Steiner Trees and Beyond: Approximation Algorithms for Network Design

Ramamurthy Ravi

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-93-41
September 1993

Steiner Trees and Beyond:
Approximation Algorithms for Network Design

by
Ramamurthy Ravi

B. Tech., Computer Science and Engineering
Indian Institute of Technology, Madras, India, May 1989

Sc. M., Computer Science
Brown University, May 1991

Thesis
Submitted in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy in the Department of Computer Science
at Brown University.

May 1994

This dissertation by Ramamurthy Ravi
is accepted in its present form by the Department of
Computer Science as satisfying the
dissertation requirement for the degree of Doctor of Philosophy.

Date ______________ ____

Philip N. Klein

Recommended to the Graduate Council

Date ______________ ____

Paris C. Kanellakis

Date ______________ ____

Franco P. Preparata

Approved by the Graduate Council

Date ______________ ____

Vita

I was born on January 8, 1969 to R. S. Ramamurthy and R. Savithri in Salem, India. I did my schooling at Holy Angels Matriculation School and then at Holy Cross Matriculation School, both in Salem. I spent the next four years from 1985 to 1989 at the Indian Institute of Technology, Madras, at the end of which I was awarded a Bachelor of Technology degree in Computer Science and Engineering. I joined Brown in the Fall of 1989, and received my Master of Science degree in Computer Science from Brown in May 1991.

Abstract

We present approximation algorithms for several NP-hard optimization problems arising in network design. Almost all of our algorithms run in polynomial time and output solutions with a worst-case performance guarantee on the quality of the output solution.

A typical problem that we consider can be stated as follows: given an undirected graph and certain connectivity requirements, find a subgraph that satisfies these requirements and has minimum cost. In this thesis, we address many different connectivity requirements such as spanning trees, Steiner trees, generalized Steiner forests, and two-connected networks. The cost criteria that we consider range from the total cost of the edges in the network, the total cost of the nodes in the network, the maximum degree of any node in the network, the maximum cost of any edge in the network to some combination of these. We also address the maximum-leaf spanning tree problem and provide the first approximation algorithms for this problem.

In the course of deriving our results, we use three distinct techniques: the primal-dual technique that draws on the theory of linear programming, local optimization, and a solution-based decomposition method. Our work highlights the applicability of these techniques to designing approximation algorithms.

Acknowledgements

Thanks to Philip Klein for his invaluable guidance, encouragement and friendship. Thanks to Profs. Paris Kanellakis and Franco Preparata for being on my thesis committee and for their encouragement.

A very special thanks to all my colleagues for their interest, time and effort in collaborating with me: Alok Aggarwal, Ajit Agrawal, J. Cheriyan, M. Goemans, S. Khuller, P. N. Klein, Hsueh-I Lu, M. V. Marathe, B. Raghavachari, V. S. Ramakrishnan, S. Rao, S. S. Ravi, D. J. Rosenkrantz, S. Sairam, Ravi Sundaram, and D. P. Williamson.

Thanks to all the folks at Brown who made it the nicest working place I have known: all the faculty including John Savage, Roberto Tamassia, and Jeff Vitter; the administrative and technical staff including Tony, Mary, Dawn, Tina, Dorinda, Max, Trina, Pheeb, and Jennet, and most of all, my contemporary graduate students who are too many to list here.

I acknowledge with gratitude the great set of educators that contributed to my development from school all the way to college, from Sister Mary Catherine, Deepak, Selvaraju, Bro. Chacko, and Bro. Aurele Tessier, to C. R. Muthukrishnan, C. P. Rangan, and K. R. Parthasarathy.

Special thanks to my close set of friends: Ajit, Uma, Sai, Bharathi, and Mala, who have continually filled my life at Brown with their love and concern. Thanks to Pski and Anu for my times at Boston.

A very sincere thanks to Music for providing an essential source of inspiration and relaxation.

My family has been a great source of inspiration and support over the years. Times spent with Mani, Samy, Amma, Appa, and Gaitri, have been rejuvenating in their own special way. My parents are in most part responsible for who I am today. I dedicate this thesis to them.

Credits

The results reported in Chapters 3 and 4 on two-connected networks and node-weighted network-design problems were obtained jointly with Philip Klein [90, 91].

The results in Chapter 5 on designing minimum-degree networks were obtained jointly with Balaji Raghavachari and Philip Klein [131].

The results in Chapter 6 on multi-objective approximation algorithms were obtained jointly with Madhav V. Marathe, S. S. Ravi, Daniel J. Rosenkrantz, and Harry B. Hunt III [129].

The results in Chapter 7 on maximum-leaf spanning trees were obtained jointly with Hsueh-I Lu [111].

All the joint work mentioned above represents many hours of stimulating discussions with my colleagues. I am extremely grateful to all of them for their time and for allowing me to include our joint work in this thesis.

In addition, I wish to thank the following people for fruitful discussions on the research described in this thesis: A. Agrawal, J. Cheriyan, M. X. Goemans, S. Khuller, V. S. Ramakrishnan, S. Rao, S. Sairam, and D. P. Williamson.

Thanks to Vijay Vazirani for bringing to our attention an error in the tight example we provided in an earlier version [130] of our two-edge-connected network-design algorithm in Chapter 3, and for suggesting a correction. Thanks to Esther Jesurum for suggesting the term "spider" that we adopted in Chapter 4. Thanks to Bobby Blumofe for pointing out an error in an early draft of this result. Thanks to Laurence Wolsey for suggesting an alternative proof of Theorem 4.3.1. Thanks to Michel Goemans for suggesting a simple proof of Theorem 6.1.10 in Chapter 6.

I was supported by an IBM Graduate Fellowship for the academic years 1991-1992 and 1992-1993. Additional support for my research came from Prof. Philip Klein's NSF PYI award CCR-9157620 and DARPA contract N00014-91-J-4052 ARPA Order No. 8225. I wish to record here my gratitude to these funding agencies. Special thanks to Alok Aggarwal at the IBM T. J. Watson Research Center for making possible a week-long visit to IBM in the summer of 1991. I also wish to thank the TOC group at MIT for providing me with office space and other facilities when I spent the year of 1992-1993 there.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Steiner trees ...

The advent of high-speed and high-bandwidth networks has triggered a spate of work in the application of combinatorial techniques from discrete mathematics and computer science to problems in network design. The *Steiner tree problem* [68] is a classical example of such a problem that arises frequently in practice. This problem arises in applications as diverse as VLSI routing, telecommunication-network design and phylogeny. The problem can be stated in its simplest form as follows: given an undirected graph with a specified subset of nodes called the terminals, find a tree with the minimum number of edges spanning all the terminals. The graph may correspond to the pins of components in a given placement of a VLSI chip. In this case, the problem considered is that of finding a minimum-size net to route a connection. The graph may correspond to a set of locations that are required to be interconnected via a communication network in a telecommunication setting. The nodes of the graph may even correspond to different species and the links may represent biological similarities; the problem in this case is to identify a phylogenetic tree providing a good hypothesis for the evolutionary relationships among the selected set of species.

1.2 ... and beyond

The Steiner tree problem introduced above illustrates the two salient features of any network-design problem: the connectivity specification and the optimization objective. In the Steiner tree problem above, the connectivity specification is that the solution must connect all the terminals. The objective of this optimization problem is to find a solution of minimum cost where the cost is the number of edges in the solution. We can derive many different network-design problems by combining different connectivity specifications with different optimization objectives. In this thesis, we address many problems formulated in this way.

1.3 Connectivity specifications

There are several natural connectivity specifications that have been identified. We first survey specifications that require one-connected or acyclic networks. The *spanning tree* [98, 127] specification is a special case of a Steiner tree in which all the nodes in the graph are specified as terminals. Thus a spanning tree is a subgraph which connects all the nodes. A *generalized Steiner forest* [97, 2] generalizes the notion of a Steiner tree. Given a set of *site-pairs* of nodes, a generalized Steiner forest is a network in which there is a path between every site-pair. Note that this specification can be easily used to model that of a Steiner tree: choose any terminal in a Steiner tree specification and specify each pair formed by pairing the chosen terminal with every other terminal as a site-pair in the generalized Steiner forest specification. Any network satisfying these specifications must contain a path from every terminal to the chosen terminal and thus connects all the terminals. A generalized Steiner forest is sometimes referred to a fixed point-to-point interconnection network [101]. An interesting generalization, *a non-fixed point-to-point interconnection network* [101] is formulated as follows: given an equal number of producer and consumer nodes, find a subgraph in which every connected component has the same number of producer and consumer nodes. We can think of a generalized Steiner forest specification as one that specifies not only the producers and consumers but the pairing between them as well. Another interesting class of one-connected networks are *T-joins* introduced by Edmonds and Johnson [38]. Given an even number of T-nodes in an undirected graph, a T-join is a subgraph in which the nodes of odd degree are exactly the T-nodes. An edge-minimal T-join for a set of two nodes is a path between them; a perfect matching is a T-join when all the nodes of the graph are specified as T-nodes. Thus the T-join generalizes both these notions.

The connectivity specifications mentioned above can be generalized by seeking more than one-connectivity between the pairs of nodes that need to be connected. Such a redundancy requirement may be important in building resilient networks in commercial applications.[1] Thus each of the specifications above can be generalized by requiring higher connectivity between nodes where only one-connectivity was required earlier. For instance, the Steiner tree problem may be generalized to a two-connected network problem as follows: given an undirected graph with a subset of the nodes called the terminals, find a subgraph in which each pair of terminals are two-edge-connected (two-vertex-connected). Any solution network for this problem must contain two edge-disjoint (node-disjoint) paths between each pair of terminals.

Networks that can survive link or node faults are termed *survivable networks*. The design of

---

[1] A recent television advertisement for the AT & T telephone company highlighted a FASPAR network that contained the capability to re-route interruptions due to link failures almost instantaneously using an alternative path in the network. This can be achieved by using an underlying two-edge-connected network.

such networks has received considerable attention in the past [66, 116, 117, 144]. The higher connectivity requirement may be an edge-connectivity requirement or a node-connectivity requirement. In this thesis we address both these generalizations. In the formulation of a network-design problem with nonunit edge-connectivity requirements, we can consider two distinct versions: one in which multiple copies of an edge may be used in achieving higher connectivity, and the other where no edge is allowed to be duplicated in the solution. The latter version of the problem is more challenging and the one we consider in this thesis.

1.4 Optimization Objectives

For each of the above specifications of connectivity requirements, the optimization objective in the corresponding network-design problem can usually be expressed as minimizing some notion of "cost" associated with the network. This cost may reflect the price of constructing the network, it may measure some form of vulnerability of the network, or it may even measure some critical price incurred in using the network.

We address three classic examples of such cost measures in this thesis. If we associate costs with feasible edges and nodes that can be used to build the network, then a natural optimization objective is to minimize the price of construction of the network. This is the *minimum-cost network-design* problem and has been well studied [1, 2, 4, 48, 60, 74, 90, 98, 100, 127, 137, 156]. A notion of cost that reflects the vulnerability of the network to single-point failures and also quantifies the amount of decentralization in the network is the maximum degree of any node in the network. Minimizing this notion of cost corresponds to the *minimum-degree network-design* problem, which has also been well studied [3, 49, 50, 131]. A cost measure that captures the notion of price incurred in using a network is the maximum cost of any edge in the network, also termed the the bottleneck cost of the network. If the cost of an edge reflects the geographical distance between its endpoints, then minimizing the bottleneck cost in a network corresponds to minimizing the maximum distance traveled in a single hop in the network. Problems in which the objective is to keep the bottleneck cost low are termed *bottleneck problems* and these have also received considerable attention [20, 72, 123].

In many applications that arise in real-world situations, the network to be built is required to minimize more than one of these cost measures simultaneously. Recent papers have identified many problems [8, 85, 89, 161] wherein multiple objectives are specified in the statement of the problem. We formulate such multi-objective problems in the area of network design and provide approximation algorithms in this thesis.

The optimization objective may also involve maximizing some notion of desirability in the network. In building one-connected networks, one such notion may be the number of pendant vertices in it. In this thesis, we consider the maximum-leaf spanning tree problem and provide approximation algorithms for this network-design problem as well.

1.5 Approximation algorithms

Like many important problems in combinatorial optimization, the Steiner tree problem and most of its variants are NP-complete to solve exactly[53, 83]. The class of NP-complete problems represents a colossal collection of polynomially-equivalent problems for which no polynomial-time algorithms are known. One of the outstanding open questions in computational complexity theory has been "$NP = P$?", i.e., whether the class of NP-complete problems admits a polynomial-time solution.

An emerging pragmatic trend in overcoming the NP-hardness of important practical problems is to compromise on the quality of the solution for the sake of computing a suboptimal solution quickly. This leads to the following natural definition of an approximation algorithm for an optimization problem. An approximation algorithm runs in time polynomial in the problem size and outputs a solution with a small multiplicative error. The multiplicative error of an approximation algorithm for a minimization problem is the maximum value, over all instances of the input, of the ratio of the solution value output by the algorithm to the optimum solution value for the instance[2]. This multiplicative error is also commonly referred to as the approximation factor or the performance guarantee or ratio of the approximation algorithm. This thesis presents approximation algorithms with performance ratios that are typically a constant or a slow-growing function like the logarithm of the input size. While the latter may sound ludicrous especially in the context of a multiplicative error, we show that in some cases, this is nearly best-possible in the following sense: no polynomial-time approximation algorithm has a substantially better performance ratio unless something nearly equivalent to $P = NP$ is true. In other words, finding approximation algorithms for such problems with considerably better performance ratios is tantamount to nearly solving the "$P = NP$?" question.

1.6 Performance guarantees

The performance guarantee of an approximation algorithm for an NP-complete optimization problem provides a good yardstick with which the relative difficulty of NP-complete problems can be determined. Intuitively, problems with approximation algorithms having good performance guarantees may be deemed easier than those without. This notion can be formalized by using results on lower-bounds on the approximability of NP-hard problems. Such lower-bound results are also termed non-approximability results since they demonstrate non-approximability of the the problem in question to better than a certain factor in polynomial time unless something nearly equivalent

---

[2] The multiplicative error is the reciprocal of this quantity for a maximization problem.

to $P = NP$ is true. Early results on non-approximability [53, 161] were sporadic and used very specialized techniques. The recent connection drawn between results from the study of interactive proof systems from communication complexity theory [5, 6, 41, 110] and proving non-approximability results has provided a unified way to prove such results.

Another tool that has been useful in applying non-approximability results is a strengthening of the notion of an NP-hardness reduction called an *L-reduction* [122]. An L-reduction reduces an NP-hard problem to another with extra care so that any approximation algorithm for the first can be used to derive one for the second with performance guarantee at most a small constant factor more than that for the first problem. Papadimitriou and Yannakakis defined this notion along with a class of maximization problems called MAX SNP. They showed that all problems in MAX SNP are interreducible to one another using L-reductions. As a corollary, they derived that either all problems in this class permit very good approximations in polynomial time or all of them do not, unless $P = NP$. Recent results on probabilistically checkable proofs [6] show that the latter is the case.

Lower bounds on approximability vary from very weak to relatively strong depending on the factor of approximation that they rule out. For an NP-complete problem with an integral solution value, a trivial consequence of NP-completeness is that no polynomial-time approximation algorithm for the problem can provide a solution with an additive error less than one unit from the optimum unless $P = NP$. Arriving at such approximations is as hard as solving the original problem. A slightly stronger type of a non-approximability result rules out polynomial-time approximation schemes for the problem at hand. For any fixed error parameter $\epsilon > 0$, a polynomial-time approximation scheme runs in time polynomial in the input size and outputs a solution with multiplicative error $1+\epsilon$. If the dependence of the polynomial running time of the algorithm on $\frac{1}{\epsilon}$ is not exponential in $\frac{1}{\epsilon}$ but only polynomial in $\frac{1}{\epsilon}$, then the scheme is termed fully polynomial. Problems in the class MAX SNP that we talked about earlier do not have a fully polynomial-time approximation scheme unless $P = NP$ [6]. Even stronger non-approximability results rule out approximations with multiplicative factors of a constant or better than a logarithm of the input size. Lund and Yannakakis [110] recently proved such a hardness result for the set-cover problem in which given a collection of sets over a ground set, the objective is to find a minimum-size subcollection that contains all the elements of the ground set. They showed that no polynomial-time approximation algorithm for the set-cover problem achieves an approximation factor smaller than one-fourth the natural logarithm of the size of the ground set unless Deterministic Time $n^{\mathrm{polylog} n}$ contains NP. The strongest non-approximability results are those that show that performance factors nearly as bad as the trivial ones cannot be achieved in polynomial time unless $P = NP$. Examples are the maximum independent set problem and the graph coloring problem [41, 110, 13], for which no polynomial-time approximation algorithms can have performance ratios better than n^{α} for some fixed $\alpha > 0$ unless $P = NP$.

Just like the non-approximability results, approximation algorithms that have been designed also vary from weak, reasonable, good to nearly best-possible. Weak approximations are those with performance ratios nearly as bad as what is trivially achievable by using any feasible solution. A large number of problems are known to have only such approximations [16, 18, 94, 95, 132], and are good candidates for proving strong non-approximability. Reasonable approximation factors are typically slow-growing functions of the input size like the logarithm. Many known approximation algorithms have poly-logarithmic performance guarantees [25, 54, 61, 69, 79, 89, 91, 103, 107, 126, 128, 129, 154]. Good approximation algorithms have constant performance ratios [9, 22, 74, 156, 164] while best-possible approximation algorithms achieve guarantees that cannot be better unless $P = NP$ [50, 152]. Nearly best-possible approximation algorithms have performance guarantees that are within a constant factor of what has been shown to be best-possible [25, 91] unless $P = NP$. Such approximations are especially interesting for problems for which what is known to be best-possible in terms of approximation factors is a quantity larger than a constant, like the set-cover problem. This thesis describes approximation algorithms with performance guarantees ranging from weak to nearly best-possible.

1.7 Results

All the results presented in this thesis have the following general outline. We consider problems in network design consisting of a connectivity specification and an optimization objective as mentioned before. The goal in all these problems is to find a feasible network satisfying the connectivity specifications that is optimal with respect to the objective. All the problems we consider in this thesis are NP-complete. We present in this thesis the first approximation algorithms for each of these problems. All our algorithms output solutions that are guaranteed to be near-optimal in terms of the objective. All but one of our algorithms run in polynomial time. An overview of the details of our results is presented in Table 1.1. Here n denotes the number of nodes in the input graph.

The first three problems consider the case when the goal is to find a network of minimum cost when edges and/or nodes have nonnegative costs. The first result addresses finding biconnected subgraphs while the second is on finding two-edge-connected networks. The third row describes a result on approximating node-weighted networks. The fourth row addresses the problem of finding networks of minimum degree. The fifth row considers the problem of finding networks that have simultaneously low degree and cost. An interesting corollary of this result is an algorithm for the minimum-degree generalized Steiner forest problem in the fourth row with a better (logarithmic) performance guarantee in polynomial time. The details of this improvement are in the relevant chapters (Chapters 5 and 6). The last problem in the table addresses finding spanning trees with the maximum number of leaves.

Connectivity Specification	*Optimization Objective*	*Performance Ratio*	*Running Time*
Biconnected Steiner subgraphs	Minimize total edge-cost	5	$O(n^3)$
Two-edge-connected generalized Steiner subgraphs	Minimize total edge-cost	3	$O(n^2 \log n)$
Generalized Steiner forests	Minimize total node-cost	$O(\log n)$	$O(n^4)$
Generalized Steiner forests, Two-edge-connected spanning subgraphs	Minimize the maximum degree	For any $\epsilon > 0$, $(1+\epsilon)$ times optimal plus $O(\log_{1+\epsilon} n)$	$n^{O(\log_{1+\epsilon} n)}$
Steiner trees Generalized Steiner forests	Minimize total node-cost subject to budget b on the maximum degree	Degree is $O(b \log n)$; Cost is $O(\log n)$ times optimal	$O(n^4)$
Spanning trees	Maximize number of leaves	5,3	$O(n^4), O(n^7)$

Table 1.1: Results in this thesis.

We note that Table 1.1 provides only an overview of the results in this thesis and does not contain the most general versions in which they may be stated. Many one-connected network-design problems can be formulated as finding a subgraph that covers a family of cuts in the graph: for a certain family of cuts in a graph, find a subgraph intersecting all the cuts in the family. The particular family of cuts depends on the problem to be formulated. Building on the work of Agrawal, Klein and Ravi [1, 2], Goemans and Williamson [60] formulated a minimum-cost one-connected network-design problem as such a cut-cover problem and provided an approximation algorithm with performance factor two. Their formulation captures many important one-connectivity requirements such as Steiner trees, generalized Steiner forests, T-joins, and non-fixed point-to-point interconnection networks. The generalized Steiner forest problem is a prototypical problem in this formulation. In each of the results in Table 1.1, the connectivity specification of generalized Steiner forest can be replaced by this more general one-connectivity specification. Also the specification of two-edge-connected generalized Steiner subgraphs may be replaced with a two-edge-connected version of this more general specification. We postpone precise description of these results until the appropriate chapters for the sake of concreteness.

1.8 Techniques

The challenge in the design of an approximation algorithm lies in the proof of the performance guarantee of the algorithm. This proof is typically arrived at by using a combinatorial estimate of the value of the optimal solution for the instance at hand. Even computing the exact value of the optimal solution for any instance is NP-hard. This necessitates the development of techniques to arrive at good estimates on the solution value for all instances. Very often, such techniques rely on discovering structural relationships between combinatorial quantities defined by the problem at hand. In this thesis, we demonstrate the efficacy of three different techniques of this form.

The techniques we use in proving the performance ratios can be classified broadly as follows:

1. The *primal-dual* technique which draws heavily from the theory of Linear Programming,

2. *local optimization* in which simple local changes are used to arrive at a locally-optimal approximate solution, and

3. A *solution-based decomposition* method in which the optimal solution is decomposed into simpler subparts, and an approximate solution is built out of such subparts.

We elaborate on these techniques below.

1.9 The Primal-Dual method

This technique uses the framework of linear programming to derive approximate solutions to NP-hard problems. This technique has been well-known since its application to approximate the set-cover problem [25, 79, 107]. There has recently been a lot of work in applying this technique to approximation algorithms [1, 2, 3, 9, 52, 55, 60, 61, 88, 103, 90, 154].

There are two distinct ways in which this method can be applied to approximation algorithms. We formulate the hard problem as an integer program and consider its fractional relaxation. We then derive the dual of this relaxation and interpret the dual combinatorially in terms of graph-theoretic quantities associated with the problem. Restricting the variables of the dual program to be zero-one or integral in general is a useful tool in arriving at such an interpretation [1, 2]. Another useful method that has been explored before [1] is to assign values to all dual variables uniformly, i.e., all non-zero variables are assigned the same value. Such assignments lead to lower-bounds for the problem at hand (assuming that this is a minimization problem) which are often good lower bounds in that the maximum value of the dual solution even under such restricted assignments is close to that (within a small factor) of an optimal solution to the hard problem. In this case, the duality gap or the ratio of the optimal solution to the hard problem and the fractional relaxation value (also equal to the dual value) is also bounded by a small quantity.

Apart from the use of this technique to formulate hard problems and arrive at good combinatorial lower bounds, we may use it even more directly in designing approximation algorithms. Classical primal-dual algorithms such as the matching algorithm [37] follow a basic framework to prove optimality of the solution that they find. These algorithms typically start with a primal infeasible solution and a dual feasible but suboptimal solution. Suppose that the problem at hand is a minimization problem. The algorithm works in iterations; in each iteration, the value of the dual solution increases by collecting appropriate dual assignments and the primal solution is augmented so as to progress towards feasibility. These iterations are guided by using the complementary slackness conditions for the two dual linear programs. During each iteration, the value of a selected set of dual variables are increased uniformly until one of the slackness conditions becomes tight. This is followed by a primal solution augmentation and identification of new dual variables to update in the next iteration. Through such iterations, the value of the dual solution collected and that of the partial primal solution are shown to be equal using the complementary slackness conditions. At termination the primal solution is feasible and by the equality of this solution value and that of the algorithmically constructed dual solution, it is also optimal. We can use the same framework in designing an approximation algorithm except that we cannot hope to show optimality of our final solution. Instead, we may maintain an approximate version of the complementary slackness condition that ensures in the end that the value of the final primal solution is close to that of the dual solution collected through the algorithm. Since the dual solution is a lower bound on the cost of any primal solution by weak duality, the performance guarantee of the algorithm is proved. This technique yields an approximation algorithm for computing a near-optimal dual solution as well. This technique was used in a paper of Agrawal, Klein and Ravi [1, 2] and made more explicit in the subsequent work of Goemans and Williamson [60]. We use the approximate equality between two combinatorial quantities proved in [2] in proving the performance guarantee in Chapter 2. We also extend and use this technique combined with the idea of applying it in phases in Chapter 3. There has been a surge of work subsequent to ours in extending our techniques to higher-connected network problems [52, 61, 154].

1.10 Local Optimization

Local optimization has been a well-known heuristic for hard problems even before the theory of NP-completeness was formalized [30]. There have been many practically successful applications of this heuristic to many important hard problems. Some notable examples are its applications to the graph partitioning [84, 43] and the Traveling Salesperson problems [102]. More recently, there has been some work on complexity issues related to this method [80, 120].

The underlying idea in this method is simple: start with an arbitrary solution, apply local changes that improve the quality of the solution until a locally-optimal solution is reached, and output this solution. There are several randomized variants of this basic technique variously termed genetic algorithms [58], tabu-search, and simulated annealing [151]. However, none of these various heuristic techniques have been known to be approximation algorithms in the sense we have described. A theoretical breakthrough in this area came with the work of Fürer and Raghavachari [50]. They showed that a variant of the local search heuristic can be applied to the problem of computing a spanning tree of an undirected graph in which the maximum degree of any node is minimum. Their work provides polynomial-time approximation algorithms with good performance ratios. In Chapter 5, we extend their results and show how they can be applied to more general minimum-degree problems. In Chapter 7, we go one step further and show an application of this technique to approximating the maximum-leaf spanning tree problem. This later result significantly adds to the handful of problems that are known to have approximation algorithms based on local search.

There are two simple features that we may use in analyzing algorithms based on this technique. We typically use the progress in the quality of the solution at hand as a potential function to argue that these algorithms terminate quickly. Second, we use constraints on the structure of a locally-optimal solution to prove the performance guarantee.

1.11 Solution-based Decomposition

The solution-based decomposition technique has been implicitly used in designing many well-known approximation algorithms. In Christofides' classical $\frac{3}{2}$-approximation for the Traveling Salesperson Problem on a graph with triangle inequalities [22], the optimal TSP tour is decomposed into two equal-sized matchings. This leads to the observation that the minimum cost of a perfect matching in this graph is at most half the minimum cost of a TSP tour. A generalization of this observation is the key to the performance analysis of Christofides' heuristic. This heuristic illustrates the basic feature of the solution-based decomposition technique: decompose the optimal solution to the hard problem into one or many solutions to a simpler or computationally tractable problem. The approximation algorithm based on this insight finds such simpler pieces optimally and using the decomposition, proves that an approximate solution constructed using these simpler pieces is near-optimal. Implicit applications of this technique can be found in [48, 49].

We use this technique to provide the first approximation algorithms for node-weighted network-design problems in Chapter 4. We also use this technique in our algorithms for multi-objective approximations in Chapter 6.

1.12 Road Map

We present our results in the order in which they appear in Table 1.1. Chapters 2 and 3 address minimum-cost two-connected network-design problems. Chapter 4 addresses node-weighted network-design problems. Chapter 5 examines minimum-degree network problems. Chapter 6 contains our results on multi-objective approximation algorithms for network design. Chapter 7 describes our results on the maximum-leaf spanning tree problem. We close with Chapter 8 presenting some conclusions and directions for future work.

Chapter 2

Biconnected Steiner Subgraphs

2.1 Problem Definition

In this chapter, we begin our study of minimum-cost network-design by addressing the minimum-cost biconnected network-design problem mentioned in the first row of Table 1.1. This is a generalization of the problem of increasing the connectivity of a given graph to two at minimum cost. Given an edge-weighted graph, a subset of the vertex set called the terminals and an initial subgraph, the objective in this problem is to find a minimum-weight augmentation of edges that makes the terminals two-vertex-connected. No edge in the initial subgraph can be used in the augmentation. Moreover, the augmentation cannot contain more than one copy of any edge. We present an approximation algorithm for this problem with performance ratios of four if the initial subgraph connects the terminals and five otherwise.

