$$

Taking expected value we obtain

$$\mathbf{E}[Z_\alpha] \geq \frac{n}{w} - k + \mathbf{E}\left[\log \binom{|C_\alpha|}{k-1}\right].$$

A rough bound on the previous expression may be obtained by using the fact that by standard Chernoff bounds, except with probability $\exp(-\frac{n}{8w})$, it holds that C_α is of size at least $\frac{n}{2w}$. Therefore

$$\mathbf{E}\left[\log \binom{|C_\alpha|}{k-1}\right] \geq \left(1 - \exp\left(-\frac{n}{8w}\right)\right) \log \binom{\frac{n}{2w}}{k-1}.$$

Summing all of the expressions above, we obtain

$$\frac{(w-1)n}{4} + w\left(\frac{n}{w} - k\right) + \left(1 - \exp\left(-\frac{n}{8w}\right)\right) w \log \binom{\frac{n}{2w}}{k-1}$$

and simplifying gives the desired result. □

Bibliography

[1] G. Brightwell. Balanced pairs in partial orders. *Discrete Mathematics*, 201(1–3):25–52, 1999.

[2] G. Brightwell and S. Goodall. The number of partial orders of fixed width. *Order*, 20(4):333–345, 2003.

[3] G. Brightwell and P. Winkler. Counting linear extensions is #p-complete. In *Proc. of ACM Symposium on Theory of Computing (STOC)*, 1991.

[4] T. M. Chan. Euclidean bounded-degree spanning tree ratios. In *Proc. of Symposium on Computational Geometry*, pages 11–19. ACM Press, 2003.

[5] K. Chaudhuri, S. Rao, S. Riesenfeld, and K. Talwar. What would edmonds do? augmenting paths and witnesses for bounded degree msts. In *Proc. of APPROX/RANDOM*, 2005.

[6] K. Chaudhuri, S. Rao, S. Riesenfeld, and K. Talwar. A push-relabel algorithm for approximating degree-bounded spanning trees. In *Proc. of International Colloquium on Automata, Languages and Programming (ICALP)*, 2006.

[7] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons Inc, New York, 1991.

[8] C. Daskalakis, R. M. Karp, E. Mossel, S. Riesenfeld, and E. Verbin. Sorting and selection in posets. *arXiv:0707.1532v1 [cs.DS]*, 2007.

[9] J. Edmonds. Maximum matching and a polyhedron with 0–1 vertices. *Journal of Research National Bureau of Standards*, 69B:125–130, 1965.

[10] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[11] M. Ellingham and X. Zha. Toughness, trees and walks. *J. Graph Theory*, 33(3):125–137, 2000.

[12] S. Even and R. Endre Tarjan. Network flow and testing graph connectivity. *SIAM Journal on Computing*, 4(4):507–518, December 1975.

[13] U. Faigle and Gy. Turán. Sorting and recognition problems for ordered sets. *SIAM J. Comput*, 17(1):100–113, 1988.

[14] T. Fischer. Optimizing the degree of minimum weight spanning trees. Technical Report 14853, Dept of Computer Science, Cornell University, Ithaca, NY, 1993.

[15] M. Fredman. How good is the information theory bound in sorting? *Theor. Comput. Sci.*, 1(4):355–361, 1976.

[16] 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.

[17] F. Fussenegger and H. N. Gabow. A counting approach to lower bounds for selection problems. *Journal of the ACM*, 26(2):227–238, 1979.

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

[19] M. Goemans. Minimum bounded-degree spanning trees. In *Proc. of IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 273–282, 2006.

[20] A. V. Golberg. A new max-flow algorithm. Technical Report MIT/LCS/TM-291, Massachussets Institute of Technology, 1985. Technical Report.

[21] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. In *Proc. of ACM Symposium on Theory of Computing (STOC)*, pages 136–146. ACM Press, 1986.

[22] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988.

[23] J. A. Hoogeveen. Analysis of christofides' heuristic: Some paths are more difficult than cycles. *Operation Research Letters*, 10:291– 295, 1991.