Fredrickson and Ja'Ja' [46] were the first to consider the problem of augmenting a given graph to make it two-connected. This is a special case of our problem in which all nodes are terminals. They showed that this problem is NP-hard and provided approximation algorithms. Subsequently, Khuller and Thurimella [86] presented a more efficient and simplified version of the results of Frederickson and Ja'Ja' [46]. The performance guarantee of these algorithms is two if the initial subgraph is connected and three otherwise. The results of Frederickson and Ja'Ja' as well as that of Khuller and Thurimella hold for both the edge and vertex-connected versions of the problem.

Watanabe, Higashi, and Nakamura [153] address a closely related augmentation problem using techniques similar to ours. In [153], Watanabe, Higashi, and Nakamura have presented approximation algorithms for the following augmentation problem: given an edge-weighted undirected graph, an initial subgraph G_0 that connects *all* the nodes, and a specified subset S of the nodes, find a minimum-cost augmentation of the initial subgraph G_0 so that each pair of nodes in S is two-connected (they consider both edge- and vertex-connectivity) in the augmented graph. They provide approximation algorithms for these problems with performance guarantees two and four for the edge- and vertex-connected versions respectively. Though their technique appears to be similar to ours, they left open the case when the initial subgraph G_0 does not span all the nodes in the graph. We obtain the same performance ratio as [153] for the biconnected problem under the weaker assumption that the initial subgraph G_0 connects only the nodes specified in S. Furthermore, even when G_0 does not connect the nodes in S, we can derive approximation algorithms with a slightly worse guarantee of five. This latter problem was left open for future research in [153].

In the next chapter, we present results on finding minimum-cost two-edge-connected networks. Using these results, one can derive approximation algorithms for the two-edge-connected version of the problem we consider in this chapter. The performance ratios for the two-edge-connected Steiner subgraph problem that we can obtain using the results in the next chapter are two when the initial subgraph connects the terminals and three otherwise. Note that this matches that of Watanabe, Higashi, and Nakamura for the two-edge-connected case.

2.2 Strategy and Results

Before we present our results, we describe a simple strategy that allows us to transform an instance of an augmentation problem to an instance of a minimum-cost problem. Let $G = (V, E)$ be an undirected graph with a nonnegative weight function w on its edges. A subset S of nodes is specified as the *terminal set*. Given an initial subgraph G_0 of G, the augmentation problem is to find an edge-augmentation of minimum weight such that the augmented G_0 biconnects the terminals. This problem is essentially equivalent to setting the weight of all the edges in G_0 to zero and finding a minimum-cost subgraph of the resulting graph in which all the terminals are biconnected. Hence we restrict our attention to approximating minimum-cost biconnected Steiner subgraphs (i.e., subgraphs biconnecting every pair of terminals).

There has been previous work on polynomial-time algorithms for the minimum-cost two-connected Steiner subgraph problems for special classes of graphs such as outerplanar graphs [157], series-parallel graphs [158] and Halin graphs [29]. Our main results are approximation algorithms for the case of general graphs and we state them below.

Theorem 2.2.1 *There is a polynomial-time approximation algorithm for the minimum-cost biconnected Steiner subgraph problem. The performance guarantee of the algorithm is five.*

When the initial subgraph G_0 in the augmentation problem connects all the terminals, we can obtain a slightly stronger result than Theorem 2.2.1. In this case, we can transform the augmentation problem to a minimum-cost tree-biconnecting problem. To do this, we choose an edge-minimal Steiner tree from the edges in G_0 (which is guaranteed to connect all the terminals, and hence contains such a tree), and set the cost of all the edges in G_0 not in this tree to zero. We now seek a minimum-cost set of edges disjoint from the tree edges whose addition to the tree biconnects all the nodes in the tree. The resulting minimum-cost tree-biconnecting problem for this Steiner tree is easily seen to be equivalent to the original augmentation problem. We have the following result for the tree-biconnecting problem.

Theorem 2.2.2 *There is a polynomial-time approximation algorithm for the minimum-cost tree-biconnecting problem. The performance guarantee of the algorithm is four.*

The key idea in all our algorithms is to construct an auxiliary graph on a subset of the nodes in the original graph and to select a tree spanning these nodes. We then use the algorithms of Frederickson and Ja'Ja' [46] or those of Khuller and Thurimella [86] to find an approximately minimum-cost augmentation of this spanning tree to biconnect it. Finally we transform this augmentation to an augmentation that biconnects the tree in the original graph. To show the performance guarantee we use a simple property that an edge-minimal augmentation that biconnects a Steiner tree is acyclic. This observation may be of independent interest.

In addition to constructing the auxiliary graph, we also adapt and use an approximate min-max equality relating approximately minimum-cost Steiner trees to packing of cuts derived by Agrawal, Klein, and Ravi [2] to prove the performance guarantee of five in Theorem 2.2.1.

In the next section, we introduce some definitions. Then we prove Theorem 2.2.2. Finally, we introduce some background on the approximate min-max equality proved in [2] and use it along with Theorem 2.2.2 to prove Theorem 2.2.1.

2.3 Definitions

Let $G = (V, E)$ be the given undirected graph with nonnegative weights w assigned to its edges and let $n = |V|$ and $m = |E|$. The weight of any subgraph is the sum of the weights of the edges in it. An undirected edge between two nodes u and v is represented by (u, v). For a graph G and a node v in G, we use $G/\{v\}$ to denote the graph obtained from G after deleting v and all its incident edges.

An undirected graph is *(one-)connected* if for any two nodes u and v, there is an undirected path from u to v in the graph. A *cutvertex* is a vertex whose removal along with its incident edges disconnects G. A graph is two-vertex-connected or *biconnected* if it contains no cutvertices. Note that in a biconnected graph, there always exists two node-disjoint paths between any pair of nodes. A *block* is a maximal biconnected subgraph of a given undirected graph.

2.4 The Steiner Tree Augmentation Problem

In this section, we prove Theorem 2.2.2. We are given an edge-minimal Steiner tree $T = (V_T, E_T)$ spanning the terminals. The problem is to find a minimum-cost set of edges from $G - E_T$ that will make all the nodes in T biconnected.

The key to deriving our results is the the following important property of any edge-minimal augmentation that biconnects[1] any tree T.

Lemma 2.4.1 *For any tree T, any edge-minimal set of edges that biconnects T is acyclic.*

Proof: Any set $H \subseteq E/E_T$ of edges biconnects T if and only if for every internal node v of T, $H \cup T/\{v\}$ is connected. Let B be an edge-minimal augmentation that biconnects T. Suppose B contains a cycle C . Without loss of generality we may assume that the cycle C is simple. We show that an edge e can be removed from this cycle in B while maintaining that, for any internal node v of T, the graph $B \cup T/\{v\} - e$ is connected. There are few cases to consider.

Case 1: The cycle contains no node of T. In this case, for any internal node v of T, the cycle C is also a cycle in $B \cup T/\{v\}$. Removing any edge e from C leaves $B \cup T/\{v\} - e$ connected.

Case 2: The cycle contains at most one node u in T. For an internal node v of T, if $v \neq u$, then this reduces to the previous case. If $v = u$, then it is easy to see that on removing an edge e that is incident on v from the cycle, $B \cup T/\{v\} - e = B \cup T/\{v\}$. Since $B \cup T/\{v\}$ is connected, so is $B \cup T/\{v\} - e$.

Case 3: The cycle C in B contains two or more nodes of T. Define a segment to be a maximal subpath of C none of whose internal nodes belong to T. We partition the cycle C into two or more segments going between nodes of T. As before, we wish to identify an edge $e \in C$ whose deletion from B leaves a set of edges that biconnects T. We use the following observation to identify such an edge.

Proposition 2.4.2 *Let T be a tree and B an augmentation that biconnects T. Suppose there exists a path P in $T \cup B$ consisting entirely of edges of B such that every internal node of P is not a node of T and has degree exactly two in $T \cup B$, and the endpoints of P lie in a simple cycle in $T \cup B - P$. Then $B - P$ biconnects T as well.*

The proof of the above proposition is immediate from the observation that a chord of a circuit in a biconnected graph can be removed without violating biconnectivity (See, for example, [107], Problem 6.35).

To use this observation, we first identify a segment S of the cycle C whose endpoints lie in a simple circuit in the graph $B \cup T - S$. However, we may not be able to delete the whole segment

---

[1] By "an augmentation that biconnects T", we mean one that biconnects every pair of nodes in T.

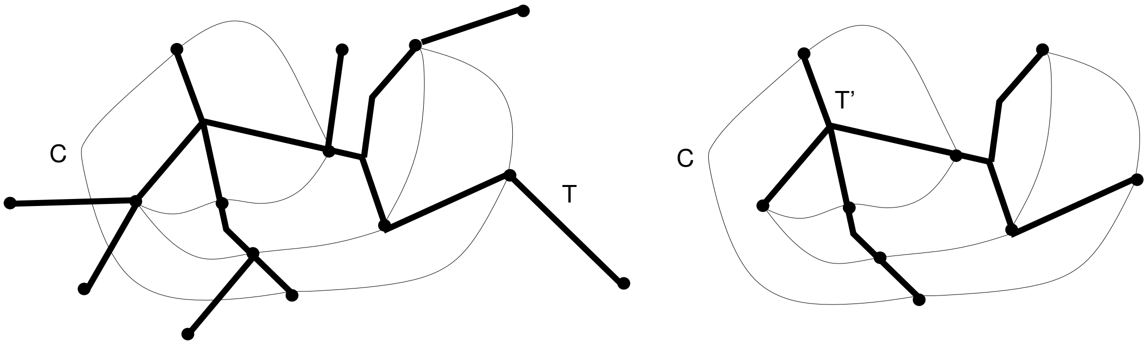


Figure 2.1: Using a cycle C to identify a subtree T' of tree T. The cycle C wraps around T'.

since it may contain nodes with degree more than two. But we show that we can always identify an edge in such a segment that can be deleted without affecting the biconnectivity of the tree T.

Claim 2.4.3 *There is a segment S of the cycle C such that the endpoints of S lie on a simple cycle in $B \cup T - S$.*

Proof: To prove the claim we restrict our attention to a subtree T' of the tree T made up of edges of T that lie on simple paths in T between nodes of C. We say that the cycle C *wraps around* the tree T' (See Figure 2.1). We show that there is a segment S of C whose endpoints lie in a simple cycle in $T' \cup C - S$. Note that the leaves of T' are in the cycle C by construction. We show the existence of the segment S in the following proposition.

Claim 2.4.4 *Let C be a cycle that wraps around a tree T_C. Let DS be a distinguished segment of the cycle C. Then there is a segment S in $C - DS$ such that the endpoints of S are in a simple cycle in $C \cup T_C - S$.*

Proof: The proof is by induction on the number of segments in the cycle C. The basis is trivial when the cycle has exactly two segments one of which is DS. In this case, the other segment, $C - DS$, has two node-disjoint paths between its endpoints, namely, DS and the tree path between these nodes. Hence this segment obeys the condition in the claim.

Assume the proposition holds for all cycles with at most k segments; we now prove it for a cycle C with $k + 1$ segments. Let S be a segment adjoining the distinguished segment DS, and let c and d be its endpoints where d adjoins DS. Consider the graph $T_C \cup C - S$. If this graph is biconnected, then the nodes c and d lie on a simple cycle, and the segment S satisfies the condition in the Claim. Otherwise, there is a cutvertex in this graph. Since $T_C \cup C$ is biconnected, any cutvertex in $T_C \cup C - S$ must lie on the path in T_C between c and d. Let t be such a cutvertex. Since C wraps around T_C, note that $C - S$ is a simple path from c to d as well. However, since

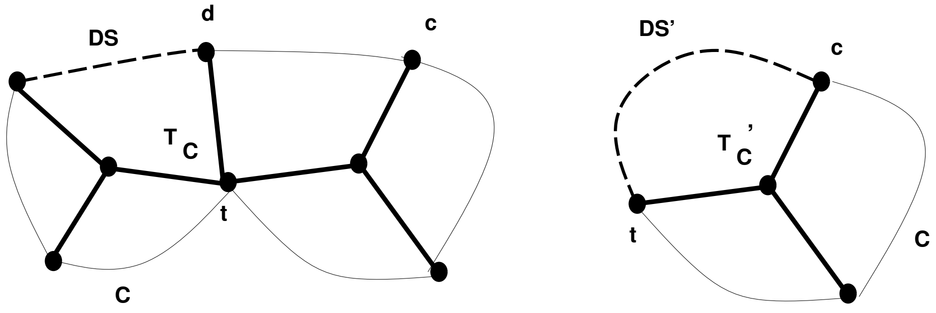


Figure 2.2: The induction step in the proof of Claim 2.4.4. By replacing a subpath of the cycle C with a segment DS', we form a cycle C' with fewer segments than C. The induction hypothesis can be applied to C'.

$T_C \cup C - S$ is not biconnected and contains t as a cutvertex, t must also lie in $C - S$. We can now clump several segments of C as a single distinguished segment to form a cycle C' to which the induction hypothesis can be applied. To do this, we consider the subpath of C between t and c containing DS as a single distinguished segment DS'. The new cycle C' is formed from C by replacing the subpath of C between t and c containing DS with a single segment DS' (See Figure 2.2). Note that C' has at most k segments. Let T'_C be the subtree of T_C that the cycle C' wraps around. By the induction hypothesis, there is a segment S other than DS' in C' whose endpoints are in a simple cycle in $C' \cup T'_C - S$. This segment is also in $C - DS$ and the simple cycle in $C' \cup T'_C - S$ corresponds to a simple cycle in $C \cup T_C - S$. This completes the proof of Claim 2.4.4. □

To prove Claim 2.4.3, we choose any segment of the cycle C as a distinguished segment and apply Claim 2.4.4. □

Now we return to complete the proof of Case 3. We use Claim 2.4.3 to identify a segment S whose endpoints c and d lie on a simple cycle in the graph formed after the removal of the segment S . If all nodes in the segment S are of degree two in $T \cup C$, then we can delete the whole segment and retain biconnectivity by Proposition 2.4.2. Otherwise there is at least one node in S of degree three or more in $T \cup C$. Let c' and d' denote the nodes of degree three or more in S that are closest to c and d respectively. We show that a portion of the cycle C in Case 3 can be removed while retaining biconnectivity. This proves that B contains no cycle of this type either and completes the proof of Lemma 2.4.1.

Claim 2.4.5 *At least one of the subpaths of the segment S from c to c' or from d to d' can be deleted from B while retaining biconnectivity of $T \cup B$.*

Proof: The proof is by contradiction. Suppose neither of the subpaths can be deleted from B while maintaining biconnectivity. Then on removing each of these subpaths, there are cutvertices in each of the remaining graphs.

Consider the graph remaining after deleting the subpath from c to c'. By choice of S, c and d lie in a cycle even after deleting S and hence are in the same block in this graph.

Since c' has degree three or more, we can follow an edge of B incident to c' but not in the cycle C along a path. Note that this path cannot lead back to the segment S without intersecting a node in T for otherwise we would have identified a cycle in B with no node in T. This is disallowed by Case 1. Therefore this path meets a node t_1 of T. Now we can follow the path in the tree from t_1 to d (Note that t_1 may be d in which case this path is empty) and then the segment S from d to c' to form a simple cycle containing c' and d. Thus c' and d are in the same block in the graph formed after deleting the subpath from c to c'.

Since $B \cup T$ is biconnected, any cutvertex formed on removing the subpath from c to c' must lie on a path between c and c' in the remaining graph. Since c and d lie on the same block even after deleting this subpath, this cutvertex must lie in the subsegment of S from c' to d. Since c' and d' are in the same block after deleting the subpath from c to c', d must be a cutvertex formed on deleting the subpath from c to c'. Also, in proving that c' and d are in the same block, note that we have identified a path P_1 between c' and d through the node t_1 which avoids the segment S.

By a symmetric argument, one of the cutvertices formed on removing the subpath of S from d to d' must be c, and there is a path P_2 between d' and c avoiding S in the block containing c and d'.

Suppose $P_1 \cap P_2 = \emptyset$. Then following the path along the segment S from c' to d' and then P_2 between d' to c, then a path between c and d in the cycle containing them and finally the path P_1 from d to c' identifies a simple cycle containing c and c' (See Figure 2.3). This contradicts the assumption that removing this subsegment violates biconnectivity of $T \cup B$.

Otherwise P_1 and P_2 share a node. Let P_1' be the initial subpath of P_1 from c' until it intersects P_2 at a, say (P_1' may be empty, e.g., when $c' = d' = a$). Now we can follow the subpath P_1' from c' to a, then the subpath of P_2 from a to c followed by the path between c and d and finally the path in the segment S from d to c' to identify a simple cycle containing c and c'. This gives a contradiction as well and completes the proof of Claim 2.4.5. □

This also completes the proof of Lemma 2.4.1. □

Now we are ready to describe the algorithms in Theorem 2.2.2. We construct an auxiliary multigraph T' on the nodes V_T. The edge-set of T' is $E' = E_T \cup E_a$ where $(u, v) \in E_a$ whenever $u \neq v$ and there is a path from u to v in $G - E_T$ all of whose internal nodes are not in T. Note that $\cup$ represents multiset union here. Each edge (u, v) in E_a is assigned weight equal to the weight

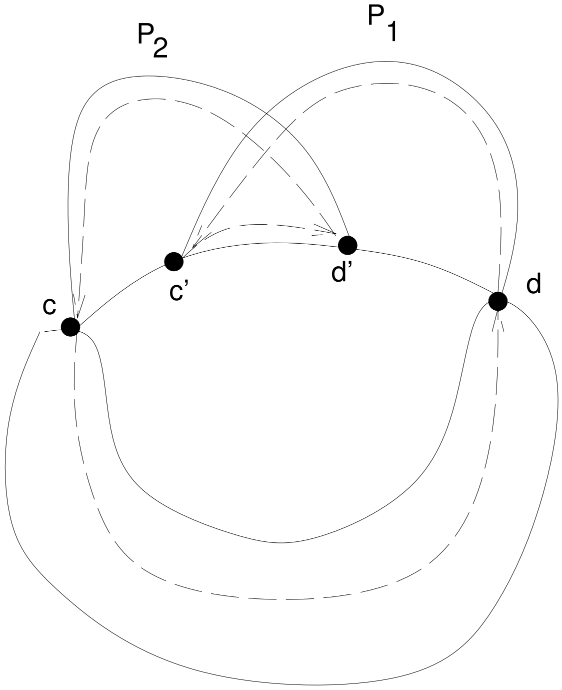


Figure 2.3: Figure illustrating the proof of Claim 2.4.5. A simple circuit is identified between the endpoints of a subpath of S, contradicting the assumption that this subpath cannot be removed from B without violating biconnectivity of $T \cup B$.

of a shortest path from u to v in $G - E_T$ all of whose internal nodes are not in T.

Lemma 2.4.6 *If $A \subseteq E/E_T$ is a set of edges of weight $w(A)$ that biconnects T in G, than there is an augmentation A' of weight at most $2w(A)$ that biconnects T in T'.*

Proof: Since all edge-weights are nonnegative, we can assume without loss of generality that A is edge-minimal, and hence by Lemma 2.4.1, A is acyclic. We now identify edges in E_a that form an augmentation of cost at most $2w(A)$ using the edges of A. Mark all the nodes of T in the subgraph induced by A. Perform an Euler tour of each tree in A and partition each tour into segments between marked nodes. Note that the sum of the weights of all such segments is exactly the sum of the weights of the Euler tours which in turn is $2w(A)$. Each segment between two marked nodes identifies an edge included in E_a between the endpoints of the segment which are nodes of T. It is easy to see that the union of all the edges in E_a identified by these segments biconnects the tree T in T'. Furthermore, the weight of all these edges is at most the weight of the segments used to identify them, and hence is at most $2w(A)$. $\square$

Now we can apply the techniques of Frederickson and Ja'Ja' [46], or that of Khuller and Thurimella [86], to find an approximately minimum-cost set of edges to augment the tree T in

T' to biconnect it. Since the optimal augmentation of T in G induces an augmentation in T' of at most twice the cost by Lemma 2.4.6, and since the performance guarantee of the approximation algorithms of [46, 86] is two, we finally get an augmentation of cost at most four times the minimum. Finally we transform the augmentation in T' that we find to the collection of shortest paths in the original graph that the edges in the augmentation correspond to. Noting that the construction of the auxiliary graph T' and the algorithms in [46, 86] are polynomial-time completes the proof of Theorem 2.2.2.

2.5 The Biconnected Steiner Subgraph Problem

In this section, we prove Theorem 2.2.1. To do this we need some background.

Packing of requirement cuts

Given a graph G and a subset of the vertices S specified as terminals, we define a *requirement cut* as a set of edges whose removal separates at least one terminal from another. Each requirement cut (henceforth referred to as a *cut*) can be written as $\Gamma(W)$, where W is a node subset of the graph, and $\Gamma(W)$ denotes the set of edges with exactly one endpoint in W. $\Gamma(W)$ is also referred to as the *co-cycle* determined by W. A fractional packing of cuts is a family of cuts $\Gamma(W_1), \Gamma(W_2), \ldots, \Gamma(W_k)$, together with a rational *weight* λ_i for each cut. A (fractional) *w-packing* of cuts is a weighted collection of cuts that have the following property: for each edge (u, v), the sum of the weights of all the cuts in this collection containing the edge is at most $w(u, v)$. The *value of the packing* is the sum of the weights of all the cuts in the packing. A *maximum packing* is one of maximum value.

Maximum packing of cuts and minimum-weight Steiner trees

Given an edge-weighted graph with a subset of the nodes specified as terminals, the problem of finding a minimum-weight Steiner tree connecting all the terminals is NP-complete. Since a requirement cut separates the terminals, any Steiner tree must have an edge in the cut. Thus it is easy to see that the weight of any Steiner tree is at least as much as the value of a maximum packing of requirement cuts. The following result was shown by Agrawal, Klein and Ravi [2].

Theorem 2.5.1 *Given an undirected graph with edge-weights, and a subset of the vertices specified as terminals, the minimum-weight of a Steiner tree connecting all the terminals is at most twice the value of a maximum packing of requirement cuts.*

The algorithm in [2] greedily finds a packing of requirement cuts and simultaneously builds a Steiner tree of weight at most twice the value of this packing.

Note that any biconnected Steiner subgraph must have at least *two* edges crossing any requirement cut since this subgraph has two node-disjoint paths between the terminals separated by this cut. Thus we have the following lemma.

Lemma 2.5.2 *The weight of any biconnected Steiner subgraph is at least twice as much as the value of a maximum packing of requirement cuts.*

Combining the results from Theorem 2.5.1 and Lemma 2.5.2, we get the following theorem.

Theorem 2.5.3 *Given an undirected graph with edge-weights, and a subset of the vertices specified as terminals, the approximately minimum-weight Steiner tree found by the algorithm in [2] has weight at most that of any minimum-weight biconnected Steiner subgraph.*

We can use the above theorem to handle the case when all terminals are not one-connected in the initial subgraph G_0. We first add edges to G_0 so as to one-connect all the terminals. Then we use the algorithm in the previous section to augment this tree. We use the above theorem in the first phase of one-connecting the terminals. For this, we set the weight of all the edges in G_0 to zero and use the approximation algorithm of Agrawal, Klein and Ravi in [2] to find an approximately minimum-weight Steiner tree one-connecting the terminals. By Theorem 2.5.3 above, the cost of the set of edges added in this first phase is at most that of a minimum-weight biconnected Steiner subgraph.

We use this tree as the starting point in applying the algorithm in the previous section. The minimum-weight biconnected Steiner subgraph also forms a solution to the tree-biconnecting problem for the tree T. Therefore, by Theorem 2.2.2, the cost of the tree-biconnecting edges added in this second phase is at most four times that of the minimum-weight biconnected Steiner subgraph. Summing the costs of edges added in both phases gives the bound in Theorem 2.2.1.

Chapter 3

Generalized Two-edge-connected Networks

We continue our study of minimum edge-cost network-design problems by considering an edge-connected generalization of the problem we addressed in the previous chapter. In this chapter, we present a general approximation technique for a class of network-design problems where we seek a network of minimum cost that satisfies certain communication requirements and is resilient to worst-case single-link failures. Our algorithm runs in $O(n^2 \log n)$ time on a graph with n nodes and outputs a solution of cost at most thrice the optimum. We extend our technique to obtain approximation algorithms for augmenting a given network so as to satisfy certain communication requirements and achieve resilience to single-link failures. In this chapter, two-connectivity refers to two-edge-connectivity unless explicitly stated otherwise.

The technique we introduce in this chapter allows one to find nearly minimum-cost two-connected networks for a variety of connectivity requirements. For example, our result generalizes earlier results on finding a minimum-cost two-connected subgraph of a given edge-weighted graph in [46, 86]. Using our technique, we can also approximately solve for the first time a two-connected version of the generalized Steiner network problem and a two-connected version of the non-fixed point-to-point connection problem. The results in this chapter were obtained jointly with Philip Klein, and appeared in [90].

3.1 Introduction: The Framework

On designing a network

The following scenario was proposed in [2]. You are in the job of providing communication links to customers. You have a set of clients with communication requirements; each client has specified a pair of cities (called a *site-pair*) between which the client must have communication capabilities. In front of you is the price list from the telephone company, which gives prices for constructing communication links between various cities. Each link built has essentially infinite communication bandwidth. Your job is to select a minimum-cost collection of communication links that can accommodate all your clients' communication requirements. The network you must construct need not be connected; all that is needed is that every client's pair of cities be connected through your network. Unfortunately, your job's NP-complete. So you come up with an approximately good solution [2] and build a network using this strategy.

Soon you notice that your clients are complaining. The links used in your network seem to be breaking down pretty often. And you notice that it is very often the case that it is just a single edge that has caused each tree in your solution to come to a standstill. So your clients want you to augment this network so that each pair of cities have at least two connections. Moreover, they do not want you to duplicate any link in the already existing network since the link failures seem to be related to the location of the cables. By duplicating links, you are only making *both* copies prone to failure. Well, your problem is NP-complete again [46]. If you were lucky to have a large enough clientèle so that your network spanned all the cities, then you could use the approximate solution in [46]. However, considering your past history with fortune, no approximation algorithms are known in the general case.

In this chapter, we give the first approximation algorithm for augmenting this generalized Steiner network to a network in which each site-pair is two-connected. Combining this with the solution offered to your earlier problem in [2], this provides an approximation for a general two-connected Steiner network design problem. In fact, you weren't so unlucky after all. For, while you set out to solve the problem of finding a minimum cost network two-connecting all the site-pairs, your technique turns out to solve a whole class of minimum-cost two-connected subgraph problems...

An example of a solution formed this way for the generalized two-connected Steiner network problem is depicted in Figure 3.1.

Framework

We can model the generalized two-connected Steiner network problem as follows: Define a cut in the graph to be *active* if it separates some site-pair. A network is said to *two-connect* a site-pair if there are two edge-disjoint paths between them in the network. Thus, any network that two-connects the site-pairs must cross this active cut twice. Conversely, if a network crosses every such active cut twice, the network indeed two-connects all the site-pairs. More formally, we can define a function f defining the activity of all cuts in the graph. If a cut C separates some site-pair, then we set $f(C) = 1$ and call C an *active* cut. Otherwise, we set $f(C) = 0$ and call C an *inactive* cut. Any set of edges that crosses each of the active cuts twice is a feasible solution to the two-connected Steiner network problem. We can generalize this two-connected network design problem by extending our notion of active cuts. For this, we turn to the work of Goemans and Williamson in [60] where they define the notion of *proper* functions. Proper functions are a class of functions used for defining the activity of cuts in a graph.

In [2], Agrawal, Klein and Ravi provided the first approximation algorithm for the generalized network Steiner problem. The algorithm presented there involved construction of feasible primal and dual solutions with an approximate relation between their values. Subsequently, Goemans and Williamson [60] generalized this result to a technique for approximating a large class of constrained forest problems. They introduced the notion of proper functions whose elegant definition captures the connectivity requirements for a variety of constrained forest problems. In this chapter, we use their notion of *proper functions* in formulating a general two-connected network design problem.

Given a graph $G = (V, E)$, a function $f : 2^V \to \{0, 1\}$ and a non-negative cost function c on the edges, we consider the following integer program:

$$\begin{array}{lcll} \text{Min } \sum_{e \in E} c_e x_e & & & \\ \text{subject to constraints:} & & & \\ x(\Gamma(S)) & \geq & 2f(S) & ; \emptyset \neq S \subset V \qquad \text{(IP)} \\ x_e & \in & \{0,1\} & ; e \in E \end{array}$$

Here $\Gamma(S)$ denotes the set of edges having exactly one endpoint in the set S and $x(F) = \sum_{e \in F} x_e$. Any feasible solution to the program above is a collection of edges that intersect each active cut under the function f at least twice. Thus, any minimal feasible solution to (IP) is a collection of two-connected subgraphs. In this chapter, we address solutions to the above program for the class of proper functions f. An important requirement of the solution subgraph that this chapter addresses is that edges are not allowed to be replicated. This will be the assumption in all the problems we consider. Thus, the program is feasible only if the input graph has a feasible solution as a subgraph. We shall assume that the input graph is itself feasible for (IP). It is not hard to verify this assumption. We define A, the set of *terminals* in V as the set $A = \{v \in V : f(\{v\}) = 1\}$. The following is our main result.

Theorem 3.1.1 *There is an $O(n^2 \log n)$ algorithm to produce an feasible solution F' to (IP) such that $\sum_{e \in F'} c_e \leq (3 - \frac{3}{|A|})OPT$ where OPT is the optimum value of (IP).*

Although we use weak linear programming duality in our analysis, our algorithm does not linear programming.

Formulating problems in this framework - Examples

The generalized two-connected Steiner network problem without edge replication is representable in the above framework. We set $f(S) = 1$ if there is a site pair that contains a site in S and one outside S, and we set $f(S)$ to 0 otherwise. The above framework also captures the following *point-to-point two-connection problem* we introduce. In this, we are given a set $C = \{c_1, \ldots, c_p\}$ of sources and a set $D = \{d_1, \ldots, d_p\}$ of destinations in the graph and we need to find a minimum-cost set F of edges such that each source-destination pair is two-connected in F. The fixed destination case, in which c_i is required to be two-connected to d_i, reduces to a case of the generalized two-connected Steiner network problem. However, we can also approximate the non-fixed destination case, where the only requirement is that each component of F contains the same number of sources and destinations. For this, we set $f(S)$ to 1 whenever $|S \cap C| \neq |S \cap D|$ and to 0 otherwise.

Strategy

The algorithm we present in this chapter works in two phases. In the first phase, the algorithm in [60] is applied to construct a partial solution F_1 that partially satisfies the requirements in (IP), namely, the solution satisfies the one-connectivity constraints: $x(\Gamma(S)) \geq f(S) \quad \emptyset \neq S \subset V$. In the second phase, we augment the partial solution F_1 such that the augmented subgraph two-connects each of the trees in the forest F_1. We show that the cost of the solution output at the first phase is at most the optimum value of (IP) and the cost of the augmentation output at the second phase is at most twice the optimum value of (IP) to obtain a performance guarantee of three. The second phase is a careful reapplication of the methods of Goemans and Williamson. We define an auxiliary function f' specifying a different notion of active cuts, based on the partial solution F_1. If f' were proper, then we could reapply the method in [60] directly. Unfortunately, f' is not proper and so we need to adapt this method.

Approximating augmentation

The techniques used in proving Theorem 3.1.1 can be adapted to provide approximation algorithms for minimum-cost augmentation. Suppose that we are given a graph $G = (V, E)$ along with an initial subgraph G_0 with edge set E_0. The *minimum-cost augmentation problem* is to choose a minimum-cost set of edges in G whose addition to G_0 yields a graph that obeys the constraints of (IP). This problem generalizes the minimum-cost bridge-connectivity augmentation problem considered in [46, 86] and also the Steiner version considered in Chapter 2. We have the following theorem about approximating the minimum-cost augmentation.

Theorem 3.1.2 *The augmentation problem described above can be approximated in* $O(n^2 \log n)$ *time within a factor of* $2(1 - \frac{1}{|A|})$ *if the incidence vector,* x_0 *of the initial subgraph* G_0 *obeys* *the one-connectivity constraints*

$$x_0(\Gamma(S)) \geq f(S) \quad \forall S \ : \ \emptyset \neq S \subset V$$

Otherwise the performance guarantee is $3(1 - \frac{1}{|A|})$.

In the next section, we look at related work. After that, we present background results and the basic technique used in proving our main result. In Section 3.4, we present the second-phase augmentation algorithm. In the following section, we prove the performance bound of this phase. Finally, we prove the results on augmentation.

3.2 Related Work

Fredrickson and Ja'Ja', in their pioneering work [46] on approximation algorithms for weighted graph augmentation problems, provided the first approximate solution for the problem of augmenting a given graph to be two-connected. The performance guarantee of their algorithm was a multiplicative factor of two. Their technique also gave an approximate solution to the minimum-cost two-connected subgraph problem. The performance guarantee is a multiplicative factor of three. Their solution for the latter problem starts with a minimum spanning tree and augments this to be two-connected. Khuller and Thurimella [86] have extended these results and provided more efficient algorithms for the same. In Chapter 2, we showed how to generalize this method achieving a weaker performance ratio to find minimum-cost two-connected Steiner subgraphs of a given graph. All the above methods are also able to solve the node two-connected versions approximately. Khuller and Vishkin have presented algorithms with better performance guarantees for the minimum two-connected subgraph problems in [87]. The bounds are $\frac{3}{2}$ for the two edge-connected case and $\frac{5}{3}$ for the two node-connected case.

When the cost function c obeys the triangle inequality, then Fredrickson and Ja'Ja' [47] present an adaptation of Christofides' heuristic to solve the minimum-cost biconnectivity augmentation problem with a performance factor of $\frac{3}{2}$. There has been considerable previous work on network survivability [59, 66, 116, 117, 144] that addresses constructing minimum-cost k-connected subgraphs under such cost functions. However, our interest is in the general case when the cost function does not satisfy the triangle inequality.

There has also been considerable work on characterizations of integer polyhedra arising from connectivity constraints [65], and construction of minimum-cost two-connected survivable networks [59, 118].

As mentioned in the previous section, Agrawal, Klein and Ravi [2] introduced a primal-dual method for approximating the minimum-cost generalized Steiner network. Though this result addressed arbitrary connectivity requirements between pairs of nodes, it allowed for edge replication.

This was consequently generalized by Goemans and Williamson in [60].

Our method combines the two-phase idea of Fredrickson and Ja'Ja' with a primal-dual method tailored to provide an approximate solution in both phases.

3.3 Background

Following [60], we define the notion of a proper function $f : 2^V \rightarrow \{0,1\}$. A function f is *proper* if the following properties hold.

- [Null] $f(\emptyset) = 0$;
- [Symmetry] $f(S) = f(V - S)$ for all $S \subseteq V$; and
- [Disjointness] If A and B are disjoint, then $f(A) = f(B) = 0$ implies $f(A \cup B) = 0$.

The functions f used to model the problems in the first section can be easily verified to be proper.

The first phase:

As mentioned in the introduction, our algorithm works in two phases. In the first phase, we construct an approximate solution to the following modified integer program:

$$\begin{array}{l} \text{Min } \sum_{e \in E} c_e x_e \\ \quad \text{subject to constraints:} \\ \quad \begin{array}{rcll} x(\Gamma(S)) & \geq & f(S) & ; \ \emptyset \neq S \subset V \qquad \text{(IP1)} \\ x_e & \in & \{0,1\} & ; \ e \in E \end{array} \end{array}$$

We do this using the method in [60]. Let the solution generated be denoted by F_1. We shall use F_1 to denote the *set of edges* in this solution returned in the first phase. It follows from [60] that F_1 induces a forest in G. The algorithm in [60] implicitly constructs a dual solution to the above linear program to prove the performance guarantee. This dual to the linear relaxation of the above integer program (using $x_e \geq 0$) is the following.

$$\begin{array}{l} \text{Max } \sum_{S \subset V} f(S) \cdot y_S \\ \quad \text{subject to constraints:} \\ \quad \begin{array}{rcll} \sum_{S: e \in \Gamma(S)} y_S & \leq & c_e & ; \ e \in E \\ y_S & \geq & 0 & ; \ \emptyset \neq S \subset V \end{array} \end{array}$$

Recall that A is the set of terminals in V. We will be using the following results from [60].

Theorem 3.3.1 ([60]) *The solution F_1 feasible for (IP1) and the dual y constructed are related by*

$$\sum_{e\in F_1} c_e \le (2-\frac{2}{|A|})\sum_{S\subset V} y_S$$

Lemma 3.3.2 ([60]) *Let $e \in N$ where N is a connected component of a minimal solution F_1. Let N_1 and N_2 be the two connected components of $N - \{e\}$. Then $f(N_1) = f(N_2) = 1$.*

Observation 3.3.3 ([60]) *If $f(S) = 0$ and $f(B) = 0$ for some $B \subseteq S$, then $f(S - B) = 0$.*

The LP relaxation of our integer program (IP) is obtained by replacing each integrality constraint $x_e \in \{0, 1\}$ with the linear constraint $0 \le x_e \le 1$. The dual to this LP relaxation is as follows.

$$\begin{array}{llll} \text{Max } \sum_{S\subset V} 2f(S)\cdot y_S - \sum_{e\in E} r_e & & & \\ \quad\text{subject to constraints:} & & & \\ \quad\quad \sum_{S:e\in\Gamma(S)} y_S & \le & c_e + r_e & ;\ e \in E \\ \quad\quad\quad\quad y_S & \ge & 0 & ;\ \emptyset \ne S \subset V \qquad\qquad \text{(LP)} \\ \quad\quad\quad\quad r_e & \ge & 0 & ;\ e \in E \end{array}$$

By weak linear programming duality, if OPT denotes the value of the optimal solution to (IP) and Z^*_{LP} denotes that of the optimal dual solution to (LP), then $OPT \ge Z^*_{LP}$. Note that the dual solution y provided by the algorithm of [60] is feasible for (LP) by setting $r_e = 0$ for all edges e. The value of the above program (LP) with this solution is $2\sum_{S\subset V} y_S$. Thus we have $OPT \ge 2\sum_{S\subset V} y_S$. Combining this with Theorem 3.3.1, we get the following bound on the first phase solution.

Lemma 3.3.4 *Let F_1 be the set of edges chosen in the first phase and let OPT denote the value of an optimum solution to (IP). Then*

$$\sum_{e\in F_1} c_e \le (1-\frac{1}{|A|})OPT$$

The second phase:

In the next section, we describe an algorithm for augmenting the forest F_1 with a set of edges F_2 disjoint from F_1. The algorithm also simultaneously constructs an implicit dual solution $\tilde{y}_S$ for $S \subset V$. We shall prove the following theorem in the next two sections.

Theorem 3.3.5 *The second phase algorithm chooses a set F_2 of edges disjoint from F_1, and an implicit dual solution $\tilde{y}$.*

(1) The set of edges in $F_1 \cup F_2$ two-connects each of the trees in (V, F_1).

(2) The solution $\tilde{y}$ will have the property that $\tilde{y}_S > 0$ only when $|\Gamma(S) \cap F_1| = 1$. Also, the solution $\tilde{y}$ will obey the normal packing constraint on all edges not in F_1. Namely, for any edge $e \in E - F_1$, we have $\sum_{S:\Gamma(S)\ni e} \tilde{y}_S \le c_e$.

We also prove the following relation, which is the basis for our performance analysis of the algorithm.

Theorem 3.3.6 *Let F_2 be the set of edges chosen by the second phase algorithm and let $\tilde{y}$ be the dual solution it implicitly constructs. Then*

$$\sum_{e\in F_2} c_e \le (2 - \frac{2}{|A|}) \sum_{S\subset V} \tilde{y}_S$$

The strategy to prove a bound of two for the second phase is to show that $\tilde{y}$ is feasible to the dual (LP) above. For this, we need the following lemma.

Lemma 3.3.7 *If $|\Gamma(S) \cap F_1| = 1$ then $f(S) = 1$.*

Proof: Suppose $\Gamma(S) \cap F_1 = \{e\}$ and suppose $e \in N$, where $N, N_1, \ldots, N_m$ are components of (V, F_1). Let P and Q be the two components of $N - \{e\}$. Assume without loss of generality that $P \subseteq S$ and $Q \cap S = \emptyset$. Since $\Gamma(S)$ only intersects the edge e from F_1, for each N_i, either $S \cap N_i = \emptyset$ or $S \cap N_i = N_i$. Thus, we can write S as the disjoint union of P and some of the components in F_1. Now $f(N_i) = 0$ for all the N_i's. Also, from Lemma 3.3.2, we have $f(P) = 1$. Disjointness and Observation 3.3.3 together imply that $f(S) = 1$. □

Now, we show that $\tilde{y}$ is a feasible solution to (LP). Namely, for each edge $e \in F_1$, set $r_e = \sum_{S:\tilde{y}_S>0,\ \Gamma(S)\cap F_1=\{e\}} \tilde{y}_S$. Set $r_e = 0$ for all the other edges. Since we have $\tilde{y}_S > 0$ only when $|\Gamma(S) \cap F_1| = 1$, we get

$$\sum_{e\in E} r_e = \sum_{e\in F_1} r_e = \sum_{e\in F_1} \sum_{S:\Gamma(S)\cap F_1=\{e\}} \tilde{y}_S = \sum_{S:|\Gamma(S)\cap F_1|=1} \tilde{y}_S = \sum_{S\subset V} \tilde{y}_S \tag{3.1}$$

Now, we show that the solution $\tilde{y}_S$ obeys all the packing constraints in (LP). For any edge $e \in F_1$, we have

$$\sum_{S:\Gamma(S)\ni e} \tilde{y}_S = r_e \le r_e + c_e$$

since $c_e \ge 0$. Also, for any edge e not in F_1, we have

$$\sum_{S:\Gamma(S)\ni e} \tilde{y}_S \le c_e \le c_e + r_e$$

using the second part of Theorem 3.3.5. Finally, the value of the solution is

$$\sum_S 2f(S)\tilde{y}_S - \sum_e r_e \ge 2 \sum_{S:|\Gamma(S)\cap F_1|=1} \tilde{y}_S - \sum_{e\in F_1} r_e = \sum_S \tilde{y}_S$$

The first inequality follows from Lemma 3.3.7 and the second equality from Equation (3.1). By weak linear programming duality, we again have $OPT \geq \sum_S \tilde{y}_S$. Combining this and Theorem 3.3.6, we get the following lemma.

Lemma 3.3.8 *Let F_2 be the set of edges chosen by the second phase algorithm and let OPT denote the value of an optimum solution to (IP). Then*

$$\sum_{e \in F_2} c_e \leq (2 - \frac{2}{|A|})OPT$$

We pause and prove that the solution $F_1 \cup F_2$ we offer is feasible.

Lemma 3.3.9 *$F_1 \cup F_2$ is feasible for (IP).*

Proof: For any $S \subseteq V$, we have $|\Gamma(S) \cap F_1| \geq f(S)$ since F_1 is feasible for (IP1). By Theorem 3.3.5, the set of edges in $F_1 \cup F_2$ two-connects all the trees in F_1. Thus if $f(S) = 1$ and $|\Gamma(S) \cap F_1| = 1$ then we must have $|\Gamma(S) \cap F_2| \geq 1$ for such S. Therefore if x is the incidence vector of $F_1 \cup F_2$ then for any S with $f(S) = 1$, we have

$$x(\Gamma(S)) = |\Gamma(S) \cap F_1| + |\Gamma(S) \cap F_2| \geq 2$$

Hence, x is feasible for (IP). □

We have described construction of a feasible solution $F_1 \cup F_2$ to (IP) and we have bounds on its value from Lemmas 3.3.4 and 3.3.8. These give us the Theorem 3.1.1. It remains to describe the algorithm to construct F_2 and the implicit dual $\tilde{y}_S$. We do this in the next section.

3.4 The Algorithm

In this section, we describe the algorithm for augmenting a given forest F_1 such that each tree in F_1 becomes two-connected. Note that F_1 is a minimal solution to the integer program (IP1). We also implicitly construct a dual solution $\tilde{y}$ such that it has the properties we required in Theorem 3.3.5. Since we require that $\tilde{y}_S > 0$ only when $|\Gamma(S) \cap F_1| = 1$, we define this condition as a new function f', namely,

$$f'(S) = \begin{cases} 1 & \text{if } |\Gamma(S) \cap F_1| = 1 \\ 0 & \text{otherwise} \end{cases}$$

It is important to note that the function f' defined this way is *not* proper. It can be easily shown to violate the disjointness requirement. Our basic algorithm for the second phase is similar to that of Goemans and Williamson but is modified to take care of this difference. As input to the algorithm, we are given an undirected graph G with edge costs $c_e \geq 0$ and the forest F_1. We output a set F_2 of edges such that $F_2 \cap F_1 = \emptyset$ and each tree in F_1 is two-connected in $F_1 \cup F_2$.

Overview

The algorithm maintains a forest, F, of edges as candidates for F_2. The forest F initially has no edges. It also maintains a set of *clusters* which partitions the vertex set at any time in the running of the algorithm. To begin with, each node in V is in its own cluster. The set of clusters at any time in the running of the algorithm can be partitioned into *active* and *dead* (or inactive) clusters, namely, those with f' value 1 and 0 respectively. The algorithm progresses in iterations, choosing edges between clusters and merging clusters. Each merge involves an active cluster. When no active cluster remains, the algorithm terminates. Whenever a chosen edge closes a cycle in $F_1 \cup F$, this cycle is collapsed: all the clusters containing nodes in the cycle are unioned into a single cluster. Note that the graph induced by $F_1 \cup F$ at the beginning of each iteration with the clusters contracted to single nodes is acyclic, because cycles are collapsed at each iteration. The algorithm also implicitly maintains a dual solution $\tilde{y}$. Only the active clusters are used to update the dual solution. As long as there remains an active cluster, there remain at least two clusters since $f'(V) = 0$. In the end, we do a clean-up to retain only essential edges and remove redundant edges.

The algorithm

We use the notation of [60]. We shall use $\mathcal{C}$ to denote the set of clusters that the algorithm maintains. For a vertex v, we let C_v denote the cluster containing v. We shall use a counter t which records the iteration number during the course of the algorithm. Henceforth, we shall use the term "at time t" to refer to iteration number t of the algorithm.

```
Initialize $F := \emptyset$, $\mathcal{C} := \{\{v\} : v \in V\}$, $t := 0$.
Comment: Implicitly set $\tilde{y}_S := 0$ for all $S \subseteq V$.
For each $v \in V$, set $d(v) := 0$.
While there is a $C \in \mathcal{C}$ such that $f'(C) = 1$
    Find edge $e = (i,j)$ with $i \in C_1 \in \mathcal{C}$, $j \in C_2 \in \mathcal{C}$, $C_1 \neq C_2$ that minimizes the reduced cost $\delta = \frac{c_e - d(i) - d(j)}{f'(C_1) + f'(C_2)}$.
    For all $v \in C \in \mathcal{C}$, set $d(v) := d(v) + \delta \cdot f'(C)$. Comment: Implicitly set $\tilde{y}_C := \tilde{y}_C + \delta \cdot f'(C)$.
    Set $t := t + 1$ and $F := F \cup \{e\}$. Comment: Edge $e$ is chosen at time $t$.
    If $e$ closes a cycle in $F_1 \cup F$, then union all the clusters containing nodes in this cycle to form a new cluster and insert this in $\mathcal{C}$. Delete each of the old clusters used in the unioning from $\mathcal{C}$. Comment: We say that a collapse has occurred at time $t$. All the edges in this cycle with their endpoints in different clusters are said to participate in this collapse.
```

```
9     If $e$ does not close a cycle, union the two clusters $C_1$ and $C_2$ to form a new cluster and
        insert this in $\mathcal{C}$. Delete the two clusters $C_1$ and $C_2$ from $\mathcal{C}$.
10  (Clean-up step) Select a subset $F_2$ of $F$ to output. Comment: This step is described later
      in this section.
```

Clean-Up

Before we describe the clean-up step, we need some notation. At any time t, let $\mathcal{C}(t)$ denote the set of clusters in $\mathcal{C}$ just *after* iteration number t of the algorithm. Let e_t denote the unique edge added in the t^{th} iteration. We define $F(t)$ to be set of edges in the set F added until time t including e_t. Let t_f be the total number of iterations before the algorithm terminates.

We shall refer to the graph (V, F_1) or any contracted version of it as the *skeleton*. Hence the edges in F_1 are termed *skeletal*. Also, a path made up of edges entirely in F_1 is called a *skeletal path*. Similarly, edges in F are termed *nonskeletal*.

Clean-up works by processing the edges in F in the *reverse* order in which they were chosen by the algorithm and discarding those that are not essential for two-connectivity. The procedure below takes as input the set of edges F chosen by the algorithm and outputs a subset $F_2 \subseteq F$ of edges to be retained in the final solution.

```
1   Initialize $F_2 := \emptyset$.
2   For $t := t_f$ down to 1 do
3     If the graph $(V, F_1 \cup F_2 \cup F(t) - \{e_t\})$ contains at least one skeletal bridge edge $b_t$, then
        set $F_2 := F_2 \cup \{e_t\}$.
```

Feasibility

We have the following observations from the algorithm.

Proposition 3.4.1 *At any iteration t of the algorithm, the set of nodes in each cluster in $\mathcal{C}(t)$ is one-connected using edges in $F_1 \cup F(t)$.*

The above proposition can be proved by induction on the iteration count t.

Definition: Define a skeletal edge e to be *enclosed* at time t if both its endpoints belong to the same cluster in $\mathcal{C}(t)$.

Observation 3.4.2 *A skeletal edge is enclosed at time t if and only if it is part of an edge-simple cycle in $(V, F_1 \cup F(t))$.*

Proof: First we prove the *if* part. In step 8, we collapse any cycle formed by a newly chosen edge, and we merge the clusters involved. It follows by induction on the iteration count t that all the nodes in any edge-simple cycle in $(V, F_1 \cup F(t))$ are in the same cluster in $\mathcal{C}(t)$. If a skeletal edge is part of such a cycle, both its endpoints are in the same cluster, and hence the edge is enclosed.

Now, we prove the *only-if* part. Suppose a skeletal edge s is enclosed for the first time during the course of the algorithm at time t'. Then a collapse must have occurred at time t' and the edge s participated in the collapse. We now identify an edge-simple cycle in $(V, F_1 \cup F(t'))$ containing s. Consider the cycle that collapsed at time t'. Each cluster in the collapsing cycle is one-connected using edges in $F_1 \cup F(t'-1)$ by Proposition 3.4.1. So we can form a cycle containing s using the edges in this collapsing cycle, along with the paths within each cluster in this cycle in the graph $(V, F_1 \cup F(t'))$ $\Box$

Note that a skeletal edge, once enclosed, stays enclosed for the rest of the algorithm.

Observation 3.4.3 *Let N be any connected component in (V, F_1). Then the set of nodes in N that occur in the same cluster in $\mathcal{C}(t)$ are two-connected in $(V, F_1 \cup F(t))$.*

Proof: Using induction on the iteration count t, we can show that the set of edges in N with both endpoints in the same cluster in $\mathcal{C}(t)$ form a subtree of N. Each edge in this subtree is thus enclosed at time t by definition and hence its endpoints are two-connected by Observation 3.4.2. Since two-connectivity is a transitive relation on the nodes, all the nodes in this subtree are two-connected. $\Box$

Definition: For any iteration number t of the algorithm, we define $F_1(t)$ and $F_2(t)$ to be the subset of edges in F_1 and F_2 respectively with their endpoints in different clusters in $\mathcal{C}(t)$. Note that $F_2(t)$ is just the set of edges in F_2 chosen after iteration t.

From the running of the main algorithm, we have the following lemma.

Observation 3.4.4 *Each connected component of F_1 is completely contained in some cluster in $\mathcal{C}(t_f)$.*

Proof: Consider the graph $(\mathcal{C}(t_f), F_1(t_f))$ that F_1 induces on the set of clusters at the time of termination of the algorithm. We claim that $F_1(t_f)$ is empty. Otherwise, consider a nonsingleton connected component H of $(\mathcal{C}(t_f), F_1(t_f))$. Each cluster in H has degree at least one. By definition of the activity function f', since each cluster is inactive at time t_f, each cluster in H has degree at least two. Hence H contains a cycle. But the algorithm collapses any such cycle into a single cluster and thus this is impossible. $\Box$

The observation above implies that every skeletal edge is enclosed at time t_f. Using Observation 3.4.2, this implies that each tree in the initial skeletal forest (V, F_1) is two-connected in the graph $(V, F_1 \cup F)$. By the definition of clean-up, we can prove the following proposition by backwards induction on the iteration count t.

Proposition 3.4.5 *The graph* $(V, F_1 \cup F_2(t) \cup F(t))$ *does not contain any skeletal bridge edge.*

Since $F_2(0) = F_2$, the above proposition implies that there is no skeletal bridge edge in the final retained solution $(V, F_1 \cup F_2)$. Thus we have the following lemma.

Lemma 3.4.6 *Each component* N *of* F_1 *is two-connected in the graph* $(V, F_1 \cup F_2)$.

This proves the first part of Theorem 3.3.5.

Definition: Note that the edges in $F_1(t)$ and $F_2(t)$ defined earlier have their endpoints in different clusters in $\mathcal{C}(t)$. So we can define an auxiliary graph $G(t)$ on the node set $\mathcal{C}(t)$ by $G(t) = (\mathcal{C}(t), F_1(t) \cup F_2(t))$. For each edge $g = (a, b)$ in $F_1(t) \cup F_2(t)$, we have an edge (A, B) in the graph $G(t)$ such that $A \ni a$ and $B \ni b$. The graph $G(t)$ may be thought of as a contracted version of the solution subgraph $(V, F_1 \cup F_2)$ at time t in which all the nodes in the same cluster in $\mathcal{C}(t)$ have been contracted into a single node and only edges between clusters are retained.

Observation 3.4.7 *The graph* $(\mathcal{C}(t), F_2(t))$ *is acyclic.*

Proof: Connectivity between clusters in $\mathcal{C}(t)$ using edges of $F_2(t)$ implies that these clusters are eventually merged into a single cluster in the course of the algorithm. Since no edges are ever chosen between nodes in the same cluster, the graph $(\mathcal{C}(t), F_2(t))$ is acyclic. □

A property of clean-up

As a result of the clean-up, we have the following lemma.

Lemma 3.4.8 *Suppose the edge* $e_t \in F_2$ *is retained in* F_2 *because it left a skeletal edge* b_t *as a bridge edge in step 3 of the clean-up procedure. Then* b_t *is also a bridge edge in the graph* $G(t') - \{e_t\}$ *for any* $t' < t$.

Proof: By the definition of clean-up, the edge e_t is retained in F_2 because it left the skeletal edge b_t as a bridge edge in the graph $(V, F_1 \cup F_2(t) \cup F(t) - \{e_t\})$. Hence, b_t is also a bridge edge in the subgraph $(V, F_1 \cup F(t) - \{e_t\})$. From Observation 3.4.2, this implies that it is not enclosed at any time $t' < t$. Thus $b_t \in F_1(t')$.

Note that we can obtain the graph $G(t') - \{e_t\}$ from $(V, F_1 \cup F_2(t) \cup F(t) - \{e_t\})$ by a series of edge-contractions and edge-deletions not involving b_t. Each cluster in $\mathcal{C}(t')$ is one-connected using edges in $F_1 \cup F(t)$ by Proposition 3.4.1; we can therefore obtain a node in $\mathcal{C}(t')$ by contracting along the edges between nodes in V in this cluster. It remains to show that we can obtain $G(t') - \{e_t\}$ from the contracted graph by deleting edges. Now by definition, $F_1(t') \subseteq F_1$. Also, since $t' < t$, the set of edges in F_2 chosen after time t' is a subset of all the edges chosen until time t and the set of edges in F_2 chosen after time t, i.e., $F_2(t') \subseteq F(t) \cup F_2(t)$. We conclude that the edges in $G(t')$ are $F_1(t') \cup F_2(t')$, a subset of $F_1 \cup F_2(t) \cup F(t)$, so we can obtain $G(t') - \{e_t\}$ from the contracted graph by deleting edges. Note that since $b_t \in F_1(t')$, we do not use it in any of these operations. Furthermore, neither of these operations destroys bridge edges and so b_t remains a bridge in the graph $G(t') - \{e_t\}$. □

We shall use the above lemma in the proof of the performance guarantee.

3.5 The Performance Guarantee

Our analysis of the algorithm is modeled on that of [60]. In this section, we prove Theorem 3.3.6. From the construction of the $\tilde{y}$ values and the choice of edges in F, we get the following lemma directly.