[24] J. E. Hopcroft and R. M. Karp. An $n^{\frac{5}{2}}$ algorithm for maximum matching in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[25] Jr. J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proc. of the American Mathematical Society*, volume 7, pages 48–50, 1956.

[26] K. Jain. Factor 2 approximation algorithm for the generalized steiner network problem. In *Proc. of IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 448–457.

[27] R. Jothi and B. Raghavachari. Degree-bounded minimum spanning trees. In *Proc. Canadian Conf. on Computational Geometry (CCCG)*, 2004.

[28] J. Kahn and J. H. Kim. Entropy and sorting. In *Proc. of ACM Symposium on Theory of Computing (STOC)*, pages 178–187, 1992.

[29] J. Kahn and M. Saks. Balancing poset extensions. *Order*, 1(2):113–126, 1984.

[30] S. Khuller, B. Raghavachari, and N. Young. Low-degree spanning trees of small weight. *SIAM J. Comput.*, 25(2):355–368, 1996.

[31] S. S. Kislitsyn. A finite partially ordered set and its corresponding set of permutations. *Matematicheskie Zametki*, 4(5):511–518, 1968.

[32] D. Knuth. *The Art of Computer Programming: Sorting and Searching*. Addison-Wesley, Massachusetts, 1998.

[33] J. Könemann and R. Ravi. A matter of degree: improved approximation algorithms for degree-bounded minimum spanning trees. In ACM, editor, *Proc. of ACM Symposium on Theory of Computing (STOC)*, pages 537–546, New York, NY, USA, 2000. ACM Press.

[34] J. Könemann and R. Ravi. A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. *SIAM Journal on Computing*, 31(6):1783–1793, December 2002.

[35] J. Könemann and R. Ravi. Primal-dual meets local search: approximating MST's with nonuniform degree bounds. In *Proc. of ACM Symposium on Theory of Computing (STOC)*, pages 389–395, New York, NY, USA, 2003. ACM Press.

[36] R. Krishnan and B. Raghavachari. The directed minimum-degree spanning tree problem. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 232–243, 2001.

[37] Ford L. R., Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[38] L. C. Lau, J. (Seffi) Naor, M. R. Salavatipour, and M. Singh. Survivable network design with degree or order constraints. In *Proc. of ACM Symposium on Theory of Computing (STOC)*, pages 651–660, New York, NY, USA, 2007. ACM.

[39] N. Linial. The information theoretic bound is good for merging. *SIAM J. Comput. SICOMP*, 13(4):795–801, 1984.

[40] Crispin Nash-Williams. Decomposition of finite graphs into forests. *J. London Math. Soc.*, 39, 1964.

[41] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[42] K. Onak and P. Parys. Generalization of binary search: Searching in trees and forest-like partial orders. In *Proc. of IEEE Symposium on Foundations of Computer Science (FOCS)*, 2006.

[43] C. H. Papadimitriou and U. Vazirani. On two geometric problems related to the traveling salesman problem. *J. Algorithms*, 5:231–246, 1984.

[44] R. C. Prim. Shortest connection networks and some generalisations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[45] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt, III. Many birds with one stone: multi-objective approximation algorithms. In *Proc. of ACM Symposium on Theory of Computing (STOC)*, pages 438–447. ACM Press, 1993.

[46] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt, III. Approximation algorithms for degree-constrained minimum-cost network-design problems. *Algorithmica*, 31, 2001.

[47] R. Ravi and Mohit Singh. Delegate and conquer: An LP-based approximation algorithm for minimum degree msts. In *Proc. of International Colloquium on Automata, Languages and Programming (ICALP)*, 2006.

[48] M. Singh and L. C. Lau. Approximating minimum bounded degree spanning trees to within one of optimal. In *Proc. of ACM Symposium on Theory of computing (STOC)*.

[49] A. Srivastav and S. Werth. Probabilistic analysis of the degree bounded minimum spanning tree problem. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2007.

[50] R. L. Rivest T. H. Cormen, C. E. Leiserson and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001.

[51] W. Trotter and S. Felsner. Balancing pairs in partially ordered sets. *Combinatorics, Paul Erdos is Eighty*, I:145–157, 1993.