Lemma 3.5.1 *The solution $\tilde{y}$ satisfies $\tilde{y}_S > 0$ only when $|\Gamma(S) \cap F_1| = 1$. Furthermore, it obeys the normal packing constraints on edges not in F_1. Namely, for any edge $e \in E - F_1$, we have $\sum_{S:\Gamma(S)\ni e} \tilde{y}_S \leq c_e$. Also, for any edge in F, this constraint is tight, i.e., for $e \in F$, $c_e = \sum_{S:\Gamma(S)\ni e} \tilde{y}_S$.*

The above lemma proves the second part of Theorem 3.3.5.

We restate Theorem 3.3.6 and in the remainder of this section, prove it.

Theorem 3.3.6 Let F_2 be the set of edges chosen by the second phase algorithm and let $\tilde{y}$ be the dual solution it implicitly constructs. Then

$$\sum_{e \in F_2} c_e \leq (2 - \frac{2}{|A|}) \sum_{S \subset V} \tilde{y}_S$$

At any time t in the running of the algorithm, recall that the set of nodes in $\mathcal{C}(t)$ can be partitioned into active nodes and dead nodes, representing clusters with f' value 1 and 0 respectively. Let $k(t)$ denote the number of active nodes in $\mathcal{C}(t)$. We prove the following theorem in the remainder of the section and use it in the induction step of our proof of Theorem 3.3.6.

Theorem 3.5.2 *For each iteration number t, the sum of the degrees of the active nodes in the graph $(\mathcal{C}(t), F_2(t))$ is at most $2(k(t) - 1)$.*

The proof of Theorem 3.3.6 is adapted from a proof by Goemans and Williamson [60]. They use a lemma similar to our Theorem 3.5.2 in order to prove Theorem 3.3.6 by induction on the iteration count t. However, our proof of Theorem 3.5.2 differs completely from the proof of its counterpart in [60]. We now show that the above theorem suffices to prove Theorem 3.3.6.

Proof of Theorem 3.3.6: The proof is by induction on the iteration count t. From Lemma 3.5.1, we can write $\sum_{e \in F_2} c_e = \sum_{e \in F_2} \sum_{S:\Gamma(S) \ni e} \tilde{y}_S$. By changing the order of summation we can rewrite the above double sum as follows: $\sum_{e \in F_2} c_e = \sum_S \tilde{y}_S \cdot |\{e \in \Gamma(S) : \ e \in F_2\}|$. Let $\tilde{y}_S(t)$ denote the value of $\tilde{y}_S$ just before the t^{th} iteration of the algorithm. We show by induction on the iteration count t that

$$\sum_S \tilde{y}_S(t) \cdot |\{e \in \Gamma(S) : \ e \in F_2\}| \leq (2 - \frac{2}{|A|}) \sum_{S \subset V} \tilde{y}_S(t)$$

The inequality clearly holds when $t = 0$. Let δ denote the reduced cost of the edge e_t chosen at time t. At this iteration of the algorithm, the increase in the left-hand side of the above inequality is

$$\sum_{C \in \mathcal{C}(t): f'(C)=1} \delta \cdot |\{e \in \Gamma(C) : \ e \in F_2\}|$$

For the induction step, we must prove that this increase is bounded by the increase in the right-hand side, namely by

$$(2 - \frac{2}{|A|}) \cdot \delta \cdot |C \in \mathcal{C}(t) : f'(C) = 1|$$

By definition, $k(t) = |C \in \mathcal{C}(t) : f'(C) = 1|$. In terms of the graph $(\mathcal{C}(t), F_2(t))$, we must prove that the sum of the degrees of the active nodes is at most $2(k(t) - \frac{k(t)}{|A|})$.

Each leaf in the initial forest F_1 is a terminal and the number of active nodes in $\mathcal{C}(t)$ at any time t is at most the number of leaves in F_1. Hence $k(t) \leq |A|$ at any time t. Hence it is sufficient to prove that the sum of the degrees of the active nodes in the graph $(\mathcal{C}(t), F_2(t))$ is at most $2(k(t) - 1)$. But this is exactly what Theorem 3.5.2 states. Thus this theorem proves the induction step. □

We shall prove Theorem 3.5.2 in the remainder of this section.

Proof of Theorem 3.5.2: The proof works by constructing a subgraph of $G(t)$ that is a forest spanning all the nodes in $G(t)$. We call this the *proof forest*; it is made up of *proof trees*. This forest will include all the edges in $F_2(t)$. It also has the property that that every edge in $F_2(t)$ is on a path between two active nodes in the forest. The following proposition asserts that we can construct such a proof forest.

Proposition 3.5.3 *For each iteration number t, we can construct a spanning forest of $G(t)$ that includes all the edges in $F_2(t)$ such that each such edge is on a path between two active nodes in the forest.*

Then, we *prune* the forest so as to delete each edge that does not lie on a path between two active nodes. Notice that since every edge in $F_2(t)$ is in a path between two active nodes, no edge in $F_2(t)$ gets pruned. Moreover, as a result of pruning, each leaf node in the forest will be an active node. Thus any inactive node in the forest is an internal node and has degree at least two.

Since the pruned proof forest contains every edge in $F_2(t)$, $(\mathcal{C}(t), F_2(t))$ is a subgraph of this forest. Hence the sum of the degrees of the active nodes in $(\mathcal{C}(t), F_2(t))$ is at most the sum of the degrees of the active nodes in the pruned proof forest. Let N_i denote the number of inactive nodes in the pruned proof forest. The sum of the degrees of all the nodes in the pruned forest is at most $2(k(t) + N_i - 1)$. The sum of the active node degrees is the sum of the degrees of all the nodes minus the sum of the degrees of the inactive nodes. Since each inactive node has degree at least two, the sum of the active node degrees is at most $2(k(t) - 1)$. This proves Theorem 3.5.2. □

Construction of the proof forest

It remains to prove Proposition 3.5.3. For this, we need a few definitions.

Definition: Define a node in $\mathcal{C}(t)$ to be a *skeletal node* if it has at least one edge of $F_1(t)$ incident on it. Thus the skeletal nodes are exactly the nodes that are not isolated nodes in the graph $(\mathcal{C}(t), F_1(t))$.

We say that an edge b is a *bridge edge for edge a* if b is a bridge edge in the graph $G(t) - \{a\}$.

Proof of Proposition 3.5.3: We prove Proposition 3.5.3 by proving the following lemma that asserts that we can construct the proof forest inductively. In the following lemma, t denotes the iteration count such that we are attempting to construct a proof forest for $G(t)$. For any $t' \geq t$, the set $F_2(t) - F_2(t')$ represent the set of edges in the solution subgraph that have been chosen after time t up to and including time t'.

Lemma 3.5.4 *Let t be an iteration number of the algorithm. Then, for every $t' \geq t$, there exists a proof forest of $(\mathcal{C}(t), F_1(t) \cup (F_2(t) - F_2(t')))$ such that the following properties hold.*

1) *All the nodes in a connected component of $(\mathcal{C}(t), F_1(t))$ are in the same tree of the proof forest.*

2) *The proof forest contains all the edges in $F_2(t) - F_2(t')$.*

3) *Each edge in a nonskeletal path between two skeletal nodes in the forest is on a proof tree path between two active nodes.*

4) *Each skeletal edge $b \in F_1(t)$ that is not in the proof forest is a bridge edge for some nonskeletal edge in $G(t)$.*

We prove Proposition 3.5.3 by using Lemma 3.5.4 with $t' = t_f$. By the second property in this lemma, we get that the proof forest includes every edge in $F_2(t)$. By Lemma 3.4.8, each edge e' in $F_2(t)$ is chosen at some time $t' > t$ and is part of a cycle in the graph $G(t) = (\mathcal{C}(t), F_1(t) \cup F_2(t))$. Furthermore, this cycle contains at least one skeletal edge since $(\mathcal{C}(t), F_2(t))$ is acyclic by

Observation 3.4.7. Since the proof forest contains $F_2(t)$, this edge forms part of a nonskeletal path between skeletal nodes in the proof forest. Then the third property in Lemma 3.5.4 implies that this edge is on a path between two active nodes. This completes the proof of Proposition 3.5.3. $\square$

Proof of Lemma 3.5.4: The proof is by induction on t'. Henceforth, we shall use the phrase *proof forest for* t' for any $t' \geq t$ to refer to the proof forest of the graph $(\mathcal{C}(t), F_1(t) \cup (F_2(t) - F_2(t')))$.

Basis: ($t' = t$) In this case, the proof forest for t' is $(\mathcal{C}(t), F_1(t))$. It is easy to verify that the properties in Lemma 3.5.4 are true for this forest.

Induction step: Let P be the proof forest for t'. We show how to obtain a proof forest $\widehat{P}$ for $t' + 1$. Let e be the edge chosen by the algorithm at $t' + 1$. If e does not appear in F_2, (i.e. it was omitted during the clean-up phase), then P itself is a proof forest for $t' + 1$. Therefore suppose the edge e does appear in F_2. There are two cases.

Case 1: The edge e does not form any new nonskeletal path between skeletal nodes. In this case, $P \cup \{e\}$ is acyclic, so we let the new proof forest $\widehat{P}$ be $P \cup \{e\}$. Since no edges are deleted, the first and fourth properties continue to hold. Since we added the edge e to $\widehat{P}$, the second property continues to hold. Also, since we form no new nonskeletal paths between skeletal nodes, the third property also continues to hold.

Case 2: The edge e forms a new nonskeletal path between skeletal nodes. There are two subcases.

Subcase 2a: The new nonskeletal path formed goes between two skeletal nodes in different proof trees in the forest P. Thus this path merges these two proof trees in the proof forest. In this case also, $P \cup \{e\}$ is acyclic, so we let the new proof forest $\widehat{P}$ be $P \cup \{e\}$. Since no edge is deleted, the first and fourth properties continue to hold. Since we added the edge e to $\widehat{P}$, the second property continues to hold. Also, in this case, we can inductively infer that each of the two merging trees contains an active node. Thus the edges in the newly formed nonskeletal path are on a path between two active nodes in the two merging trees. This proves the third property for the proof forest $\widehat{P}$.

Subcase 2b: The new nonskeletal path formed goes between two skeletal nodes in the same proof tree T in the forest P. In this case, addition of the edge e causes the formation of a cycle C in the proof tree T. We shall add the edge e to the proof forest P and identify a skeletal edge b in this cycle to delete so that the remaining graph is acyclic and the four properties are maintained. The first property will be maintained if we delete *any* edge in the cycle C since this still leaves all the nodes in the proof tree T connected. Since we add the edge e to form the new proof forest and do not delete any nonskeletal edge, the second property would also continue to hold. However, it is harder to find a skeletal edge to delete so as to maintain the third and fourth properties. We introduce some definition to talk about the skeletal edge we delete.

Definition: A node in the cycle C is called an *entry node* if it is reachable in $T - C$ from an active node (See Figure 3.2).

In the following claim, we state the properties of the skeletal edge b that we delete from C to form the new proof tree.

Claim 3.5.5 *There is a skeletal edge b in C such that*

1) b is on a skeletal path between two entry nodes in the cycle C.

2) b is a bridge edge for a nonskeletal edge in $G(t)$.

Assume that the above claim holds. To form the new proof forest $\widehat{P}$ from the old, we add the edge e and delete the edge b whose existence is stated by the claim above. From the above discussion, this maintains the first two properties for $\widehat{P}$. We prove that $\widehat{P}$ also satisfies the third and fourth properties.

The fourth property follows from the second part of the above claim. To show that the third property is maintained, we must show that each edge in any nonskeletal path between skeletal nodes is on a path between two active nodes. Since the edge b is on a skeletal path between two entry nodes, each nonskeletal edge in the cycle C is on the complementary path in C between these two entry nodes. Since each of these entry nodes can reach an active node in $T - C$ by definition, all nonskeletal edges in the cycle C lie on a path between two active nodes in the new proof tree. Since the third property is inductively true for the tree T, any nonskeletal edge not in C and on a nonskeletal path between skeletal nodes is on a proof tree path between two active nodes in T. It is easy to verify that deletion of the skeletal edge b in C maintains this property for such edges. Thus the third property is maintained in the new proof tree $\widehat{P}$ in this subcase.

This completes the proof of Lemma 3.5.4. □

It remains to prove Claim 3.5.5. We do so in the remainder of this section.

Proof of Claim 3.5.5

We need a few preliminaries.

Claim 3.5.6 *For any cycle C_1 in $G(t)$, for any nonskeletal edge e' in C_1,*

(1) No edge of $G(t) - C_1$ is a bridge edge for e'.

(2) C_1 must contain a skeletal bridge edge for e'.

Proof: First we prove (1). We have by Proposition 3.4.5 that $G(t)$ has no skeletal bridge edge. Hence a bridge edge in $G(t) - \{e'\}$ must separate the endpoints of e'. But $C_1 - \{e'\}$ is a path between the endpoints of e', so any such bridge edge must belong to $C_1 - \{e'\}$.

Next we prove (2). By Lemma 3.4.8, $G(t) - \{e'\}$ must contain a skeletal bridge edge. By (1), the bridge edge must occur in C_1. □

Recall that the cycle C is the unique cycle in $T \cup \{e\}$.

Definition: A path consisting of edges of the cycle C is called a *segment* of C. A segment is *nonskeletal (skeletal)* if it consists entirely of nonskeletal (skeletal) edges. For any two nodes x and y in the cycle C, note that there are two segments between x and y. Given an edge e' of C, we can distinguish between these two segments as follows: the segment containing e' is denoted $\langle_{e'} x, y\rangle$, and the segment not containing f is denoted ${}_{e'}\langle x, y\rangle$.

Short-cuts and properties

Definition: A pair (x, y) of nodes in C is called a *short-cut* if there is a path from x to y in $G(t)$ using no edge of C.

Note that a pair (x, x) is trivially a short-cut. If $x \neq y$, we say that (x, y) is a *proper* short-cut. We have the following properties from the definition of short-cuts.

Claim 3.5.7 *Let e' be a nonskeletal edge of the cycle C and let (x, y) be a short-cut. Then*

(A) ${}_{e'}\langle x, y\rangle$ cannot contain a skeletal bridge edge for e'.

(B) $\langle_{e'} x, y\rangle$ must contain a skeletal bridge edge for e'.

Proof: Let P be the path from x to y using no edges of C. By applying Claim 3.5.6 to the cycle $C_1 = C \cup P - {}_{e'}\langle x, y\rangle$, we infer from (1) that ${}_{e'}\langle x, y\rangle$ contains no skeletal bridge edge for e'. This proves (A). Since C must contain such a skeletal bridge edge, it must occur in $C - {}_{e'}\langle x, y\rangle$ which is $\langle_{e'} x, y\rangle$. This proves (B). □

Corollary 3.5.8 *For each proper short-cut (x, y), the two segments between x and y each must contain a skeletal edge.*

Proof: Let S be one of the segments between x and y. Suppose for a contradiction that S consists entirely of nonskeletal edges. Let e' be one of these edges. By part (B) of Claim 3.5.7, $\langle_{e'} x, y\rangle$, which is S, contains a skeletal bridge edge for e', contradicting the fact that S is nonskeletal. □

Transition nodes and properties

Recall that we are working with a proof tree T such that C is the unique cycle in $T \cup \{e\}$.

Definition: A *transition node* is a node of C with one incident skeletal and one incident nonskeletal edge in C. Thus every maximal nonskeletal segment is bounded by transition nodes.

Lemma 3.5.9 *Let x be a transition node of C. Then there is an active node a_x reachable from x by a skeletal path that avoids C.*

Proof: The proof is illustrated in Figure 3.3. If x is itself active, we are done, so assume x is inactive. It has at least one incident skeletal edge, the one in the cycle C. Because it is inactive, it must have at least one other incident skeletal edge e' outside the cycle C. Let a_x be a leaf reachable from x by a skeletal path P_s going through e'. We show that P_s does not intersect the cycle C.

Suppose for a contradiction that the skeletal path intersects C. Let $x_1 \neq x$ be the first point of intersection of the path from x to a_x. Now x and x_1 are connected via a skeletal path P_1. They are also connected via a path P_2 along the cycle using edges of the proof tree T. Since the proof tree T is acyclic, there is a skeletal edge b' in this skeletal path P_1 from x to x_1 that is deleted in forming the proof tree. Inductively, property (4) of Lemma 3.5.4 is true for T and so b' must be a bridge edge for some nonskeletal edge c in $G(t)$. Since $x_1 \neq x$, note that x_1 and x are also connected in $G(t)$ via the path $P_3 = C - P_2$. Therefore the nonskeletal edge c for which b' is a bridge edge must lie in the path P_1. But since the path P_1 is skeletal, this is a contradiction. $\Box$

Observation 3.5.10 *For each transition node x of the cycle C, there is a entry node x' in C such that (x, x') is a short-cut.*

Proof: By Lemma 3.5.9, there is an active node a_x reachable from x by a skeletal path P_s that avoids C. Inductively, property (1) of Lemma 3.5.4 holds for T, so there is a path P_t in T from a_x to x. Let x' be the first node of P_t that lies on the cycle C; then x' is an entry node. Following P_s from x to a_x, then continuing along P_t to x' yields a path from x to x' using no edge of C. So (x, x') is a short-cut. $\Box$

Definition: For every transition node x in the cycle C, we designate an entry node x' that can be found using the above observation as the *entry node corresponding to x*.

Forbidden segments and their properties

Definition: A segment S is said to be *forbidden* if the endpoints of S are entry nodes, and S consists of three subsegments $S_1S_2S_3$, where S_2 is a nonskeletal segment, the endpoints of S_1 form a short-cut, and the endpoints of S_3 form a short-cut (See Figure 3.4). For any edge f in C, if f is not in $S_1 \cup S_3$, we say that S is forbidden *for f*. This terminology is motivated by the following claim.

Let e' be any nonskeletal edge in the cycle C. By part (2) of Claim 3.5.6, there is a skeletal edge $b_{e'}$ in C that is a bridge edge for e'.

Claim 3.5.11 *Any skeletal edge $b_{e'}$ that is a bridge edge for a nonskeletal edge e' in C cannot occur in a segment that is forbidden for e'.*

Proof: Let $S = S_1S_2S_3$ be a segment forbidden for e'. Since S_2 is nonskeletal, the edge $b_{e'}$ cannot lie in this subsegment. Let the endpoints of S_1 be x and y. By definition, the edge f does not occur in S_1, so we have $S_1 = {}_{e'}\langle x, y\rangle$. Since (x, y) is a short-cut, by part (A) of Claim 3.5.7, ${}_{e'}\langle x, y\rangle$ cannot contain a skeletal bridge edge for e'. Hence S_1 cannot contain $b_{e'}$. Similarly, we show that S_3 cannot contain $b_{e'}$. Thus any skeletal edge $b_{e'}$ that is a bridge edge for e' cannot occur in S. $\Box$

From the definition of segments forbidden for an edge $f \in C$, we have the following observation.

Observation 3.5.12 *Let S be a forbidden segment. For any nonskeletal edge f' in $C - S$, the segment S is forbidden for f'.*

The following lemma is illustrated in Figure 3.5.

Lemma 3.5.13 *Let e' be any nonskeletal edge in C, and let P be the maximal nonskeletal segment containing e'. Let l and r be the endpoints of P, and let their corresponding entry nodes be l' and r'. Then the segment $\langle_{e'} l', r'\rangle$ is a forbidden segment for e'.*

Proof: The entry nodes l' and r' cannot be internal nodes in the path P, else the short-cuts (l, l') and (r, r') would violate Corollary 3.5.8. Hence ${}_{e'}\langle l', l\rangle$, P, and ${}_{e'}\langle r, r'\rangle$ are contiguous edge-disjoint segments, and $\langle_{e'} l', r'\rangle = {}_{e'}\langle l', l\rangle \cup P \cup_{e'} \langle r, r'\rangle$. Thus $S = \langle_{e'} l', r'\rangle$ has entry nodes as endpoints and can be written as $S_1S_2S_3$ where $S_1 = {}_{e'}\langle l', l\rangle$, $S_2 = P$, and $S_3 = {}_{e'}\langle r, r'\rangle$ and (l, l') and (r, r') are short-cuts. So S is a forbidden segment. Moreover, since $e' \notin S_1 \cup S_3$, S is forbidden for a. $\Box$

Finding the skeletal edge to delete

Now we prove Claim 3.5.5 by providing an iterative procedure to find a skeletal edge b with the properties stated in the Claim. Note that we do not need to implement this algorithm since it is purely for the purpose of proof.

The algorithm to find the skeletal edge b maintains a nonskeletal edge f in the cycle C and a set of segments of the cycle C that are forbidden for f. On each iteration of this procedure, it either finds a skeletal edge b as required in Claim 3.5.5, or finds a new nonskeletal edge f' and adds a new forbidden segment to the set W. This new forbidden segment contains at least one edge not contained in any previous forbidden segment, so the number of iterations of this procedure is bounded by the size of the cycle C.

Find-skeletal-edge(f,W)

```
1 Initialize the nonskeletal edge $f := e$. By Lemma 3.5.13, we can find a segment $\langle_e u', v'\rangle$ that is forbidden for $e$. Initialize $W := \{\langle_e u', v'\rangle\}$.
2 Repeat
3     By part (2) of Claim 3.5.6, since $f$ is a nonskeletal edge in $C$, there is a skeletal edge $b_f \in C$ that is a bridge edge for $f$. By Claim 3.5.11, this skeletal bridge edge $b_f$ cannot occur in any segment in $W$ for these are forbidden for $f$. So $b_f$ must occur in a maximal segment $S$ of $C$ that is not overlapped by any segment in $W$. Note that the endpoints of $S$ must be entry nodes.
4     If the segment $S$ is wholly skeletal, the bridge edge $b_f$ for $f$ is one of the edges in this segment. This skeletal edge $b_f$ has the properties required in Claim 3.5.5, so we are done in this case. We stop and output the edge $b_f$ as the required skeletal edge.
5     Otherwise, the segment $S$ contains at least one nonskeletal edge $f'$. In this case, observe that since $f'$ is not in any of the segments in $W$, by Observation 3.5.12 each segment in $W$ is forbidden for $f'$. Now we use Lemma 3.5.13 to find a segment $\langle_{f'} l', r'\rangle$ forbidden for $f'$ and add this segment to the set $W$. We then set $f := f'$ and continue.
      Notice that the set $W$ with the segment $\langle_{f'} l', r'\rangle$ added consists of a set of segments forbidden for the new nonskeletal edge $f'$. Also, since $f'$ was not in any of the forbidden segments in $W$ earlier, but is included in the segment $\langle_{f'} l', r'\rangle$ added to $W$, the number of edges in the union of all the segments in $W$ has increased by at least one.
```

Thus the above procedure eventually finds a skeletal edge b as required in Claim 3.5.5. This completes the proof of the performance guarantee of the algorithm.

3.6 Approximating Augmentation

In this section, we restate Theorem 3.1.2 and prove it.

Theorem 3.1.2 *Given a graph $G = (V, E)$ along with an initial subgraph G_0 with edge set E_0, the* minimum-cost augmentation problem *is to choose a minimum-cost set of edges in G whose addition to G_0 yields a graph that obeys the constraints of (IP). This problem can be approximated in $O(n^2 \log n)$ time within a factor of* $2(1 - \frac{1}{|A|})$ *if the incidence vector, x_0 of the initial subgraph G_0 obeys the one-connectivity constraints*

$$x_0(\Gamma(S)) \geq f(S) \quad \forall S \ : \ \emptyset \neq S \subset V$$

Otherwise the performance guarantee is $3(1 - \frac{1}{|A|})$.

Proof: First we prove the weaker bound. For this, we modify the cost function c on the edges of G by setting the cost of the edges in the initial subgraph G_0 to zero. Let c' be the modified cost function. Then we apply the algorithm in Theorem 3.1.1 to the graph G with cost function c'. The algorithm finds a set of edges F' whose incidence vector is feasible for (IP). Moreover, the cost of the set of edges F' is at most $3(1-\frac{1}{|A|})$ times the cost of an optimum solution to (IP) under the cost function c'. Consider the minimum-cost augmentation of G_0. Let the cost of the edges in this augmentation be Aug^*. This set of edges along with the zero-cost edges in E_0 is a feasible solution to (IP) of value Aug^* under c'. Thus the cost of the edges in F' is at most $3(1-\frac{1}{|A|})Aug^*$ under c'. Finally we observe that since F' is feasible for (IP), the edges in $F' - E_0$ form a valid augmentation of E_0. This augmentation has cost at most $3(1-\frac{1}{|A|})Aug^*$ under c. This proves the weaker bound.

We now turn to the case when the incidence vector of the edges in E_0 satisfies the one-connectivity constraints:

$$x(\Gamma(S)) \geq f(S) \quad \forall S \; : \; \emptyset \neq S \subset V$$

In this case, we choose an arbitrary spanning forest F of G_0. It is easy to verify that the incidence vector of F satisfies the one-connectivity constraints. We then define a *minimal* version F_1 of F by retaining only the edges in F that are critical. Formally, we define

$$F_1 \leftarrow \{e \in F \; : \; \text{for some connected component } N \text{ of } (V, F - \{e\}),\ f(N) = 1\}$$

It follows from the results in [60] that if F is feasible for the one-connectivity constraints, then so is F_1. We can then use F_1 as the skeletal forest and apply the second-phase algorithm in Section 3.4. However, as before, we first modify the cost function c on the edges of G by setting to zero the cost of the edges in E_0. We apply the algorithm in Section 3.4 to the graph G with the modified cost function c' and the input skeletal forest F_1. The algorithm chooses a set of edges F_2 disjoint from F_1. We first show that $F_2 - E_0$ provides a valid augmentation and then we show that it has low cost.

By Lemma 3.3.9, the subgraph $F_1 \cup F_2$ is feasible for (IP). Since F_1 is contained in E_0, the subgraph $F_2 - E_0$ is a valid augmentation of E_0. Now we show that $F_2 - E_0$ has low cost under c. It suffices to show that F_2 has low cost under c'. By Lemma 3.3.8, the cost of the edges in F_2 is at most $2(1-\frac{1}{|A|})$ times that of an optimum solution to (IP) under the cost function c'. The optimum augmentation of G_0 together with E_0 is a feasible solution to (IP). The cost of this solution under c' is just the cost of the augmentation. Hence the cost of the edges in F_2 is at most $2(1-\frac{1}{|A|})$ times the cost of the optimal augmentation. This completes the proof of the stronger bound in Theorem 3.1.2. □

3.7 Implementation Issues

We note that our algorithm can be implemented in time $O(n^2 \log n)$ where $n = |V|$ using the techniques in [60]. We address the implementation of the clean-up steps. The important observation here is the initial forest F_1 and the candidate set of edges F both form acyclic subgraphs. Hence the total number edges in them is $O(n)$. At each step of clean-up, we identify the two-connected components of a graph with $O(n)$ edges. The time taken is linear in the number of edges and hence the time to implement each step of clean-up is linear in n. Since there are only a linear number of clean-up steps, the total time for performing the clean-up is $O(n^2)$. The overall running time is thus dominated by the time for implementing the main loop and is $O(n^2 \log n)$.

3.8 A Tight Example

We note that the performance bounds of the algorithms we have presented in Theorems 3.1.1 and 3.1.2 are existentially tight. This can be seen by their performance on an example graph adapted from [46]. This graph is shown in Figure 3.6. The thick (solid and dashed) edges are of unit cost and the thin edges have cost ϵ close to zero. We use the proper function f defined by $f(S) = 1$ for every nonempty proper subset S of V, i.e, we seek a minimum-cost two-connected subgraph. The initial spanning tree constructed in the first phase of our algorithm may consist of all the thick solid lines but one and all the thin lines of cost $(k-1) + 3k\epsilon$. All but one of the thick dashed lines may be chosen as the second phase solution. The cost of this set is $(2k-1)$. Thus our algorithm outputs a solution of total cost $3k - 2 + 3k\epsilon$. A minimum-cost two-connected subgraph of this graph consists of alternating thick dashed lines with all the thin lines. This subgraph has cost $k + 3k\epsilon$. Thus this example shows that the performance bounds of our algorithms are existentially tight.

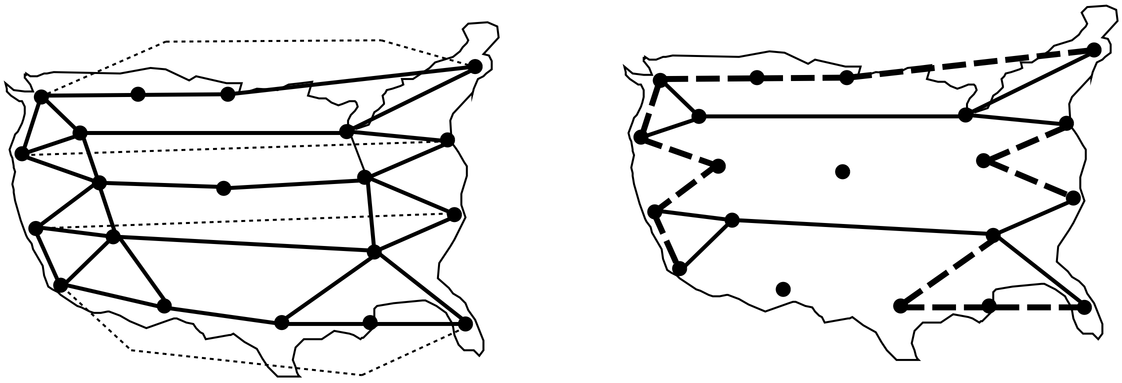


Figure 3.1: An instance of the unweighted generalized two-connected Steiner network problem and an optimum solution. Solid edges correspond to unit-cost links; dotted edges connect site pairs. In the figure on the right, the solid edges correspond to first phase one-connecting solution. The thick dashes form an augmentation to two-connect the solid trees.

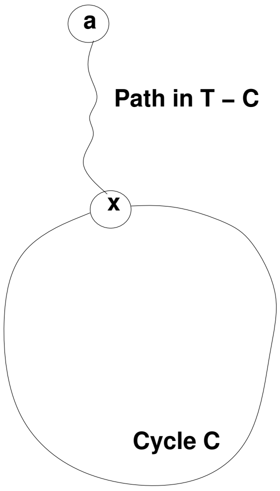


Figure 3.2: A node x in the cycle C is called *an entry node* if it is reachable in $T - C$ from an active node a.

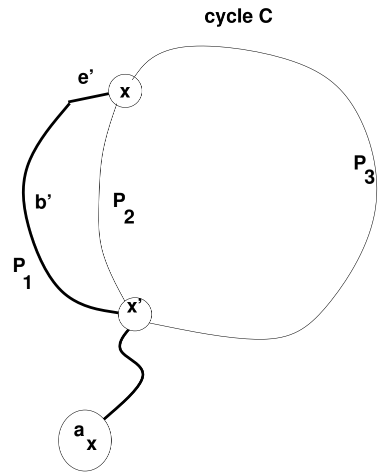


Figure 3.3: x is a transition node of C. e' is a skeletal edge incident to x outside C and a_x is a leaf reachable from x by a skeletal path P_s going through e'. $x_1 \neq x$ is the first node in the cycle C where the path P_s meets C. P_1, the subpath of P_s from x to x_1 is shown in dark.

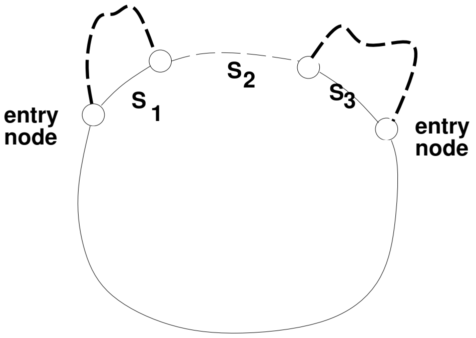


Figure 3.4: A segment S is said to be *forbidden* if the endpoints of S are entry nodes, and S consists of three subsegments $S_1S_2S_3$, where S_2 is a nonskeletal segment, the endpoints of S_1 form a short-cut, and the endpoints of S_3 form a short-cut. The dark dashed paths represent paths in $G(t) - C$ between the endpoints of S_1 and between the endpoints of S_3.

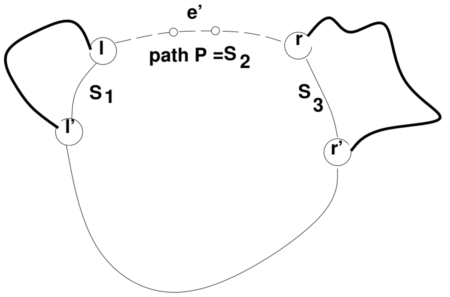


Figure 3.5: Lemma 3.5.13: e' is some nonskeletal edge in C, and P is the maximal nonskeletal path containing e' in C. Let l and r be the endpoints of P, and let their corresponding entry nodes be l' and r'. Then the segment $\langle_{e'} l', r'\rangle$ is a forbidden segment for e'. The dark lines show paths in $G(t) - C$ between each of l and r and the entry nodes l' and r' corresponding to them.

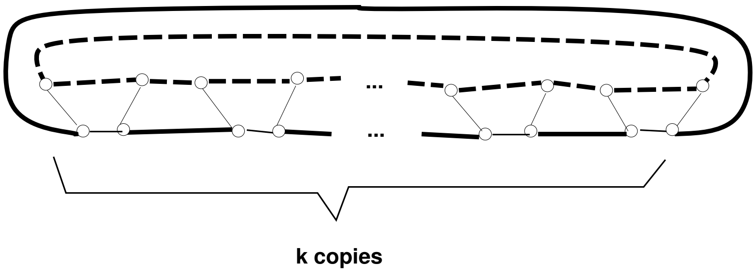


Figure 3.6: An example showing that the performance bounds of the algorithms in Theorems 3.1.1 and 3.1.2 are existentially tight.

Chapter 4

Node-weighted Network Design

We continue with our study of minimum-cost network-design problems and address an important generalization of the problem when the nodes of the network are assigned weights as well. This is a strict generalization of an edge-weighted problem since an edge-weighted problem can be reduced to a node weighted problem by subdividing each edge to introduce a new node with cost equal to that of the edge. In this chapter, we give the first approximation algorithm for the node-weighted Steiner tree problem. Its performance guarantee is within a constant factor of the best possible unless $\tilde{P} \supseteq NP$. [1] We generalize our algorithm to handle other node-weighted network-design problems. The results in this chapter were obtained jointly with Philip Klein, and appeared in [91].

4.1 Problem Definition

The Steiner tree problem in networks that we introduced in Section 1.1 is a classic hard problem in combinatorial optimization. Much research has been devoted to heuristics for its solution [33, 68, 112, 134, 135, 163]. Despite a slew of new approximation algorithms for this problem and some of its variants, no approximation algorithm has been given for perhaps the most natural variant: the *node-weighted Steiner tree problem*, in which costs can be assigned to nodes as well as edges. Indeed, Winter's survey [156] on the network Steiner problem closes with the sentence, "Further investigation of the vertex-weighted SPN [Steiner problem in networks] is needed."

One reason for the dearth of results on the node-weighted variant may be that it is harder than the standard problem. Indeed, while constant-factor approximations are known for the standard problem [96, 125, 148, 164] and even some of its generalizations [2, 60], the node-weighted version cannot be approximated to within less than a logarithmic factor unless $\tilde{P} \supseteq NP$ [14, 110].

In this chapter, we give the first approximation algorithm for the node-weighted Steiner tree problem. The performance guarantee is logarithmic. Thus assuming $\tilde{P} \not\supseteq NP$, the accuracy

---

[1] $\tilde{P}$ stands for the complexity class Deterministic Quasi-polynomial time, or DTIME[$n^{\text{polylog } n}$].

of our approximation is within a constant factor of the best-possible approximation achievable in polynomial time.

The algorithm we propose is only a slight variant of a heuristic proposed by Rayward-Smith and Clare in 1986 [74, 135, 134] for the standard edge-weighted Steiner tree problem. The key to our analysis is a decomposition lemma for trees; this lemma may be useful in other contexts as well.

We also show how to generalize the algorithm and its analysis to handle more general connectivity requirements modeled using the framework of proper functions introduced in Section 3.3. Thus we obtain approximation algorithms for node-weighted versions of, e.g, fixed and non-fixed point-to-point connection problems.

Preliminaries

Let G be a graph with nonnegative costs assigned to its nodes and edges. The cost of a subgraph of G is the sum of the costs of its nodes and edges. Let A be a subset of the nodes of G, called *terminals*. A *Steiner tree for A in G* is a connected subgraph of G containing all the nodes of A. (Note that an edge-minimal Steiner tree is indeed a tree.) The Steiner tree problem is to find a minimum-cost Steiner tree.

We introduce a problem that turns out to be closely related. Let B be a ground set, and let $S_1, \ldots, S_s$ be subsets of B with costs $c_1, \ldots, c_s$. A *set cover* is a collection of sets S_i whose union is B. The *set cover problem* is to find a minimum-cost set-cover.

Berman [14] showed that, in the presence of node-weights, approximating the minimum-cost Steiner tree is as hard as approximating set cover. More specifically, he showed that any instance of set-cover can be formulated as an instance of the node-weighted Steiner tree problem. The reduction is illustrated in Figure 4.1. Thus an approximation algorithm for minimum-cost Steiner tree could be used to achieve the same approximation for set-cover.

It has recently been proved [110] that no polynomial-time approximation algorithm for set-cover achieves an approximation factor smaller than $\frac{1}{4} \ln |B|$ (unless Deterministic Time $n^{\text{polylog} n}$ contains NP). By Berman's reduction, the same holds for the node-weighted Steiner tree problem. Thus we cannot expect to obtain an approximation algorithm that achieves a performance ratio better than logarithmic.

Theorem 4.1.1 *There is a polynomial-time algorithm to approximate the node-weighted Steiner tree problem in networks. The performance ratio is $2 \ln k$, where k is the number of terminals.*

An example due to Chvátal [25] can be adapted to show that our analysis of the algorithm is nearly tight (See Figure 4.2).

Our method can be easily applied to more general node-weighted network-design problems. For example, consider the following generalization of the Steiner problem: given a set of pairs of nodes (s_i, t_i), find a minimum-cost subgraph in which each s_i is connected to t_i. The edge-weighted version of this problem was addressed in [2]; the node-weighted version can be approximately solved using the method of this chapter.

We use a framework due to Goemans and Williamson [60] to formulate problems like that described above. Many network-design problems can be formulated as finding a subgraph of minimum-cost that covers a family of cuts in the graph (the particular family depends on the problem). For certain families of cuts, the edge-weighted problem can be approximated to within a factor of two [60]. We show in Section 4.5 that the node-weighted variants of these network-design problems can also be approximated; the performance is as in Theorem 4.1.1.

4.2 The Algorithm

In this section we describe the algorithm. Note that since any Steiner tree must include all the terminals, we can assume without loss of generality that the terminals have zero cost.

The algorithm maintains a node-disjoint set of trees containing all the terminals. Initially, each terminal is in a tree by itself.

The algorithm uses a greedy strategy to iteratively merge the trees into larger trees until there is only one tree. In each iteration, it selects a node v and a nonsingleton subset of the current trees so as to minimize the ratio

$$\frac{\text{cost of the node } v \text{ plus sum of distances to the trees}}{\text{number of trees}} \tag{4.1}$$

Here the distance along a path does not include the costs of its endpoints. Thus the choice minimizes the average node-to-tree distance. The algorithm uses the shortest paths between the node and the selected trees to merge the trees into one.

It is easy to implement an iteration. For each node v, define the ***quotient cost*** of v to be the minimum value of (4.1), taken over all nonsingleton subsets of the the current trees. To find the quotient cost of v, compute the distances d_i from v to each of the trees T_i; assume without loss of generality that the trees are numbered so that $d_1 \leq d_2 \leq \cdots \leq d_k$. In computing the quotient cost of v, it is sufficient to consider subsets of the form $\{T_1, T_2, \ldots, T_i\}$. Thus the quotient cost for a given vertex can be computed in polynomial time; by computing the quotient cost of all the vertices, we can determine the minimum quotient cost, and thus carry out an iteration in polynomial time.

In Figure 4.2, we adapt an example due to Chvátal [25] to show that the performance ratio in Theorem 4.1.1 is nearly tight.

In Section 4.4, we show that the cost of all nodes and edges selected by the algorithm is not much more than the minimum cost of a Steiner tree.

4.3 Spider Decompositions

The proof of the performance guarantee of the algorithm involves showing the existence of a node with low quotient cost relative to the minimum cost of a Steiner tree. To show this, we prove that a minimum Steiner tree can be decomposed into subtrees we call *spiders*. It follows that one of these spiders has low quotient cost relative to the cost of the minimum Steiner tree. It then follows that the cost of an iteration of the greedy algorithm is small.

Definition: A *spider* is a tree with at most one node of degree greater than two. A *center* of a spider is a node from which there are edge-disjoint paths to the leaves of the spider. Note that if a spider has at least three leaves, its center is unique. A *foot* of a spider is a leaf, or, if the spider has at least three leaves, the spider's center. Note that every spider contains disjoint paths from its center to all of its feet. A *nontrivial spider* is one with at least two leaves.

Definition: Let G be a graph, and let M be a subset of its nodes. A *spider decomposition* of M in G is a set of node-disjoint nontrivial spiders in G such that the union of the feet of the spiders in the decomposition contains M.

Theorem 4.3.1 *Let G be a connected graph, and let M be a subset of its nodes such that $|M| \geq 2$. Then G contains a spider decomposition of M.*

Proof: We provide a simple proof of the theorem due to L. Wolsey [160]. Let T be any rooted spanning tree of G. The depth of a node in the tree is defined as the distance of the node from the root.

We prove the theorem by induction on $|M|$. Choose a node v of maximum depth in the tree such that the subtree rooted at v contains at least two nodes in M. Ties are broken arbitrarily. By choice of v, all the paths from the nodes in M to the node v in the subtree of v are node-disjoint, and together form a nontrivial spider centered at v.

We now delete the subtree rooted at v from the tree. If no node in M remains in the tree, we are done. If the tree contains two or more nodes in M, then we can find a spider decomposition of these nodes by the induction hypothesis. Otherwise, there is exactly one node in M remaining in the tree. In this case, we add the path in the tree from this node in M to the spider centered at v. This leaves a spider centered at v and we are done. $\square$

4.4 The Performance Guarantee

Now we prove the performance guarantee for the greedy algorithm. Let ϕ_i denote the number of subtrees in the solution just after iteration i. Thus, for instance, ϕ_0 is the number of terminals in G. Let the number of trees merged at iteration i be h_i. Then we have

$$\phi_i = \phi_{i-1} - (h_i - 1) \tag{4.2}$$

Let C_i denote the cost of the subgraph added by the algorithm in iteration i. Let OPT denote the cost of a minimum-cost Steiner tree spanning the terminals. The key ingredient of our proof is the following lemma, which depends on our spider-decomposition theorem.

Lemma 4.4.1 *At any iteration i of the algorithm,*

$$h_i \geq \frac{C_i \cdot \phi_{i-1}}{OPT} \tag{4.3}$$

Let T^* be a minimum-cost Steiner tree. Let $T_1, \ldots, T_{\phi_{i-1}}$ be the current trees at the beginning of iteration i of the algorithm. Let T_i^* be the graph obtained from T^* by contracting each T_j to a supernode of zero cost. Note that T_i^* is connected and contains all supernodes. Let M be the set of supernodes. We apply Theorem 4.3.1 to the graph T_i^* to obtain a spider decomposition of M. Furthermore, the cost of all spiders in the decomposition is at most that of T_i^* which is in turn at most OPT.

Let $c_1, \ldots, c_r$ be the centers of the spiders in the decomposition. For a spider with only two leaves, i.e., a path, pick any node in the path as its center. Let $\ell_1, \ldots, \ell_r$ denote the number of nodes of M in each of these spiders respectively. Let the cost of the spider centered at c_j be $Cost_j$. Since every spider in the decomposition is nontrivial, each ℓ_j is at least two. Moreover, a spider with center c_j induces a subset of the current trees, namely the ℓ_j trees whose supernodes belong to this spider. The cost of c_j plus the sum of distances to these trees is exactly $Cost_j$. Hence the quotient cost of c_j is at most

$$\frac{Cost_j}{\ell_j}$$

Since the algorithm chooses a vertex of minimum quotient cost, for each spider in the decomposition we have

$$\frac{Cost_j}{\ell_j} \geq \frac{C_i}{h_i}$$

Summing over all the spiders in the cover yields

$$\sum_{j=1}^{r} Cost_j \geq \frac{C_i}{h_i} \sum_{j=1}^{r} \ell_j$$

By node-disjointness, the sum $\sum_{j=1}^{r} \ell_j$ of the number of nodes of M in each of the spiders is exactly the number ϕ_{i-1} of current trees. Also $\sum_{j=1}^{r} Cost_j$ is exactly the cost of the spider decomposition, which is at most OPT. Substituting in the above equation and simplifying yields

$$h_i \geq \frac{C_i \phi_{i-1}}{OPT}$$

□

We now use the above lemma in conjunction with an analysis technique due to Leighton and Rao [103] to complete the proof of the performance guarantee.

Substituting Equation (4.3) into (4.2) and using the inequality $h_i \leq 2(h_i - 1)$, we get

$$\phi_i \leq \phi_{i-1}(1 - \frac{C_i}{2 \cdot OPT}) \tag{4.4}$$

If the total number of iterations of the algorithm is p, then $\phi_p = 1$. Unraveling (4.4), we obtain

$$\phi_p \leq \phi_0 \prod_{j=1}^{p} (1 - \frac{C_j}{2 \cdot OPT})$$

Taking natural logarithms on both sides and simplifying using the approximation $\ln(1 + x) \leq x$, we obtain

$$2 \ln(\frac{\phi_0}{\phi_p}) \cdot OPT \geq \sum_{j=1}^{p} C_j$$

Since we assumed that all the terminals have zero cost, note that the cost of the final solution is the exactly the sum $\sum_{j=1}^{p} C_j$. Noting that $\phi_0 = k$ and $\phi_p = 1$ we finally have

$$\sum_{j=1}^{p} C_j \leq 2 \ln k \cdot OPT \tag{4.5}$$

This proves the performance guarantee in Theorem 4.1.1. □

4.5 General Node-weighted Network-design Problems

In this section, we generalize our algorithm to handle more general network-design problems. As discussed in Section 1.7, many such problems can be formulated as cut-covering problems: for a certain family of cuts in a graph, find a minimum-cost subgraph intersecting all the cuts in the family. We described the class of cut families that can be represented using proper functions f introduced by Goemans and Williamson in Section 3.3.

Given a proper function f, recall that the *terminals* are the nodes v of G such that $f(\{v\}) = 1$. As for the Steiner tree problem, every solution subgraph must contain all the terminals. Hence we can assume without loss of generality that the terminal nodes have zero cost.

Goemans and Williamson [60] gave a 2-approximation algorithm for edge-weighted cut-cover problems where the cut-family corresponds to a proper function f. In this chapter we give a complementary result for node-weighted problems.

Theorem 4.5.1 *Let G be a graph with node- and edge-weights. Let f be a proper function on the node-subsets of G. There is a polynomial-time approximation algorithm for finding a minimum-cost subgraph covering all cuts in the family defined by f. The performance guarantee is $2 \ln k$, where k is the number of terminals.*

Proof of Theorem 4.5.1: The algorithm in the theorem follows the outline of the algorithm in Section 4.2 very closely. As before, the algorithm maintains a set of node-disjoint trees. Initially each terminal is in a tree by itself; the algorithm merges them iteratively. An important difference is that only some of the trees are candidates for merging.

We designate a tree to be *active* if $f(\{\text{nodes in the tree}\}) = 1$. At each iteration, the algorithm proceeds as before but considers only active trees in computing the quotient costs of nodes. That is, the algorithm selects a node and a set of at least two active trees so as to minimize

$$\frac{\text{cost of the node plus sum of distances to the active trees}}{\text{number of active trees}} \tag{4.6}$$

Then the algorithm uses the paths from the node to the selected trees and merges them into a single tree.

As long as there is at least one active tree in the network, it follows by the symmetry and disjointness properties of f that there are at least two. The algorithm reduces the number of active trees in the network by at least one in each iteration, so it eventually terminates and outputs a set of inactive trees as the final solution. Thus each connected component of the solution is inactive. It follows [60] that the solution is in fact a cut-cover.

The proof of the performance guarantee proceeds as before, except that we define the value of the potential function ϕ_i to be the number of active trees in the current solution, rather than the number of trees. To prove the analogue of Lemma 4.4.1, we consider an optimal cut-cover and contract the current trees. This may result in a subgraph with possibly many inactive connected components. We then apply Theorem 4.3.1 to each of these connected components to obtain a spider decomposition of the contracted active trees. We use the result in the inequality

$$\phi_i \leq \phi_{i-1} - (h_i - 1) \tag{4.7}$$

where h_i is the number of active trees merged in the ith iteration. Note that with our new definition of the potential function ϕ_i we have inequality in (4.7) instead of equality as in (4.2) because the tree resulting from the merge may be inactive. The rest of the analysis is identical. This completes the proof of Theorem 4.5.1. □

4.6 Concluding Remarks

In this chapter, we have described approximation techniques for a variety of one-connected node-weighted network-design problems. In Chapter 6, we show how these techniques can been generalized to handle node-weighted network design problems with an additional constraint on the maximum degree of any node in the tree.

Two other natural generalizations of the node-weighted Steiner tree problem are at least as hard to approximate as the set cover problem, the directed Steiner problem and the group Steiner problem. In the directed Steiner problem, we are given a directed arc-weighted graph, a distinguished node, and a set of terminals. The goal is to find a minimum-cost arborescence rooted at the distinguished node and spanning the terminals. A transformation of the node-Steiner problem to the directed version was proposed by A. Segev [139].

The group Steiner problem, proposed by Reich and Widmayer [136], arises in VLSI design. In this problem, we are given an undirected edge-weighted graph and a collection of node-subsets, called *groups*. The goal is to find a minimum-cost connected subgraph containing at least one node from each group.

We now show how to reduce the set cover problem to a group Steiner problem: construct a graph with an auxiliary node and one node for each subset in the set cover problem. Each subset-node has an edge to the auxiliary node of cost equal to the cost of the subset. In formulating the group Steiner problem on this graph, define a group for each ground set element in the set cover problem: namely, the set of nodes corresponding to the subsets that contain the ground element. It is easy to check that a group Steiner tree in this graph corresponds to a set cover of the same cost and vice-versa.

We also observe that an approximation algorithm for the latter would yield one for the former. An instance of the group Steiner problem can be transformed to an instance of the directed Steiner problem as follows: replace each edge in the input graph with a pair of antiparallel arcs of the same cost. For each group $g = \{u_1, \ldots, u_t\}$ of vertices, introduce a new vertex v_g and add zero-cost arcs from each $u_i \in g$ to v_g. It is easy to verify that a outward-directed Steiner tree originating from a node in any group in this transformed graph can be converted to a group Steiner tree of the same edge-cost in the original graph and vice-versa.

This chapter concludes our study of minimum-cost network-design problems. We continue by studying a new notion of cost, namely, the maximum degree of any node in the network, in the next chapter.

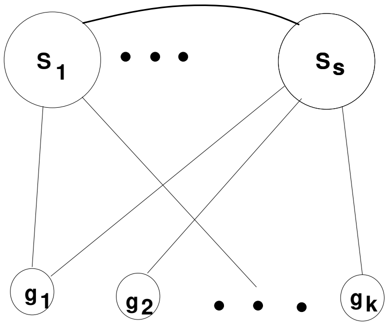


Figure 4.1: Let G be the bipartite graph with k nodes on one side of the bipartition, one for each ground element, and s nodes on the other side, one for each set S_i. There is an edge between a ground element node and a set node if the set contains the element. In addition, all the set nodes are pairwise adjacent. The cost of each set node is that of the set it represents. All other nodes and all edges have zero cost. It is easy to check that a Steiner tree for the k nodes corresponds to a set cover of the same cost, and vice versa.

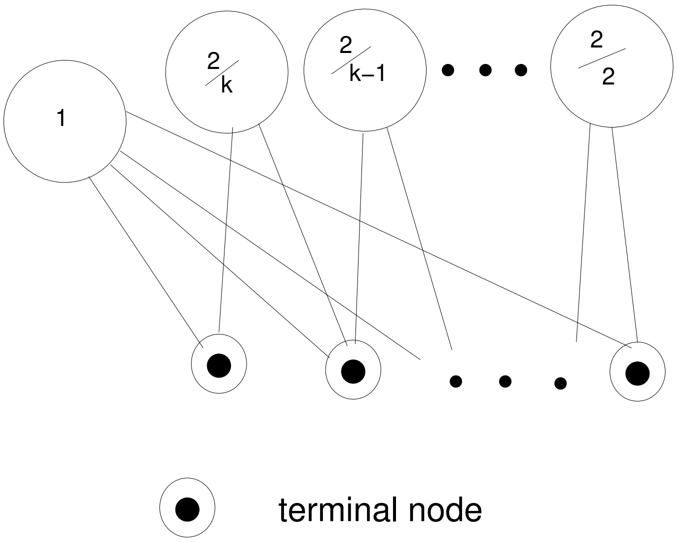


Figure 4.2: In the example above, the minimum-node-cost Steiner tree consists of the node of unit cost with edges to all the terminals. The greedy algorithm may choose each of the other nodes from left to right successively to produce a Steiner tree of cost $2(H(k)-1)$ where $H(k)$ denotes the k^{th} harmonic number.

Chapter 5

Minimum-degree Network-design

5.1 Motivation

In this chapter, we consider the problem of designing minimum-degree communication networks. The criterion of minimizing the maximum degree reflects decentralization of the communication network. The advantage of a network with low degree is that the failure of a node does not adversely affect the rest of the network too much. Keeping the maximum degree small is also essential in designing switching networks where the same switch is to be installed at each of the nodes. The switch must be designed to handle as many connections as the maximum degree of any node in the network in which it is to be used. Minimum-degree networks are also useful in building networks for broadcast where we wish to minimize the amount of work done at each site and also in designing power grids where the cost of a node increases with the degree of splitting the power [50]. The purpose of the network itself gives rise to certain connectivity requirements in the network.

In this chapter, we consider essentially two distinct types of such connectivity requirements on the solution subgraph. In the first type, we consider network design problems where the connectivity requirements can be modeled by a $\{0,1\}$-valued function f on all the cuts in the graph. In the second type, we consider the problem of designing a minimum-degree two-edge-connected spanning subgraph of a given graph. For problems of both types, we provide an approximation algorithms to output a network whose degree is guaranteed to be at most $1+\epsilon$ times optimal, plus an additive error of $O(\log_{1+\epsilon} n)$ for any $\epsilon > 0$. Both the algorithms run in quasipolynomial time $O(n^{\log_{1+\epsilon} n})$. As a simple consequence, we obtain polynomial-time approximation algorithms for both the problems with performance ratio $O(n^{\delta})$ for any $\delta > 0$ (by setting $\epsilon = n^{\delta}$). Other than having the same performance guarantees and running times, both the algorithms we describe use the technique of local optimization: perform local improvements to the solution that decreases the degrees of nodes with high degree in the solution.

As a consequence of our analysis, we show that the minimum degree in both the problems above is well-estimated by certain polynomially solvable linear programs. This fact suggests that the linear programs we describe could be useful in obtaining optimal solutions via branch-and-bound.

The performance factor achievable in polynomial time for the minimum-degree one-connected network problem using results in this chapter can be improved using our techniques in Chapter 6. We describe there a technique for finding networks with low degree as well as low cost. As a consequence of one of the results we derive there, we can obtain a performance guarantee for the minimum-degree one-connected network-design problem addressed in this chapter that is logarithmic in the number of nodes in the graph. This improvement is elaborated in Section 6.4. However, the result described for this problem in this chapter may be used to derive approximation factors even better than logarithmic at the expense of higher running times. The results in this chapter were obtained jointly with Balaji Raghavachari and Philip Klein, and appeared in [131].

In Section 5.2, we start by reviewing previously known results on special cases of the problem we consider in this chapter. In Section 5.3, we describe our results on general minimum-degree one-connected networks. In Section 5.4, we address the minimum-degree two-edge-connected spanning network-design problem.

5.2 Previous Work

The minimum-degree spanning tree problem: The problem is to find a spanning tree of the graph whose maximum degree is minimized. This problem is a generalization of the Hamiltonian Path problem and hence is clearly NP-hard [53]. The first result on approximating the minimum-degree spanning tree was that of Fürer and Raghavachari [49]. They gave a polynomial time approximation algorithm for minimum-degree spanning tree with performance guarantee $O(\log n)$. Their algorithm further generalizes to find rooted spanning trees in directed graphs of approximately minimum indegree. In subsequent work [50], they improved their previous results and provided another polynomial time algorithm to approximate the minimum-degree spanning tree to within one of the optimal. Clearly no better approximation algorithms are possible for this problem.

In distantly related work on this problem, Lawler [99] showed that matroid methods sufficed to solve the following variant of the minimum-degree spanning tree problem in polynomial time: given a graph G and an independent set I of nodes of G, find a spanning tree that minimizes the maximum degree of any node in I. Gavish [56] formulated the minimum-degree spanning tree problem as a mixed integer program and provided an exact solution using the method of Lagrangian multipliers.

The minimum-degree Steiner tree problem: This is an extension of the above problem, where given an input graph and a distinguished subset of the vertices, D, we seek to find a Steiner tree (spanning at least the set D) whose maximum degree is minimum. The first polynomial-time approximation algorithm was provided by Agrawal, Klein and Ravi [3]. The performance guarantee is a factor of $O(\log k)$, where k is the size of D. This was improved by Fürer and Raghavachari in [49] and they provide a polynomial-time approximation algorithm for the problem with the performance guarantee essentially the same as that we show here. In fact, our work is a generalization of this algorithm and reduces to their algorithm for this special case. Their analysis of this special case of our algorithm also shows that in this case, our algorithm is polynomial-time. Fürer and Raghavachari [51] later demonstrated a polynomial-time algorithm for the Steiner case that finds a tree whose degree is within one from optimal.

5.3 The Minimum-degree Constrained Forest Problem

We begin this section by formulating general one-connected network problems using the framework of proper function introduced in Section 3.3. We then introduce the notion of weakness, a generalization of the reciprocal of the toughness of a graph. We then state results relating this quantity to the minimum degree of networks and prove these results algorithmically.

Formulation

We recall the framework proposed by Goemans and Williamson [60] for specifying connectivity requirements that we introduced in Section 3.3. This framework captures a wide variety of specifications of requirements, including Steiner and generalized Steiner connectivity, general point-to-point connectivity, and T-joins. In this chapter, we show how to find a network satisfying the requirements that has nearly minimum degree. All we use of [60] is their framework for specifying connectivity requirements; our algorithm and analysis are based on the work of Fürer and Raghavachari [50].

Consider a spanning tree of a graph, and any cut in the graph. At least one edge of the spanning tree must cross this cut. Conversely, if a network crosses every cut, it must span all nodes. More generally, in order to specify connectivity requirements for a network, we designate a subset of the cuts in a graph as *active* cuts, and we require that the network cross every active cut.

Goemans and Williamson [60] introduced the formalism of specifying which cuts are active using a 0–1 function f on the node-subsets of a graph. For a node-subset S, recall that $\Gamma(S)$ denotes the set of edges each having exactly one endpoint in S. To specify that the cut $\Gamma(S)$ is active, we set $f(S)$ to be 1. Using this formalism, one can formulate an integer program for a minimum-degree network crossing all active cuts:

Min d

subject to constraints:

$$\begin{array}{rcll} x(\Gamma(S)) & \geq & f(S) & ;\ \forall\, \emptyset \neq S \subset V \\ \sum_{e\in\Gamma(\{v\})} x_e & \leq & d & ;\ \forall\, v \in V \\ x_e & \in & \{0,1\} & ;\ \forall\, e \in E \end{array} \qquad \text{(IP)}$$

Here $x(F) = \sum_{e \in F} x_e$. We call any feasible solution to (IP) an f-join. Minimal f-joins are forests [60].

Goemans and Williamson [60] focused on the class of *proper* function f. Recall that a function $f : 2^V \rightarrow \{0, 1\}$ is proper if it obeys the following properties: (i) [Null] $f(\emptyset) = 0$; (ii)[Symmetry] $f(S) = f(V - S)$ for all $S \subseteq V$; and (iii)[Disjointness] if A and B are disjoint, then $f(A) = f(B) = 0$ implies $f(A \cup B) = 0$. We are interested in solutions to (IP) for the class of proper functions f. One of the main results of this chapter is the following.

Theorem 5.3.1 *Suppose f is a proper function. Let d^* denote the optimum value of (IP). There is an $n^{O(\log_{1+\epsilon} n)}$-time algorithm for finding a feasible solution to (IP) whose objective value is at most $(1 + \epsilon)d^* + O(\log_{1+\epsilon} n)$ for any $\epsilon > 0$.*

Some examples of problems that can be solved approximately using the algorithm in the above theorem are the minimum-degree generalized Steiner forest problem, the minimum-degree T-join problem, and minimum-degree point-to-point connection problems. As we mentioned earlier, we present an approximation algorithm with a better (logarithmic) performance ratio and polynomial running time for the problem addressed in Theorem 5.3.1 using entirely different techniques in Section 6.4

Toughness, Weakness and generalizations: A lower bound

Toughness, first defined by Chvátal in [26], is a measure of how tough it is to disconnect a graph into many components by removing few nodes. The toughness of a graph is the minimum ratio of nodes removed to the number of components in the remaining graph. That is, it is the minimum ratio of $|X|$ to the number of components in $G - X$ where X ranges over all non-trivial node-subsets of G. Computing the toughness of a graph was recently shown NP-complete by Bauer, Hakimi and Schmeichel [68]. The definition of toughness we have given differs slightly from Chvátal's original definition in [26]. According to our definition, the minimum toughness ratio is never more than 1 (for nonsingleton graphs), since even a singleton X yields a ratio of at most 1. Chvátal defines toughness to be the same minimum ratio, but the minimum is taken only over those node-subsets X for which $G - X$ has at least two components. According to this definition, the minimum ratio can be arbitrarily large.

We generalize the above notion to allow active components defined by proper functions f. For any $X \subseteq V$, a connected component of $G - X$ is *active* under a proper function f whenever $f(X) = 1$. The f-toughness of a graph for any given function f is the minimum ratio of the nodes removed to the number of active components formed. In other words, it is the minimum ratio of $|X|$ and the number of active components in $(G - X)$ where X ranges over all non-trivial node-subsets of G. For any subset X of nodes, this ratio is termed the f-toughness of the set X. Thus the

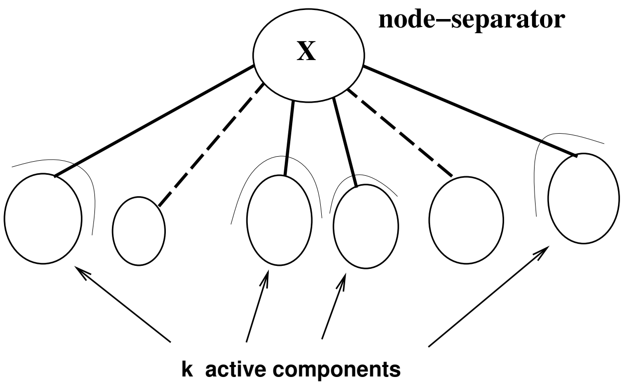


Figure 5.1: The f-weakness of the graph is a lower bound on the maximum degree of any node in any f-join.

f-toughness of the graph is the minimum f-toughness of any of its non-trivial node subsets. Note that ordinary toughness is a special case of f-toughness as we have defined it since it corresponds to a function f where every non-trivial subset of the nodes of the graph is active. We are interested in computing the f-toughness of the graph for proper functions f. Call any single node that forms an active set a *terminal*. Note that as long as there is at least one terminal, the f-toughness ratio is at most 1, since this is the ratio achieved on removing a single terminal.

We shall refer to the reciprocal of the f-toughness of a graph as its *f-weakness*. The notion of weakness of a graph arises as a dual to the problem of constructing minimum-degree networks. In fact, this notion can be arrived at by looking at the dual of (IP) and assigning values uniformly to all the variables in it as described in Section 1.9. The following lemma can then be proved in this way as a consequence of weak duality. However, to illustrate the relationship between the minimum value of the degree of any f-join and the f-weakness of the graph more directly, we provide a simple proof.

Lemma 5.3.2 *For any function f, the optimum value of (IP) is at least the f-weakness of the graph.*

Proof: Consider the node set X achieving the optimal f-weakness (See Figure 5.1). Let the number of active components in $G - X$ under f be k. In any f-join, each of the k active components in $G - X$ must have an edge to X. Thus the average degree of a node of X in an optimal solution to (IP) is $k/|X|$, the f-weakness of the graph. Hence the maximum degree of any node in X in an optimal solution to (IP) is also at least $k/|X|$. □

An approximate min-max relation and applications

How good a lower bound is the f-weakness of a graph for the minimum degree problem? In the course of proving the performance guarantee for the solutions we construct for (IP), we demonstrate an approximate min-max equality between the optimum value of (IP) and the f-weakness of the graph.

Theorem 5.3.3 *The optimum value of (IP) is at most* $(1+\epsilon)w^* + O(\log_{1+\epsilon} n)$ *for any* $\epsilon > 0$ *where* w^* *is the* f*-weakness of the graph.*

The proof of Theorem 5.3.3 is algorithmic. We provide an algorithm that constructs a f-join whose degree is close to the f-weakness ratio of a set we identify. Theorem 5.3.3 and Lemma 5.3.2 together prove the performance bounds given in Theorem 5.3.1. In addition, we also have the following result about approximating the f-weakness of the graph for any proper function f.

Theorem 5.3.4 *Let* f *be a proper function on the nodes of an* n*-node graph. Let* w^* *denote the* f*-weakness of a graph and let* ϵ *be a small positive constant. There is an* $n^{O(\log n)}$*-time algorithm to determine a node subset* X *for which the* f*-weakness ratio is at least* $(1-\epsilon)w^* - O(\log_{1+\epsilon} n)$.

The f-weakness of the graph is an estimate of the vulnerability of any f-join. A node subset with high value of f-weakness represents a weak spot in such a network: damage to this subset results in nearly the worst disconnection among the nodes in the network.

An important application of the approximate min-max equality presented in Theorem 5.3.3 arises from the fact that the relaxed version of (IP) is polynomially solvable. This follows from the observation that separation over (IP) is equivalent to solving an instance of finding the minimum-cut around any active set. The latter problem can be solved using the fact that proper functions are uncrossable and can be inferred from the results in [52, 154]. Using the Ellipsoid method [64], which provides a polynomial-time reduction of the optimization problem to the separation problem, we have that the fractional relaxation of (IP) can be solved in polynomial-time. It follows from Theorem 5.3.3 that the value of this linear program is a good estimate of the minimum degree. While solving the linear program does not give us a solution network with this degree, just knowing the value can be useful, namely in a branch-and-bound search for an optimal solution [119].

The algorithm and its analysis

We now describe the algorithm for providing an approximate solution to (IP) and prove Theorem 5.3.3. As input we are given an undirected graph $G = (V, E)$, an arbitrary number $\epsilon > 0$ to determine the performance accuracy, and a proper function f defined on G. We can assume without loss of generality that the graph G is connected, for otherwise we can work on each connected piece independently. We find a forest F that is feasible for the covering constraints of (IP) using an iterative local-improvement algorithm. Note that we can always assume without loss of generality that any feasible solution to (IP) is the incidence vector of a forest [60]. The claim on the running time of the algorithm is proved using a potential function argument.

Lemma 5.3.5 ([60]) *Let F denote a subset of the edge set of G. F is a feasible solution to (IP) if and only if $f(N) = 0$ for each connected component N of F.*

Overview

The algorithm starts with a feasible solution to (IP) and iteratively applies improvement steps aimed at reducing the degree of high-degree vertices. Intuitively, if we find a cycle in the graph that contains a node w of high degree, and if all edges that must be added to the current feasible solution to form this cycle are incident to low degree nodes, then we can improve the current solution by adding in the cycle and deleting an edge incident to w. This reduces the degree of w by one.

Constructing an initial feasible solution

We start with any spanning tree T of the given connected graph as the initial feasible solution. Note that the null and symmetry conditions on f imply that $f(V) = 0$ and so, by Lemma 5.3.5, T is feasible.

An improvement step

Denote the set of nodes in the current forest with degree at least d by S_d. An improvement step with target d tries to reduce the degree of a node in S_d. We use two types of improvement steps. The first and simpler type determines whether the forest F remains feasible on deleting an edge incident on a node in S_d. If so, we delete this edge from F to obtain the new forest F' and proceed to the next improvement step. The second type of improvement step is more involved. Starting from G, we delete all the edges in $E - F$ that are incident to nodes in the forest having degree at least $d - 2$, i.e., edges that are incident to nodes in S_{d-2}. In this graph, we examine if any node in S_d is in a cycle. If so, we add all the edges in $E - F$ in this cycle to the forest F and delete an edge of F incident to the the node in S_d in this cycle. This gives a new forest F'.

Note that after performing an improvement step with target d, the degree of a node in S_d reduces by one and the resulting degree of each of the affected vertices increases by at most two and is at most $d - 1$. We note that the resulting forest F' is a f-join.

Lemma 5.3.6 *The forest F' formed at the end of an improvement step is feasible.*

Proof: The lemma follows immediately if the improvement step is of the first type. Therefore suppose the improvement is of the second type. Let F be the feasible forest before the improvement step. Consider the cycle involved in the improvement. Since F is acyclic, the addition of the cycle and the deletion of an edge clearly leaves F' acyclic. But, we have merged many of the trees in F to form a single tree containing the edges of the cycle in F'. However, since F was feasible, Lemma 5.3.5 guarantees that each of the trees in the merged tree has f value 0. Since the merged component is a disjoint union of these components, the disjointness property of f implies that the f value of this new component is also 0. All the remaining components in F' were also components of F having f-value 0. Applying Lemma 5.3.5, we conclude that F' is feasible. □

The algorithm

The algorithm starts with an initial feasible solution. Let the maximum degree of any node in the current feasible solution be k. We apply improvement steps with target d for $d \geq k - \lceil \log_{1+\epsilon} n \rceil$ until no such improvements are possible. The resulting forest is our *locally optimal* approximate solution.

Definition 5.3.7 *Define an edge e of a feasible forest F to be* critical *if $F - e$ is not feasible.*

As a result of performing the first type of improvement step, note that all the edges incident on nodes of degree at least $k - \lceil \log_{1+\epsilon} n \rceil$ are critical in the locally optimal solution.

Termination

Each improvement step is polynomial-time implementable. We adapt a potential-function argument from [50] to bound the number of improvement steps. Define the potential of a vertex v with degree d in the forest F to be $\phi(v) = n^d$ where n is the number of nodes in the graph. The potential of the forest F is defined as $\phi(F) = \sum_v \phi(v)$. Initially $\phi(F) \leq n^n$. Each improvement step with target d reduces the potential by at least n^{d-2}, since the degree of a node of degree d is reduced by 1 and the degree of all the other nodes may increase to $d-1$. Since $d \geq k - \lceil \log_{1+\epsilon} n \rceil$ where k is the maximum degree of the current forest, this reduction in potential is at least a fraction $n^{-O(\log_{1+\epsilon} n)}$ of the current potential of the forest $\phi(F)$. It follows that the number of improvement steps is at most $n^{O(\log_{1+\epsilon} n)}$.

Performance guarantee

We prove the performance guarantee by identifying a node separator from the solution subgraph whose f-weakness value is very close to the degree of the solution. Let k denote the maximum degree of the solution subgraph. The following proposition is immediate using a simple contradiction argument.

Proposition 5.3.8 *There is some* $i \in [k - \lceil \log_{1+\epsilon} n \rceil, k]$ *with* $|S_{i-2}| \leq (1+\epsilon)|S_i|$.

Proof of Theorem 5.3.3: Let i be as in Proposition 5.3.8. We prove Theorem 5.3.3 by estimating the f-weakness of S_{i-2}. In the following lemma, we show that the number of active components in $G - S_{i-2}$ is at least $i|S_i| - (2+\epsilon)|S_i|$. By choice of i, note that $|S_{i-2}| \leq (1+\epsilon)|S_i|$. Therefore the f-weakness of S_{i-2} and hence of G is at least $\frac{i-(2+\epsilon)}{1+\epsilon}$. Note that the degree of the solution F is at most $i + \log_{1+\epsilon} n$. Combining these proves Theorem 5.3.3. □

Lemma 5.3.9 *Let* F *be the locally optimal solution network and let* i *be the index in Proposition 5.3.8. The number of active components in* $G - S_{i-2}$ *is at least* $i|S_i| - (2+\epsilon)|S_i|$.

We prove the above lemma in the remainder of this section. Call a tree in F *relevant* if it contains a node in S_i. Define the equivalence class $\sim$ on the edges of F by the rule: $e_1 \sim e_2$ if there is a path between an endpoint of e_1 and an endpoint of e_2 in F avoiding S_i. Define an auxiliary forest F_i whose nodes are S_i together with the equivalence classes of $\sim$. F_i is bipartite, and there is an edge between a node $v \in S_i$ and an equivalence class C of $\sim$ if there is an edge in C incident on v. Note that since F is acyclic, there is at most one edge in any equivalence class C incident on any node v in S_i. Thus if a node $v \in S_i$ has degree j in F, it also has degree j in F_i. Note also that F_i is acyclic and has exactly as many components as the number of relevant trees in F.

Claim 5.3.10 *The number of equivalence classes* C *of degree one in* F_i *is at least* $(i-2)|S_i| + 2r$ *where* r *is the number of relevant trees in* F.

Proof: Let t be the number of classes of degree one and let p be the number of other classes. The number of edges in the forest F_i is the number of nodes in the forest minus the number of components in it. This is exactly $t + p + |S_i| - r$. The sum of the degrees of the nodes of F_i is twice the number of edges. That sum is at least $i \cdot |S_i| + 2p + t$. The claim follows. □

Each class C of degree one defines a set X_C of nodes as follows: a node v is in X_C if C contains all the edges of F that are incident on v. Note that each X_C is a component of $F - S_i$ and thus represents a subset of nodes of a relevant tree (See Figure 5.2). Conisder an edge in F incident on a node in S_i. Since F is feasible, by Lemma 5.3.5, the component of F containing this edges is inactive. Since this edge is critical on account of the first type of improvement step, the two distinct components formed in F on removing this edge and containing its endpoints must both be active. Consider an edge e in a degree-one equivalence class C of $\sim$ that is incident on a node in S_i. One of the connected components formed on its removal from F is X_C, and so this set is active. This is summarized in the following proposition.

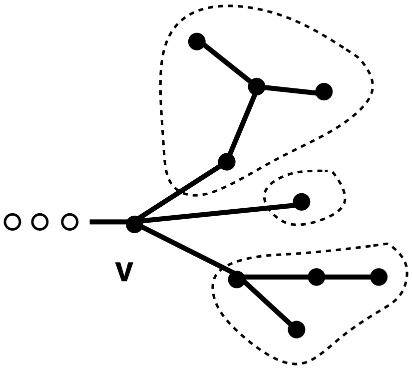


Figure 5.2: Each equivalence class C of $\sim$ with degree one in F_i defines a set of nodes X_C, namely those nodes all of whose incident edges in F are in C. In the figure above, v is a node in S_i and three sets identified using degree-one equivalence classes of $\sim$ are shown.

Proposition 5.3.11 *Each set X_C defined by a class C of degree one in F_i is an active set.*

We now derive a lower bound on the number of active components of $G - S_{i-2}$. Most of the node sets representing degree-one classes of F_i are useful in forming these active components. There are two ways in which such a set might fail to be an active component of $G - S_{i-2}$. When nodes of $S_{i-2} - S_i$ are removed, some of these components may break up, but few can break up this way. By choice of i, the number of such components is at most $|S_{i-2}| - |S_i|$, which is at most $\epsilon |S_i|$. We will disregard these broken-up components. Secondly, when nonforest edges are added, some of the remaining components may merge, possibly resulting in fewer components and inactive components. We show that in fact there remain many active components.

Claim 5.3.12 *The total number of such sets X_C's that merge with others on adding edges of $G - S_{i-2}$ is at most $2(r-1)$.*

Proof: Suppose for a contradiction that more than $2(r-1)$ such X_C's merge on adding the edges of $G - S_{i-2}$. We show that this identifies a cycle in $F \cup (G - S_{i-2})$ containing a node of S_i and so F permits an improvement of the second type.

To show this we consider the auxiliary graph F_i and augment it with edges representing merging X_C's. For every pair of X_C's that merge, we include an edge between the nodes corresponding to the them in F_i. If more than $2(r-1)$ such X_C's merge, then more than $r-1$ edges must be included in the auxiliary graph in addition to F_i. Since F_i has exactly r connected components, and more than $r-1$ edges are added to it, the resulting graph must be cyclic. A cycle in this graph directly corresponds to a cycle in $F \cup (G - S_{i-2})$ containing a node of S_i and so F permits an improvement of the second type. This contradicts the local optimality of F. $\Box$

The sets X_C's that remain might merge with inactive components of F on adding the nonforest edges of $G - S_{i-2}$. By the properties of a proper function, the resulting component in $G - S_{i-2}$ containing X_C is also active. By Claim 5.3.10, the number of degree-one equivalence classes is at least $(i-2)|S_i| + 2r$, and by Proposition 5.3.11 each of these defines an active set. We discount at most $\epsilon |S_i|$ of them that may break up on removing nodes in $S_{i-2} - S_i$. The number of such classes that may merge with one another is at most $2(r-1)$ by Claim 5.3.12. Thus the number of such active sets that remain is at least $i|S_i| - (2+\epsilon)|S_i|$. This completes the proof of Lemma 5.3.9.

5.4 Minimum-Degree Two-Edge-Connected Subgraphs

In this section, we consider minimum-degree networks of a different type. Given as input a two-edge-connected undirected graph, we consider the problem of finding a two-edge-connected spanning subgraph of minimum degree. This problem can be easily seen to be NP-hard by a reduction from the Hamiltonian cycle problem. Using a local-improvement algorithm similar to that used in proving Theorem 5.3.1, we obtain an approximate solution for this problem as well.

Theorem 5.4.1 *Let Δ^* be the minimum degree of a two-edge-connected subgraph of a given graph G and let ϵ be a positive number. There is an $n^{O(\log_{1+\epsilon} n)}$-time algorithm for finding a two edge-connected subgraph of G having degree at most $(1+\epsilon)\Delta^* + O(\log_{1+\epsilon} n)$.*

As in Section 5.3, it is easy to cast this problem as an exponential-sized integer program. The fractional relaxation of this program is solvable in polynomial time by using a minimum-cut procedure to solve the corresponding separation problem. As before, the above theorem can be applied to show that this linear program solution value is a very good estimate of the value of the exact solution to the minimum-degree problem. As we mentioned earlier, such estimates can be used in pruning the search space for exact solutions to this problem.

Preliminaries

Let $G = (V, E)$ be an arbitrary k-edge-connected graph. Consider the problem of computing a k-edge-connected subgraph N of G which spans V such that the maximum degree of N is minimum. The case of $k = 1$ corresponds to the minimum-degree spanning tree problem and is NP-hard [53]. So far there have been no results when $k > 1$. In this section, we consider the case when $k = 2$ (two-edge-connected graphs are also called bridge-connected). In this section, "connectivity" always stands for edge connectivity (and not vertex connectivity). For a graph G, we use $\Delta^*(G)$ to denote the minimum degree of a two-connected spanning subgraph of G.

We first give some preliminary definitions which show that the notion of k-connected components is meaningful in the context of edge connectivity. It allows the partition of V into k-connected components.

Definition 5.4.2 *A pair of vertices u and v are said to be k-*connected *in a graph N if there are k edge-disjoint paths between u and v in N. This relation is an equivalence relation. It partitions the vertices into equivalence classes of vertices. Within each class, each pair of vertices is k-connected. Such a class is called a k-*component.

Definition 5.4.3 *For a graph H, the* bridge-connected forest (bcf) *of H is obtained by contracting each two-component of H to a supervertex and deleting self-loops. A* leaf two-component *of H is a two-component that has degree one in the bcf of H. An* isolated two-component *of H is one of degree zero in the bcf of H.*

The following lemma is obvious from the definition of two-edge-connected graphs.

Lemma 5.4.4 *Let N be any 2-connected spanning subgraph of G, and let X be any subset of vertices. Let C be a two-component of $G - X$. If C is a leaf two-component, then there is at least one edge of N between C and X. If C is an isolated two-component, then there are at least two edges of N between C and X.*

The above lemma motivates the following definition.

Definition 5.4.5 *Let H be a graph with ℓ leaf two-components and k isolated two-components. Then the* deficiency *of H is defined as $\ell + 2k$ and is denoted by* $\mathrm{def}(H)$.

We now derive an easy lower bound on the minimum degree of a two-connected spanning subgraph. Consider removing a node subset X from an undirected graph G; the remaining graph has deficiency $\mathrm{def}(G - X)$. In any two-connected spanning subgraph of G, there must be at least $\mathrm{def}(G - X)$ edges going between $G - X$ and the nodes in X by Lemma 5.4.4. Thus, the maximum degree of a node in X is at least $\frac{\mathrm{def}(G-X)}{|X|}$. Applying this argument to the minimum-degree two-connected spanning subgraph, we get the following corollary.

Corollary 5.4.6 *Let X be a subset of the nodes of a graph G. Then*

$$\Delta^*(G) \geq \frac{def(G - X)}{|X|}$$

The algorithm and its time-complexity

The local-search algorithm proceeds as follows. Fix ϵ to be an arbitrary quantity greater than zero. The algorithm starts with an arbitrary 2-connected subgraph, and iteratively applies local improvement steps to reduce the degrees of high-degree vertices. When a local minimum is reached, the algorithm outputs the current subgraph.

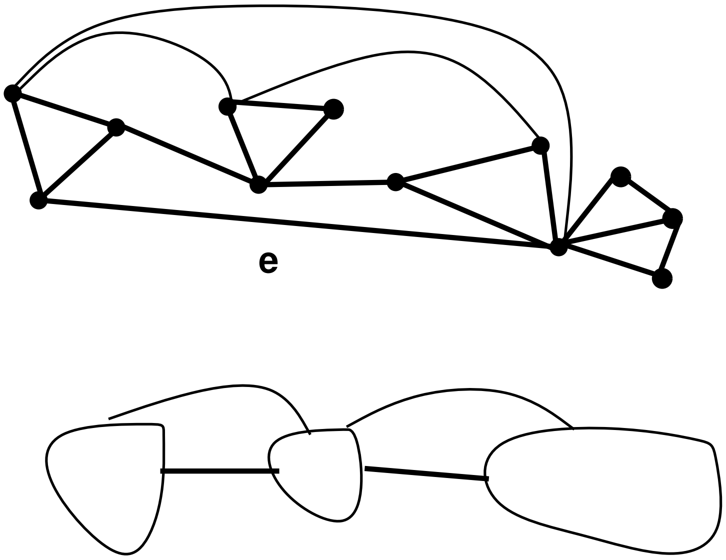


Figure 5.3: An example of a degree-5 candidate graph for an edge e. The figure on top illustrates the network N with dark edges. The figure in the bottom is the degree-5 candidate graph for the edge e formed by contracting each of the three two-components of $N - e$ to single nodes and deleting edges in $G - N$ incident on nodes of degree five or more.

Now we give definitions pertaining to an improvement step. Let N denote a two-connected subgraph of G. For an edge $e \in N$, we define N_e to be the bcf of $N - e$. Since N is two-connected, it follows that N_e is a simple path.

Definition 5.4.7 *The* degree-d candidate graph *for an edge e in N is obtained from G as follows: contract each two-component of $N - e$, and then delete e and the edges of $G - N$ incident to nodes of degree more than d in N (See Figure 5.3).*

A vertex w of degree d in N is said to admit an improvement *if there is an edge e incident to w in N such that the degree-$(d - 3)$ candidate graph for e is two-connected.*

An improvement step consists in replacing e with edges from the candidate graph so that the resulting network remains two-connected. We require that the degree of no vertex is increased beyond $d - 1$. The following lemma shows that this can be done. We use $deg_H(v)$ to denote the degree of a node v in the subgraph formed by a collection H of edges.

Lemma 5.4.8 *Let Q be a minimal collection of edges whose addition two-connects N_e. Then for every node v of N_e, $\deg_Q(v) \leq 2$.*

Proof: Suppose that $\deg_Q(v) \geq 3$. Since N_e is a path, v splits N_e into two subpaths. Then at least two of v's three incident edges must be incident on nodes in the same subpath. Let e be the one whose other endpoint is closer to v in the subpath. Then $Q - e$ also 2-connects N_e, contradicting the minimality of Q. □

Note that an improvement step for a vertex w reduces w's degree by one, and does not increase the degree of any vertex to more than w's new degree.

Definition 5.4.9 *Let N be a two-connected spanning subgraph of G, and let Δ be the degree of N. Then N is called* locally optimal *if no vertex of degree at least $\Delta - \lceil \log_{1+\epsilon} n \rceil$ admits an improvement.*

Using a potential function argument as in the previous section, it is easy to show that $n^{O(\log_{1+\epsilon} n)}$ improvement steps suffice to yield a locally optimal subgraph. Since each improvement step can be executed in polynomial time, the running time is as claimed in Theorem 5.4.1. In the remainder of this section, we show that a locally optimal subgraph has degree not much more than Δ^*.

Performance guarantee

Theorem 5.4.10 *The degree of any locally optimal subgraph of a graph G with n nodes is at most $(1+\epsilon)\Delta^*(G) + \lceil \log_{1+\epsilon} n \rceil + 4(1+\epsilon)$.*

Let S_i denote the set of nodes whose degree in N is at least i. The proof of Theorem 5.4.10 consists in showing that there is an i in the range $[\Delta - \lceil \log_{1+\epsilon} n \rceil, \Delta]$ for which $G - S_i$ has many leaf and isolated two-components. We show this in Lemma 5.4.13. Theorem 5.4.10 then follows from Corollary 5.4.6.

Lemma 5.4.11 *Let G be a 2-connected graph, and let S be a set of vertices of G. Then there is a 2-connected subgraph A of G that contains all vertices of S, such that $\sum_{v \in S} \deg_A(v) \leq 4|S|$.*

Proof: We provide an algorithm to compute A. A graph H is defined to be a minor of graph G if one can obtain H from G by repeatedly contracting or deleting edges. For a minor H of G, we say a vertex v of H is *active* if one of the vertices contracted to form v belongs to S. Consider the following algorithm:

- Initialize $H_0 := G$.
- Let $j := 0$ and let A_0 be the empty graph.
- While S is not 2-connected via A_j, do
  - Let C_{j+1} be a shortest cycle in H_j containing at least two active vertices.

- Let $A_{j+1} := A_j \cup C_{j+1}$.
- Obtain H_{j+1} from H_j by contracting together all vertices of C_{j+1}.
- $j := j + 1$.

Let k be the number of iterations. By the termination condition, S is 2-connected in A_k. For each j, let x_j be the number of active vertices in C_j. Since the edges of C_j form a cycle in H_j, the degree due to these edges on active vertices in the cycle is at most twice the number of active vertices in the cycle.

$$\sum_{v \in S} deg_{C_j}(v) \leq 2x_j. \tag{5.1}$$

It follows that

$$\sum_{v \in S} \deg_A(v) = \sum_{j=1}^{k} \sum_{v \in S} \deg_{C_j}(v) \leq \sum_{j=1}^{k} 2x_j \tag{5.2}$$

Next we bound the right-hand side of (5.2). Since at each iteration j, we contract x_j active vertices of H_j into a single active vertex in H_{j+1}, we have the following claim.

Claim 5.4.12 *(No. of active vertices in H_{j+1}) = (no. of active vertices in H_j) $-(x_j - 1)$*

Since H_k contains 1 active vertex and H_0 contains $|S|$ active vertices, it follows by Claim 5.4.12 that

$$\sum_{j=1}^{k} (x_j - 1) = |S| - 1 \tag{5.3}$$

In each iteration $x_j - 1 \geq 1$, so the number of iterations k is at most $|S| - 1$. Hence $\sum(x_j - 1) \geq \sum x_j - (|S| - 1)$. Combining this inequality with (5.3) yields $\sum x_j \leq 2(|S| - 1)$. Combining this with (5.2) yields $\sum_{v \in S} \deg_A(v) \leq 4(|S| - 1)$. This completes the proof of Lemma 5.4.11. □

Lemma 5.4.13 *Let N be a locally optimal subgraph with degree Δ. Then there is an integer i in the range $[\Delta - \lceil \log_{1+\epsilon} n \rceil, \Delta]$ such that the deficiency of $G - S_{i-2}$ is at least $|S_i|(i - 4(1 + \epsilon))$ and $|S_{i-2}| \leq (1 + \epsilon)|S_i|$.*

Proof: As in Proposition 5.3.8, there is an i in the given range such that $|S_{i-2}| \leq (1 + \epsilon)|S_i|$. By Lemma 5.4.11, there exists a 2-connected subgraph A of N such that S_{i-2} is contained in A and $\sum_{v \in S_{i-2}} \deg_A(v) \leq 4|S_{i-2}|$. By choice of i, the value $4|S_{i-2}|$ is at most $4(1 + \epsilon)|S_i|$. To continue the proof, we show that a large portion of the degree of the nodes in S_{i-2} can be used to infer that $G - S_{i-2}$ has high deficiency.

Definition 5.4.14 *We say an edge e of a 2-connected graph H is* critical *in H if $H - e$ is not 2-connected.*

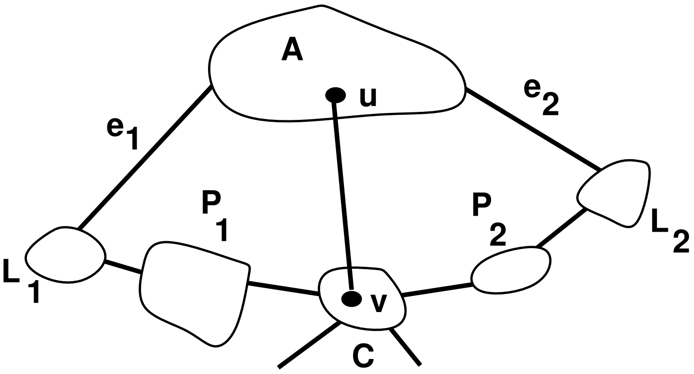


Figure 5.4: The edge (u, v) goes between $u \in S_{i-2}$ and a vertex v in a two-component C in the bcf of $G - S_{i-2}$. Suppose C is internal, we can identify two edge-disjoint paths between u and C in the degree-$(i-3)$ candidate graph for e contradicting the local optimality of the network N.

By local optimality, for every edge e of N incident on S_i, the degree-$(i-3)$ candidate graph for e is not two-connected. Note that this also implies that the edge e is critical in N.

Claim 5.4.15 *Let e be an edge of N not in A such that one endpoint of e belongs to S_i. Then the other endpoint of e belongs to a leaf 2-component or isolated 2-component of the* bcf *of $G - S_{i-2}$.*

Proof: We give a proof by contradiction. Let $e = \{u, v\}$, where u belongs to S_i. First suppose that v belongs to A; this includes the case where v belongs to S_{i-2}. Since A is 2-connected and contains u, it follows that e is noncritical, which is a contradiction. It follows that v belongs to some 2-component C of $G - S_{i-2}$. By the same argument as above, C contains no vertices of A.

Next suppose that C has degree two or more in bcf$(G - S_{i-2})$. Then C is internal to some path in bcf$(G - S_{i-2})$ between leaf 2-components L_1 and L_2. For $j = 1, 2$, let P_j be the subpath from C to L_j. By Lemma 5.4.4, there is an edge e_j of N between each L_j and S_{i-2} for $j = 1, 2$. We have constructed edge-disjoint paths P_1e_1 and P_2e_2 from C to A. The edges of these paths are in $N \cup (G - S_{i-2})$ (See Figure 5.4). Recall that A is two-connected and contains only edges in N, and that u is in A. It follows that C is two-connected to u using edges in $(N \cup (G - S_{i-2}) - \{e\})$. This contradicts the fact that the degree-$(i-3)$ candidate graph for e is not two-connected. □

Now we complete the proof of Lemma 5.4.13. The sum of the degrees of all the nodes of S_i in N is at least $i \cdot |S_i|$. By choice of the set A, we have $\sum_{v \in S_{i-2}} \deg_A(v) \leq 4(1+\epsilon)|S_i|$. Hence there are at least $|S_i|(i - 4(1+\epsilon))$ edges incident on S_i not in A. By Claim 5.4.15, each of these edges goes to a leaf 2-component or isolated 2-component of bcf$(G - S_{i-2})$. By the criticality of these edges in N, it also follows that there is at most one such edge to a leaf-component and at most two such edges to a isolated-component in bcf$(G - S_{i-2})$. Hence each such edge contributes one to the deficiency of $G - S_{i-2}$. This concludes the proof of Lemma 5.4.13. □

This chapter concludes our study of single-objective network-design problems. In Chapters 2, 3 and 4, we studied problems in which the the total cost of the network is to required be minimized. In this chapter, we studied problems in which the maximum degree of any node in the network is required to be minimized. In the next chapter, we turn to problems where we require a single network in which both these costs are low.

Chapter 6

Multi-objective Approximation Algorithms

6.1 Problem Definitions and Results

In the previous chapters we studied single-objective network-design problems where either the cost of the network or the maximum degree of the network is required to be minimized. In this chapter, we study network-design problems with multiple design objectives. In particular, we look at problems where both the cost measures we addressed earlier, the total cost of the network and the maximum degree of any node, are required to be simultaneously minimized in the network. We formulate this problem as one of minimizing the total cost of the network subject to a constraint on the maximum degree of any node in the network. Our main result can be roughly stated as follows: given an integer b, we present polynomial-time approximation algorithms for a variety of network-design problems on an n-node graph such that the degree of the output network is $O(b\log(\frac{n}{b}))$ and the cost of this network is $O(\log n)$ times that of a minimum-cost degree-b-bounded network. Our algorithms can handle costs on nodes as well as edges. Moreover, we can construct such networks so as to satisfy a variety of connectivity specifications including spanning trees, Steiner trees and generalized Steiner forests.

We also address in this chapter the special case in which the costs obey the triangle inequality and present approximation algorithms with better performance guarantees. For the problem of constructing spanning networks in this special case, we also show how to simultaneously approximate yet another objective: the maximum cost of any edge in the network. The results in this chapter were obtained jointly with Madhav V. Marathe, S. S. Ravi, Daniel J. Rosenkrantz, and Harry B. Hunt III, and appeared in [129].

Degree-bounded minimum-cost network design

We call the following the b-MST problem: Given an undirected edge-weighted graph and an integer $b \geq 2$, find a spanning tree in which the maximum degree of any node is at most b and the total cost is minimum. We assume that the edge-weights are nonnegative and integral. It is straightforward to show that this problem is NP-complete by a reduction from the Hamiltonian path problem. To examine what kind of approximations we can hope for, we first make the following simple observation (This is proved in the first part of Theorem 6.7.2).

Observation 6.1.1 *For any fixed rational number $R \geq 1$, finding a spanning tree of G of maximum degree at most b and of total cost at most R times that of a minimum-cost degree-b-bounded spanning tree of G is NP-hard.*

Given this difficulty in conforming to the degree restriction in any approximate solution, we set our goals a little lower and seek a solution that is approximate in terms of both the objectives. We present the first such result for approximating the b-MST problem.

Theorem 6.1.2 *There is a polynomial algorithm that, given an undirected graph G on n nodes with nonnegative costs on its edges, and a target degree b, constructs a spanning tree of maximum degree $O(b \log \frac{n}{b})$ and of cost $O(\log \frac{n}{b})$ times that of the minimum-cost b-bounded spanning tree of G.*

Our techniques generalize to the case of constructing Steiner trees as well as generalized Steiner forests. Given an undirected graph and a subset of the nodes called *terminals*, recall that a Steiner tree for the terminals is a subgraph spanning the terminals. Agrawal, Klein and Ravi [2] consider a generalization of Steiner trees called generalized Steiner forests. Given an undirected graph and a set of *site-pairs* of nodes, a generalized Steiner forest for the site-pairs is a subgraph in which there is a path between every site-pair. In [2], they provide the first approximation algorithm for finding minimum-cost generalized Steiner forests. A b-bounded generalized Steiner forest for the site-pairs is a generalized Steiner forest for the site-pairs in which the maximum degree of any node is at most b. We can extend Theorem 6.1.2 as follows.

Theorem 6.1.3 *There is a polynomial-time algorithm that, given an undirected graph G with nonnegative costs on its edges, a set of site-pairs of nodes, and a target degree b, constructs an $O(b \log \frac{k}{b})$-bounded generalized Steiner forest for the site-pairs of cost $O(\log \frac{k}{b})$ times that of the minimum-cost b-bounded generalized Steiner forest for the site-pairs. Here k represents the number of nodes of G that are sites.*

Building on the work of Agrawal, Klein and Ravi [2], Goemans and Williamson [60] considered a class of constrained forest problems using the formalism of proper functions. We described their formalism in Section 3.3. The generalized Steiner forest problem is a prototypical example of such a constrained forest problem. Our techniques directly extend to approximating degree-bounded minimum-cost networks of this general form. We postpone description of this extension until the relevant section.

Extension: Handling arbitrary degree specifications

Theorems 6.1.2 can be generalized in a very nice way. Instead of specifying an integer b as the bound on the maximum degree of any node in the networks to be constructed, we can allow an arbitrary integer-valued function d that assigns a nonnegative integer value to each vertex v in the graph. The value $d(v)$ specifies the maximum allowable degree of the network at node v. Given a function d, we can define a d-constrained minimum-cost network design (spanning tree, Steiner tree etc.) problem as follows: Find a minimum-cost network in which the degree of node v is at most $d(v)$. We have the following generalization of Theorem 6.1.2.

Theorem 6.1.4 *There is a polynomial-time algorithm that, given an undirected graph G on n nodes and a function d assigning nonnegative values to the nodes of G, constructs a spanning tree of G of cost $O(\log n)$ times that of a minimum-cost d-constrained spanning tree in which the degree of any node v is $O(d(v) \log n)$.*

Like Theorem 6.1.2, the above theorem also generalizes to approximating more general d-constrained forest problems such as Steiner trees and generalized Steiner forests.

Extension: Approximating minimum weighted-degree spanning trees

We can extend the techniques used to prove Theorem 6.1.2 to approximate the following generalization of the minimum-degree spanning tree problem [53, 49, 50]. Given nonnegative weights on the edges, define the weighted-degree of any node in a spanning tree to be the sum of the weights of the edges incident to the node in the tree. The minimum weighted-degree spanning tree problem is that of finding a spanning tree of a given graph in which the maximum weighted degree of any node is minimum. This problem is a natural capacitated generalization of the minimum-degree spanning tree problem which results if all the edges in the graph are assigned unit weight. While there is an approximation algorithm that provides a solution to the minimum-degree spanning tree problem to within an additive error of one [50], no approximations are known for the weighted-degree problem. We provide the first approximation algorithm for this problem.

Theorem 6.1.5 *There is a polynomial-time algorithm that, given an undirected graph G on n nodes with nonnegative weights on the edges, constructs a spanning tree of G in which the maximum weighted-degree of any node is $O(\log n)$ times the minimum possible.*

This theorem, however, does not generalize to the case of more general constrained forest problems addressed in [2, 60]. It does generalize to the case in which, given costs and weights (two distinct measures on the edges), and a bound B on the maximum weighted-degree of any node, we are required to find a minimum-cost spanning tree under this weighted-degree constraint.

Theorem 6.1.6 *There is a polynomial algorithm that, given an undirected graph G on n nodes with nonnegative costs and weights on its edges, and a target weighted-degree B, constructs a spanning tree of maximum weighted-degree $O(B \log n)$ and of cost $O(\log n)$ times that of the minimum-cost spanning tree of G in which the maximum weighted-degree is at most B.*

Extension to the node-weighted case

We can strengthen our results above by considering nonnegative costs on the nodes and aiming to find a small degree network of minimum total cost. However, the performance guarantee on the cost of the solution worsens slightly as a result of this generalization.

The case of spanning trees treated in Theorem 6.1.2 is less interesting in the case of node-weighted graphs since every node must be included in the solution. This problem then reduces to computing a minimum-degree spanning tree that has been well-studied [49, 50].

We focus our attention on the more interesting case of Steiner trees. In Chapter 4, we presented the the first polynomial-time approximation algorithm for node-weighted Steiner trees. The performance guarantee of this approximation algorithm is logarithmic in the number of terminals specified in the problem. We can extend this result to the case when the degree of the Steiner tree is also required to be bounded and derive the following analogue of Theorem 6.1.2 for Steiner trees.

Theorem 6.1.7 *There is a polynomial-time algorithm that, given an undirected graph G on n nodes with nonnegative costs on its nodes, a subset of k nodes called terminals, and a target degree b, constructs a Steiner tree spanning all the terminals, of maximum degree $O(b \log(\frac{k}{b}))$ and of cost $O(\log k)$ times that of the minimum-cost b-bounded Steiner tree of G spanning the terminals.*

Note that a Steiner tree problem with costs on nodes and edges can be transformed to one with costs only on the nodes: replace every edge $e = (u, v)$ of cost $c(e)$ by two edges (u, x_e) and (x_e, v) where x_e is a new node of cost $c(e)$. Thus the above theorem is a strict generalization of Theorem 6.1.2.

As before, we can extend the above theorem to generalized Steiner forests and to more general constrained forest problems addressed in [2, 60].

An application: Approximating minimum-degree generalized Steiner forests

Theorem 6.1.3 has an important application. We can use this theorem to provide a polynomial-time approximation algorithm for a class of minimum-degree forest problems that we considered in Chapter 5. Therein we addressed the problem of finding two-edge-connected spanning subgraphs and one-connected networks of a general form such that the maximum degree is minimum. We presented in Chapter 5 quasi-polynomial ($n^{O(\log_{1+\epsilon} n)}$-time) approximation algorithms for these problems that provide solutions of degree at most $(1+\epsilon)$ times the minimum with an additive error of $O(\log_{1+\epsilon} n)$ for any $\epsilon > 0$. A prototypical example of the one-connected network problem we considered is the minimum-degree generalized Steiner forest problem: given an undirected graph with site-pairs of nodes, find a generalized Steiner forest for the site-pairs in which the maximum degree is minimum. As we mentioned in Chapter 5, these results can be adapted to provide polynomial-time approximation algorithms with performance ratio $\Omega(n^{\delta})$ for any constant $\delta > 0$ by setting $\epsilon = n^{\delta}$. We can improve the approximation factor achievable in polynomial time for this problem by an application of Theorem 6.1.3.

Theorem 6.1.8 *There is a polynomial-time algorithm that, given an undirected graph G on n nodes and a set of site-pairs of nodes, constructs a generalized Steiner forest for the site-pairs in which the maximum degree of any node is $O(\delta^* \log \frac{k}{\delta^*})$. Here k is the number of nodes of G that are sites and δ^* is the minimum degree of any generalized Steiner forest for the site-pairs.*

Approximations under triangle inequality

One way to circumvent the difficulty exhibited in Observation 6.1.1 is to consider more structured cost functions on the edges. In this direction, we turn to the case of cost functions on the edges satisfying the triangle inequality. The underlying graph is assumed to be complete with costs only on the edges and these costs obey the triangle inequality. Define the *bottleneck cost* of a network to be the maximum cost of any edge in it. In this case, we present approximations that strictly conform to the degree restriction in the input problem and approximate the bottleneck cost of the output network as well. We introduce a short-cutting technique to prove the following theorem.

Theorem 6.1.9 *There is a polynomial-time algorithm that, given an undirected graph with edge costs satisfying the triangle inequality and an integer $b \geq 3$, outputs a spanning tree in which the maximum degree of any node is at most b, the total cost of the tree is at most $2 - \frac{(b-2)}{(n-1)}$ times that of a minimum spanning tree, and the bottleneck cost is at most twice that of the minimum-bottleneck spanning tree.*

If we insist on a Hamiltonian path (i.e., require $b = 2$) or a Traveling Salesperson (TSP) tour, then the simple short-cutting heuristic of Rosenkrantz, Stearns and Lewis [137] provides a TSP tour of cost at most $2(1 - \frac{1}{n})$ times that of a minimum spanning tree (MST) in an n-node graph. Deleting an edge from this tour gives a Hamiltonian path with the same guarantee. But there is no guarantee on the bottleneck cost of the tour. We tailor our short-cutting technique to obtain a TSP tour with small total cost as well as small bottleneck cost.

Theorem 6.1.10 *There is a polynomial-time algorithm that, given a undirected graph with edge costs satisfying the triangle inequality, outputs a TSP tour of total cost at most two times the cost of a MST and of bottleneck cost at most three times that of a minimum bottleneck-cost spanning tree.*

Using the above theorem, we can extend Theorem 6.1.9 to higher-connected networks as follows.

Theorem 6.1.11 *There is a polynomial-time algorithm that, given an undirected graph with edge costs satisfying the triangle inequality, an integer $k \geq 2$ (the vertex-connectivity requirement), and an integer $b \geq k + 1$ (the target degree), outputs a k-connected spanning subgraph of G in which the degree of any node is at most b, the total cost of all the edges in the subgraph is at most $k + 2$ times that of a minimum-cost k-connected subgraph, and the bottleneck cost of the subgraph is at most $3 \cdot \lceil \frac{k}{2} \rceil$ times that of a minimum bottleneck-cost spanning tree.*

The above theorem is proved by using short-cuts that induce higher-connected graphs.

In the next section, we discuss related work. Then we present the algorithm for approximating degree-bounded edge-weighted networks, and prove Theorem 6.1.2. After this, we present the proof of its extensions and its applications. We then present the algorithm for degree-bounded node-weighted networks in the following section. Following this, we outline the algorithms for the problems under triangle inequality. Finally, we briefly outline some hardness results and some simple observations about a related problem.

6.2 Related Work

Minimizing one of the cost measures

Much work has been done on approximating each of the cost measures that we simultaneously minimize.

A series of recent results have addressed the problem of minimum-degree networks of various forms: spanning trees [49, 50], Steiner trees [3, 50], generalized Steiner forests and more general one-connected networks as well as two-edge-connected spanning networks [131].

A lot of research effort has been dedicated to the study of minimum-cost network problems: e.g., spanning trees [4, 98, 127], TSPs [48, 100], Steiner trees [74, 156, 164], generalized Steiner trees [2] and even more general one-connected networks [52, 60, 154].

Bottleneck problems have been investigated in [20, 72, 123].

Multi-objective approximations

While there has been much work on finding minimum-cost networks for each of the cost measures that we simultaneously minimize, there has been relatively little work on approximations for multi-objective network-design. In this direction, Bar-Ilan and Peleg [8] considered balanced versions of problems of assigning network centers. In the balanced version, a budget is imposed on the number of nodes that any center can service. They extended existing approximation algorithms for center problems to the balanced versions. Wong [161] examined a budget network design problem in which a network is to be built whose cost is at most a certain budget such that the sum of the path-lengths of commodities to be routed using this network is minimized. He showed that even finding near-optimal solutions is NP-hard if we are required to conform to the budget requirement and approximate only the sum of path-lengths. Lin and Vitter [106] provided approximations for the s-median problem where s median nodes must be chosen so as to minimize the sum of the distances from each vertex to its nearest median. The solution output is approximate in terms of both the number of median-nodes used and the sum of the distances from each vertex to the nearest median. Iwainsky et al. [75] formulated a version of the minimum-cost Steiner problem with an additional cost based on node-degrees. Duin and Volgenant [36], motivated by practical considerations, formulated the degree-bounded Steiner tree problem.

Other researchers have addresses multi-objective approximation algorithms for problems arising in areas other than network design. Agrawal, Klein and Ravi [89] provided an approximation algorithm for finding an elimination ordering for sparse Gaussian elimination to simultaneously minimize the fill-in, the total operation count and the elimination height. Khuller, Raghavachari, and Young [85] gave an algorithm for finding a rooted spanning tree of weight at most a constant times that of a MST such that the distance in this tree from the root is at most a constant times the distance in the input graph. Shmoys and Tardos [140] studied the problem of scheduling unrelated parallel machines with costs associated with processing a job on a given machine. Given a budget on the cost of the schedule, they presented an approximation algorithm for minimizing the makespan of the schedule. Mitchell, Piatko and Arkin [115] studied bicriteria optimization problems arising in computational geometry.

6.3 The Edge-weighted Case

In this section, we sketch a proof of the results on edge-weighted degree-bounded networks.

Background

In this section, we describe some background material on a degree constrained subgraph (DCS) problem. The general DCS problem can be stated as follows: Given an undirected graph with nonnegative costs on the edges, and an integer-valued function f defined on the vertices of the graph, find a minimum-cost subgraph (if one exists) such that the degree of any vertex v in this subgraph is $f(v)$. This problem is referred to as the f-factor problem [109], and is known to be polynomially solvable [38, 109]. The following variant of this problem is also known to be polynomially solvable using matching techniques [109]. Denote the degree of a node v in a subgraph H by $deg_H(v)$.

Fact 6.3.1 *[109] (b-bounded even DCS problem) The following problem has a polynomial-time solution: Given an undirected graph $G = (V, E)$ such that $V = S \cup T$ and $S \cap T = \emptyset$, and an integer $b \geq 2$, find a subgraph H (if one exists) of G of minimum cost such that*

- *for all vertices $v \in T$, we have $deg_H(v) = 1$, and*
- *for all vertices $v \in S$, we have $0 \leq deg_H(v) \leq b$ and $deg_H(v) \equiv 0 \ (\ mod\ 2)$.*

The b-bounded even DCS problem described above is a generalization of the T-join problem [38]. When b is allowed to be unbounded, this problem reduces to the T-join problem.

We now prove a simple tree-decomposition result.

Claim 6.3.2 *Let R be a tree with an even number of marked nodes. Then there is a pairing $(v_1, w_1), \ldots, (v_k, w_k)$ of the marked nodes such that the $v_i - w_i$ paths in R are edge-disjoint.*

Proof: We begin with a definition.

Definition: For any two paths P and Q that intersect only at one endpoint, we use QP to denote the path formed by the the concatenation of these paths at their point of intersection.

Let $(v_1, w_1), \ldots, (v_k, w_k)$ be the pairing minimizing the number of occurrences of edge-overlap between the $v_i - w_i$ paths, P_i, in R. We claim that the paths P_i are edge-disjoint. To see this, suppose for a contradiction that P_r and P_s share an edge xy. Let Q_r^x be the terminal subpath of P_r ending at x and not passing through y. Similarly define Q_r^y, Q_s^x, and Q_s^y. Then by replacing P_r and P_s with $Q_r^x Q_s^x$ and $Q_r^y Q_s^y$, we obtain a pairing that has fewer occurences of overlaps, namely at the edge xy. This contradicts the minimality of the initial pairing. The proof is depicted in Figure 6.1.

□

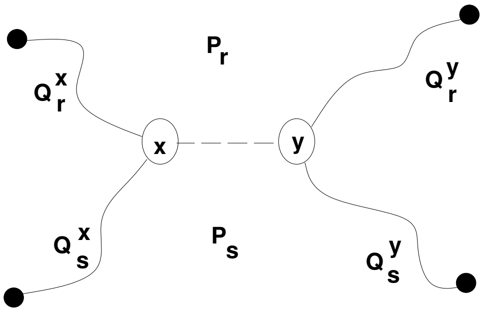


Figure 6.1: P_i is the path in the tree R between v_i and w_i as defined in the proof of Claim 6.3.2. The paths P_i are edge-disjoint. Suppose that P_r and P_s share an edge xy. Let Q_r^x be the terminal subpath of P_r ending at x and not passing through y. Similarly define Q_r^y, Q_s^x, and Q_s^y. Then by replacing P_r and P_s with $Q_r^x Q_s^x$ and $Q_r^y Q_s^y$, we obtain a pairing that has fewer occurences of overlaps, namely at xy. This contradicts the minimality of the initial pairing.

Any minimal solution to the b-bounded even DCS problem is a forest in which each tree contains an even number of nodes of T. Applying the above claim to each tree in such a solution we have the following.

Proposition 6.3.3 *Let H be any subgraph satisfying the conditions in Fact 6.3.1. Then there is a pairing of the nodes of T in H such that there are edge-disjoint paths in H between each pair of vertices.*

We use the above results in the proof of the performance guarantee.

The approximation algorithm for b-MST

We now describe the algorithm referred to in Theorem 6.1.2. We use b to denote the degree bound specified in the problem, and OPT_b to denote the minimum cost of any b-bounded spanning tree of the input graph.

Overview

Our algorithm follows the same skeletal outline as an early algorithm [49] of Fürer and Raghavachari for approximating the minimum-degree spanning tree. However, we generalize it to ensure that the cost of the solution chosen is small as well.

The algorithm works in $O(\log \frac{n}{b})$ iterations where n is the number of nodes in the original graph. To begin with, the solution subgraph is empty and each node is in a connected component by itself in the current solution. At each iteration, we add edges between the components, thereby reducing the number of connected components in the current solution by a factor of half. When the number of connected components falls to $O(b)$ we run a standard MST algorithm to connect up all these components. Thus there are $O(\log \frac{n}{b})$ iterations in all. We also ensure that in each iteration, the degree of any node in the graph increases by $O(b)$ and the set of edges added has cost at most OPT_b. Thus we prove the bound on the degree and the cost of the solution subgraph as stated in Theorem 6.1.2.

The Algorithm

```
1    Initialize the set of edges in the solution subgraph $F := \emptyset$, and the iteration count $i := 1$.
2    Repeat until the number of connected components of $(V, F)$ is $O(b)$
3      Let $\mathcal{C} = \{C_1 \ldots, C_p\}$ be the set of connected components of $(V, F)$.
4      Construct an auxiliary graph $G_i$ as follows.
5        The node set of $G_i$ is $V \cup \mathcal{C}$.
6        We now describe the edges included in $G_i$: Between all the nodes of $V$ in each connected
         component $C_j \in \mathcal{C}$, include edges of zero-cost to form a clique in $G_i$.
         Let $E' \subseteq E$ represent the set of edges of $G$ whose endpoints are in different components
         in $\mathcal{C}$. For each edge $e = (u, v) \in E'$ of cost $c(e)$, include four edges in $G_i$: let $u \in C_u$
         and $v \in C_v$. We include the edges $(u, v), (u, C_v), (v, C_u)$ and $(C_u, C_v)$, each of cost
         $c(e)$, in $G_i$.
7        If $|\mathcal{C}|$ is odd, we include an extra dummy node $z$ in $\mathcal{C}$ and include zero-cost edges between
         $z$ and every $C_j \in \mathcal{C}$ in $G_i$.
8      Find a $2b$-bounded even DCS of $G_i$ with nodes in $V$ having even degree (between 0 and
       2b) and the nodes in $\mathcal{C}$ having degree one.
9      $F := F \cup$ the set of edges of $G$ found by the DCS algorithm in Step 8.
10     $i := i + 1$.
11   The number of connected components of $(V, F)$ is now $O(b)$. Contract each of these connected
     components to single nodes and find a MST of these nodes. Add the edges of this MST to
     the set $F$.
12   Output a spanning tree of $(V, F)$.
```

Note that at each iteration, the solution to the DCS problem may contain one or more copies of an edge $e \in E'$. In any case, in Step 9, we include only one copy of such an edge in the set F.

Performance Guarantee

We prove the performance guarantee using a series of lemmas.

At each iteration, applying Proposition 6.3.3 to the solution to the DCS problem in Step 8, we can derive a pairing of the connected components in $\mathcal{C}$ such that there are paths between these pairs in the solution. Thus at each iteration, the number of connected components of (V, F) reduces by a factor of half. Using this observation, it is easy to prove the following.

Lemma 6.3.4 *The total number of iterations of the above algorithm is* $O(\log \frac{n}{b})$.

The following lemma is immediate from the constraint on the degree of the nodes in the subgraph added at each iteration of the algorithm.

Lemma 6.3.5 *The degree of the spanning tree output by the above algorithm is* $O(b \log \frac{n}{b})$.

Lemma 6.3.6 *At each iteration i of the algorithm, the cost of the solution to the DCS problem in Step 8 in this iteration is at most OPT_b.*

Proof: This is trivial to see in the last iteration of the algorithm since a b-MST of cost OPT_b induces a spanning tree on the remaining $O(b)$ components of cost at most OPT_b. For every other iteration i of the algorithm, we show that there exists a solution of cost at most OPT_b to the DCS problem set up in the iteration.

To construct a feasible solution of value at most OPT_b for the DCS problem on G_i, consider a minimum-cost b-bounded spanning tree T^* of cost OPT_b. Let $\mathcal{C}_i$ represent the set of connected components in $\mathcal{C}$ at the beginning of iteration i. For each component in $\mathcal{C}_i$, contract all the nodes in it to form a single supernode representing this component. It is easy to derive a subgraph T_i of T^* such that T_i is a spanning tree on the supernodes $\mathcal{C}_i$. The cost of T_i is at most OPT_b.

We use the edges of T_i to construct a solution to the DCS problem set up in Step 8. Assume for the sake of simplicity that $|\mathcal{C}_i|$ is even.[1] Applying Claim 6.3.2 to the tree T_i with all the supernodes marked, we can find a pairing $\mathcal{P}$ of all the components in $\mathcal{C}_i$ such that the paths in T_i between the pairs are edge-disjoint. We convert these paths into a feasible solution to the DCS problem as follows:

(i) Consider all pairs of components in $\mathcal{P}$ such that the path between them in T_i is a single edge. Let C_u, C_v be such a pair joined by an edge $(u, v) \in E'$. By the construction in Step 6 of the algorithm, there is an edge (C_u, C_v) of cost $c(e)$ in G_i. We include this edge (C_u, C_v) of cost $c(e)$ in the DCS solution.

(ii) Consider pairs of components in $\mathcal{P}$ between which the paths in T_i consist of more than a single edge. Let C_x, C_y be such a pair and let the path between them in T_i be $(C_x, C_1), (C_1, C_2), \ldots, (C_q, C_y)$. For each edge in this path, there is a corresponding edge in T^* from which this edge is derived. Let these edges in T^* be $(v_x, v_{1,in}), (v_{1,out}, v_{2,in}), \ldots, (v_{q,out}, v_y)$ respectively. Note

---

[1] The case when $|\mathcal{C}_i|$ is odd can be treated similarly by adding the dummy node included in Step 7 to the set $\mathcal{C}_i$.

that for $1 \leq j \leq q$, both $v_{j,in}$ and $v_{j,out}$ are in the component C_j but they may be two distinct vertices. However there is a zero-cost edge in G_i joining them. Thus, using these edges, we can form a path $P_{x,y}$ in G_i between every such pair C_x, C_y in $\mathcal{P}$. We then take the modulo-two sum of all these paths[2] and include this subgraph in the DCS solution.

It is routine to verify that the set of edges identified above form a valid solution to the DCS problem and that the cost of this solution is at most OPT_b. This completes the proof of Lemma 6.3.6. □

Combining Lemmas 6.3.4, 6.3.5 and 6.3.6 gives Theorem 6.1.2.

6.4 Extensions and an Application

In this section, we show how to extend Theorem 6.1.2 proved in Section 6.3 and prove Theorems 6.1.3, 6.1.4, and 6.1.5. Then we outline a proof of Theorem 6.1.8.

Extension to generalized Steiner forests

We now provide a proof of Theorem 6.1.3 We sketch the algorithm for the generalized Steiner case. The algorithm follows the same outline as the algorithm in Section 6.3. We highlight only the differences here. The main difference is that every vertex does not start out in a connected component that needs to be merged with all the others. Instead we introduce the notion of *active components*, namely components that separate a site-pair. Thus, to begin with, each node in a site-pair is in an active component by itself.

In constructing the auxiliary graph G_i, we add zero cost edges as before between nodes in the same active component. We also add all the other graph edges between the nodes of V. This permits the use of paths in the graph G to connect up the active components in a DCS solution.

We also modify the subgraph constructed at each iteration of the algorithm as follows: In an iteration with q active components, we add $\lfloor \frac{q}{3} \rfloor$ dummy nodes with zero cost edges to all the active components in the auxiliary graph G_i. Even in this case, at least $\frac{2q}{3}$ of the active components will be paired among themselves in the DCS solution found. This pairing reduces the number of active components by $\frac{q}{3}$, ensuring that the number of iterations is still $O(\log(\frac{k}{b}))$ where k is the number of nodes that are in site-pairs. The performance guarantee of the algorithm thus remains unaffected by the modification.

We use this modification in the auxiliary graph G_i to prove the existence of a feasible solution to the DCS problem of value at most OPT_b (Lemma 6.3.6) in each iteration. In the proof of Lemma 6.3.6, we start with the optimal generalized Steiner forest and contract the active components to

---

[2] An edge is in the modulo-two sum only if it occurs in an odd number of the paths that are being summed.

single nodes. This may leave a forest where the active components are in several trees. Note that each of these trees must contain at least two active components. We can apply Claim 6.3.2 only to trees with an even number of active components. However the total number of trees with an odd number of active components is at most $\lfloor \frac{q}{3} \rfloor$ since each such tree has at least two active components. To apply Claim 6.3.2 to such a tree, we consider the active components in the tree along with a dummy node in G_i. Note that we have have included a sufficient number of dummy nodes to find such a pairing for all such trees. Thus we can derive Lemma 6.3.6 in this case as well.

In the last iteration, when the number of active components falls to $O(b)$, we run the approximation algorithm for minimum-cost generalized Steiner forests [2] using the contracted active components as sites. The maximum degree of any node in this forest is $O(b)$ and the cost is at most twice OPT_b using the performance guarantee in [2]. Also, the final subgraph output has no active component and so the final subgraph contains a generalized Steiner forest for the site-pairs. The proof of the performance guarantee proceeds exactly as before. Note that the number of iterations is now $O(\log(\frac{k}{b}))$ where k is the number of nodes in site-pairs. The rest of the proof is the same as before and this proves Theorem 6.1.3.

We can generalize our algorithm to handle even more general network-design problems using the framework of proper functions introduced in Section 3.3. We formulate the network-design problem as before as that of finding a cut-cover for a cut-family defined by a proper function f. Recall that the *terminals* are the nodes v of G such that $f(\{v\}) = 1$. We can generalize Theorem 6.1.3 for cut-cover problems defined by proper functions f as follows.

Theorem 6.4.1 *There is a polynomial-time algorithm that, given an undirected graph G with nonnegative costs on its edges, a proper function f defined on the node subsets of G, and a degree bound b, constructs a cut-cover for the family of cuts defined by f in which the maximum degree of any node is $O(b \log \frac{k}{b})$ and the cost of the cover is $O(\log \frac{k}{b})$ times that of the minimum-cost cut cover for f with degree at most b. Here k represents the number of terminals defined by f.*

The proof of the above theorem is identical to that of Theorem 6.1.3 with the only difference that active components are now defined directly by the function f. It follows from the properties of proper functions [60] that each active component must be connected with at least one other active component in any solution. Hence adding $\lfloor \frac{q}{3} \rfloor$ dummy nodes as before in an iteration with q active components is sufficient to derive a solution to the subgraph problem set up at each iteration using the optimal solution. The rest of the analysis is identical and proves Theorem 6.4.1.

Arbitrary degree constraints

We now describe how we can handle arbitrary degree specifications on the nodes and still find networks that approximately obey these degree constraints and are of nearly minimum cost under this restriction.

The key observation is that we can solve in polynomial time a more general degree constrained minimum-cost subgraph problem than that described in Fact 6.3.1. In this general version of the f-factor problem [109], we are given upper and lower bounds on the value of a degree of each node in the network and required to find a subgraph obeying these degree restrictions and has minimum possible cost under these restrictions. To use these results, we form an auxiliary graph as in Section 6.3 and instead of looking for a subgraph with degree at most $2b$ at the nodes of V, we set the upper bound on the degree of a vertex $v \in V$ as $2d(v)$. Using the optimal solution to the d-constrained problem, we can derive a solution to this problem that we set up during this iteration of cost at most the optimal. Since we can solve this more general problem in polynomial-time as well [38, 109, 138], we can obtain a solution of cost at most the optimal. The rest of the performance guarantee is the same as before. To derive the $O(\log n)$ guarantee on the degree and the cost, we run this algorithm to completion until all the components have been joined in a spanning tree. This proved Theorem 6.1.4.

The extension of this result to more general one-connected networks like Steiner trees, generalized Steiner trees and proper function cut-covers is immediate using techniques from the previous subsection along with those described above.

Approximating weighted-degree spanning trees

In this section, we describe how our techniques can be applied to approximate minimum weighted-degree spanning trees. The algorithm for finding such an approximate solution follows the same framework as the algorithm in Section 6.3.

The algorithm works in $O(\log n)$ phases where n is the number of nodes in the input graph. As before, to begin with, the solution subgraph is empty and each node is in a connected component. At each phase, we add edges between the components thus reducing the number of connected components in the current solution by a factor of half. We set up a minimum cost DCS problem to achieve this in Section 6.3. However for the special case of spanning trees, this reduces to a simple minimum-cost $2b$-matching algorithm in which we match the cluster nodes in $\mathcal{C}$ to nodes in V at minimum cost (using inter-cluster edges only). The added restriction is that we want the degree of any node in V in this matching to be at most $2b$.

In the case of weighted spanning trees, let B^* denote the minimum weighted-degree of a spanning tree for the graph. At each iteration we form the same auxiliary graph as before. Then we draw an analogy with a scheduling problem considered by Lenstra, Shmoys and Tardos [104] as follows. The cluster nodes are analogous to "jobs" while the nodes of V are analogous to "machines". The cost of an inter-cluster edge between a cluster C and a node v is analogous to the processing time of job C on machine v. Under this analogy, considering the nodes of V as unrelated parallel machines and minimizing the makespan of the jobs in $\mathcal{C}$ corresponds to matching each cluster (or job) node with a node v of V (or machine) such that the maximum total weight of matched edges incident on any node v (makespan of machine v) is minimized. From the optimal solution with weighted-degree B^*, it is easy to derive a schedule of makespan at most B^*. Lenstra, Shmoys and Tardos describe a 2-approximation for the makespan problem. Thus using their algorithm as a subroutine in implementing an iteration, we can reduce the number of components by a half while incurring a cost of at most B^* in terms of weighted-degree at any node. This proves Theorem 6.1.5.

The extension to finding minimum-cost spanning trees while keeping the weighted-degree low follows by using a generalization of the result of Lenstra, Shmoys and Tardos by Shmoys and Tardos [140]. In this generalization, Shmoys and Tardos show that if there exists a schedule of cost C and makespan T, one can obtain in polynomial time a schedule of cost at most C and makespan at most $2T$. Using this method as a subroutine in each phase of our algorithm, Theorem 6.1.6 follows.

Better approximations for minimum-degree generalized Steiner forests

We now show how to apply Theorem 6.1.3 to approximate the problem of finding a minimum-degree generalized Steiner forest in an unweighted graph, and thus prove Theorem 6.1.8. We are given an undirected graph G with a set of site-pairs and we wish to find a generalized Steiner forest for the site-pairs whose degree is nearly minimum.

To do this, we introduce costs between every pair of nodes in G as follows: If there is an edge between u and v we assign the edge (u, v) unit cost. Otherwise, we assign the edge a very high cost C (say, some number much larger than n^2 where n is the number of nodes in G). Now we use the algorithm in Theorem 6.1.3 on the graph G with these weights and with the same set of site-pairs. However we run this algorithm for all values of the degree bound from n down to 1. Let b^* be the minimum-degree of any generalized Steiner forest for the site-pairs in the input graph. Then as long as the value of the degree bound is at least b^*, the algorithm in Theorem 6.1.3 outputs a forest of degree $O(b^* \log \frac{k}{b^*})$ where k is the number of nodes in site-pairs. Furthermore, the cost of this forest is $O(n \log \frac{k}{b^*})$ since there is a generalized Steiner forest (namely, the one with minimum-degree b^*) whose cost is at most $n-1$. Since edges not in G have excessive cost (exceeding n^2), the generalized Steiner forest provided by the algorithm must use only edges in G. We run the algorithm with decreasing values of the degree bound as long as we obtain a solution of cost less than n^2, i.e., using only the original edges. We output the solution consisting of original edges with the minimum degree among all the runs of our algorithm. From the discussion above, we know that when the degree bound is set to b^* we will obtain a generalized Steiner forest using edges of G of degree $O(b^* \log \frac{k}{b^*})$. This proves Theorem 6.1.8. Note that we may speed things up by doing a binary search on the value of the degree bound to converge on the smallest value such that the output solution uses only original edges in G.

6.5 The Node-weighted Case

In this section, we present the algorithm for node-weighted networks and prove Theorem 6.1.7.

The algorithm for node-weighted networks

The algorithm maintains a set S of nodes and a set F of edges. Initially S contains all the terminals and F is empty. During the course of the algorithm, the connected components of the graph (S, F) are node-disjoint trees containing all the terminals. Define a connected component of (S, F) to be *active* if it contains at least one terminal and does not contain at least one terminal.

As in Section 6.3, the algorithm works in $O(\log(\frac{k}{b}))$ iterations. However, in each iteration, instead of a DCS problem we follow the approach in Chapter 4: we run a greedy algorithm to choose a subgraph of small degree and node-cost whose addition to the current solution reduces the number of connected components of (S, F) by a constant factor. As before, we use OPT_b to denote the minimum cost of any b-bounded Steiner tree of the input graph. In the algorithm, we use the term "shortest path" between two vertices to mean the minimum node-cost of any path between the vertices excluding the costs of these vertices.

The Algorithm

```
Initialize $S$ to be the set of terminals, and $F$ to be the empty set.
Repeat while there are active components in $(S, F)$
  Let $\mathcal{C}$ be the set of active components of $(S, F)$. Let $\mathcal{C} = \{C_1 \ldots, C_q\}$ where $q = |\mathcal{C}|$. Let
  $G' := G$.
  Repeat while the number of active components in $(S, F)$ is greater than $\frac{11q}{12}$ (If $q = O(b)$,
  then we run this iteration until there are no active components.)
    Let $V'$ be the nodes in $G'$.
    Construct an auxiliary complete bipartite graph $H$ from $G'$ as follows.
      The node set of $H$ is $V' \cup \mathcal{C}$. The cost of a node $v$ is zero if $v \in S$, otherwise it is as
      specified in the input graph $G$.
      The edge $(v, C_j)$ in $H$ is assigned cost $c(v, C_j)$ equal to that of a shortest path from $v$
      to any node in $C_j$ in $G'$.
```

```
9           Find a node $v \in V'$ in the graph $H$ minimizing the ratio
                $\min_{2 \le r \le b+1} \min_{\{C_1,\ldots,C_r\} \subseteq \mathcal{C}} \frac{c(v) + \sum_{j=1}^{r} c(v, C_j)}{r}$
10          Let $v$ be the node and $C_1, \ldots, C_r$ be the components in $\mathcal{C}$ chosen in the previous step of
            the algorithm. Let $P_1, \ldots, P_r$ be a set of shortest paths in $G'$ connecting $v$ to $C_1, \ldots, C_r$
            respectively. Add an acyclic subgraph of $\cup_{j=1}^{r} P_j$ to the current solution $(S, F)$ so as to
            merge $C_1, C_2, \ldots, C_r$ into one. Update $\mathcal{C}$.
11          For every node $v \in V'$, if the degree of the node using edges added in this iteration is
            between $2b$ and $3b$, delete this node from $G'$. In the last iteration, i.e., when $q = O(b)$,
            we ignore this step and delete no nodes from $G'$.
12        The number of active components in $(S, F)$ is now at most $\frac{11q}{12}$. We go on to the next
          iteration.
13      Output $(S, F)$ as the solution.
```

It is easy to implement Step 9 as shown for the algorithm in Chapter 4. We repeat the argument here for completeness. For each node v, define the *quotient cost* of v to be the minimum value of the ratio in Step 9 achieved by this node. To find the quotient cost of v, we can order the components in $\mathcal{C}$ as $C_1, C_2, C_3, \ldots$ in nondecreasing order of $c(v, C_j)$. In computing the quotient cost of v, it is sufficient to consider subsets of $\mathcal{C}$ of the form $\{C_1, C_2, \ldots, C_j\}$ where $2 \le j \le b + 1$. Thus the quotient cost for a given vertex can be computed in polynomial time; by computing the quotient cost for each vertex, we can determine the minimum quotient cost, and thus carry out Step 9 in polynomial time.

Performance guarantee

We prove the performance guarantee using a series of lemmas. As in Section 6.3, the following two lemmas are immediate.

Lemma 6.5.1 *The number of iterations of the algorithm is* $O(\log(\frac{k}{b}))$ *where* k *is the number of terminals.*

Lemma 6.5.2 *At any iteration of the algorithm, the degree of any node due to edges added in this iteration is* $O(b)$.

Now we turn to the cost of the subgraph added in an iteration.

Lemma 6.5.3 *At any iteration of the algorithm except the last, the cost of the set of nodes added to the solution in this iteration is* $O(OPT_b)$.

To bound the cost of nodes added in the last iteration, we notice that the number of active components is $O(b)$ at the beginning of this iteration. For this iteration, our algorithm reduces to that in Chapter 4 for node-weighted Steiner trees with the active components as terminals. Hence using Theorem 4.1.1 with the number of terminals being $O(b)$, we arrive at the following bound on the cost of nodes added in this iteration.

Lemma 6.5.4 *The cost of the set of nodes added to the solution in the last iteration is* $O(OPT_b \log b)$*.*

We prove Lemma 6.5.3 in the remainder of this section.

Proof of Lemma 6.5.3

We prove an averaging lemma and use this in conjunction with a potential function argument due to Leighton and Rao [103] to prove Lemma 6.5.3. First we prove a simple lemma bounding the number of nodes deleted from G' in each iteration.

Fix an iteration and let q denote the number of active components at the beginning of the iteration. In the beginning of this iteration, we initialize the graph $G' := G$. During the course of this iteration, we may delete nodes from G' in Step 11.

Claim 6.5.5 *At any iteration of the algorithm, the number of nodes deleted from* G' *is at most* $\frac{q}{2b}$*.*

Proof: Assume for a contradiction that more than $\frac{q}{2b}$ nodes were deleted from G' during this iteration. By the condition for the deletion of a node in Step 11, each of the deleted nodes has degree at least $2b$. Hence the sum of the degrees of all the deleted nodes is at least q. The idea of the proof is to show that a large fraction of this degree contributes to merging the q active components in this iteration, using the fact that the subgraph added in this iteration is acyclic. This allows us to derive a contradiction to the fact that the inner loop terminates when a small factor (i.e., $\frac{q}{12}$) of the active components are merged using edges added in this iteration.

We construct an auxiliary graph $H(i)$ for this iteration i using edges added in the current iteration i. Let the active components in the beginning of this iteration i be $C_1, C_2, \ldots, C_q$. The node set of $H(i)$ is the set of clusters at the beginning of this iteration. Note that this includes all of $C_1, C_2, \ldots C_q$ as well as all the remaining inactive clusters. The set of edges in $H(i)$ is the set of all edges added in Step 10 in this iteration. These edges form an acyclic subgraph on the set of cluster nodes. Moreover, by the running of the algorithm, all the the leaves in this subgraph are active components. A leaf in $H(i)$ is thus one of the active components $C_1, \ldots, C_q$.

The set of nodes that are deleted from G' during the course of this iteration is exactly the set of nodes in V whose degree in $H(i)$ is at least $2b$. We call such nodes *deleted* nodes. By assumption, there are more than $\frac{q}{2b}$ such nodes. We show that in this case, a large fraction of the q active components in the beginning of this iteration would have merged on adding the edges in $H(i)$.

Denote the trees in $H(i)$ that contain deleted nodes by $H_1, \ldots, H_z$. Let d_j denote the number of deleted nodes in the tree H_j. By our assumption, we have $\sum_{j=1}^{z} d_j > \frac{q}{2b}$. By definition, the degree of each deleted node is at least $2b$ and so the degrees of all the deleted nodes in H_j sum to at least $2bd_j$. The set of edges between these d_j nodes contribute at most $2d_j$ to this degree since H_j is acyclic. Thus, the number of leaves in H_j is at least $2bd_j(1 - \frac{1}{b}) \geq bd_j$ since $b \geq 2$. Note that all these leaves are active components from the set $\{C_1, \ldots, C_q\}$ and the tree H_j merges these components together. Thus the decrease in the number of active components due to addition of the edges in H_j to the solution is at least $bd_j - 1 \geq \frac{bd_j}{2}$ since $bd_j \geq 2$.

Summing over all the trees H_j, the decrease in the number of active components due to addition of edges in this iteration is

$$\sum_{j=1}^{z} \frac{bd_j}{2} = \frac{b}{2} \cdot \sum_{j=1}^{z} d_j > \frac{b}{2} \cdot \frac{q}{2b} \geq \frac{q}{4}$$

But this iteration terminates when more than $\frac{q}{12}$ of the active components have been merged, giving a contradiction. Thus the total number of nodes deleted in G' at any iteration beginning with q active components is at most $\frac{q}{2b}$. $\Box$

Spider decompositions

We employ the notion of spider decompositions introduced in Chapter 4 in showing that the each node chosen in Step 9 has small quotient cost with respect to the optimal solution. We recall a few definitions we introduced in Chapter 4.

Definitions: A *spider* is a tree with at most one node of degree greater than two. A *center* of a spider is a node from which there are node-disjoint paths (called *legs*) to the leaves of the spider. Note that if a spider has at least three leaves, its center is unique. A *foot* of a spider is a leaf, or, if the spider has at least three leaves, the spider's center. Thus every spider contains disjoint paths from its center to all of its feet. A *nontrivial spider* is one with at least two leaves. Let G be a graph, and let M be a subset of its nodes. A *spider decomposition* of M in G is a set of node-disjoint nontrivial spiders in G such that the union of the feet of the spiders in the decomposition contains M. We restate Theorem 4.3.1 below.

Theorem 6.5.6 *Let G be a connected graph, and let M be a subset of its nodes such that $|M| \geq 2$. Then G contains a spider decomposition of M.*

An averaging lemma

Let v be a node chosen in Step 9 in this iteration and let C denote the cost of the subgraph added subsequently in Step 10. Let this subgraph merge r trees. We prove the following claim.

Claim 6.5.7

$$r \geq \frac{5Cq}{12OPT_b} \tag{6.1}$$

Proof: Let T^* be a minimum-cost b-bounded Steiner tree. Let $C_1, \ldots, C_p$ be the active components when the node v was chosen. Let $T^*(v)$ be the graph obtained from T^* by contracting each C_j to a supernode of zero cost. $T^*(v)$ is connected and contains all supernodes.

Delete all edges incident to nodes in $V - V'$ in $T^*(v)$. The number of nodes in $V - V'$ is at most $\frac{q}{2b}$ by Claim 6.5.5. Since any node has degree at most b in T^*, the number of edges deleted from $T^*(v)$ is at most $\frac{q}{2}$. The deletion of these edges breaks $T^*(v)$ into many subtrees. But at least $p - \frac{q}{2}$ of the supernodes are in subtrees with at least two or more supernodes. Since $p \geq \frac{11q}{12}$, at least $\frac{5q}{12}$ supernodes are in such trees.

Proposition 6.5.8 *Let M denote the subset of supernodes that are in subtrees with two or more supernodes. Then we have*

$$|M| \geq \frac{5q}{12}$$

We apply Theorem 6.5.6 to each subtree of $T^*(v)$ with at least two supernodes to obtain a spider decomposition of M. We now compare the quotient cost of the node v chosen by the algorithm with that of each spider in the decomposition. To do this however, the center of each spider in the decomposition must be a real node (not a supernode) and the number of legs of each spider must be at most $b+1$.

We can further partition a spider centered at a supernode into many nontrivial spiders each centered at a real node contained in this supernode such that the union of their feet contain the feet of the original spiders and the number of legs in each resulting spider is bounded. To do this, first consider all the real nodes in the central supernode with at least two legs of the spider incident on them. Each such real node can be made the center of a nontrivial spider with all the legs incident on it as the legs of this spider. Since the degree of any real node in T^* is at most b, the number of legs of any such spider is at most b as well. If no legs remain after removing such legs from the spider, then we may add the central supernode as a foot of one of the spiders found. Otherwise, each of the remaining legs is incident to a different real node in the supernode. We may arbitrarily partition these legs into classes with at most b legs in each class and use any real node by which a leg in this class is incident to the supernode as a center. Note that in the auxiliary graph H, the distance of the center to the foot of any leg is at most the cost of the nodes in the leg since all nodes inside the central supernode have been assigned zero cost. Thus the distance of this center to the feet of all the legs in this class in the auxiliary graph H is at most the cost of the legs. These classes with their centers form spiders that we may use to compare with the subgraph chosen by the algorithm. As before, the central supernode can be included as a foot of one of these spiders. Let the centers of the resulting spider decomposition be the set of real nodes $v_1, \ldots, v_t$.

For a spider with only two legs, i.e., a path, pick any node in the path as its center. Let $\ell_1, \ldots, \ell_t$ denote the number of nodes of M in each of these spiders respectively. Since every spider in the decomposition is nontrivial and is derived as above, each ℓ_j is at least two and at most $b+1$. Moreover, a spider with center v_j induces a subset of the current active components, namely the ℓ_j components whose supernodes belong to this spider. Let the cost of the spider centered at v_j (i.e., cost of v_j plus the sum of the node-costs of the paths from v_j to the ℓ_j components) be $Cost_j$. Then the quotient cost of v_j in the auxiliary graph H constructed in this loop is at most $\frac{Cost_j}{\ell_j}$.

Since the algorithm chooses a vertex of minimum quotient cost in H, for each spider in the decomposition we have $\frac{Cost_j}{\ell_j} \geq \frac{C}{r}$. Summing over all the spiders in the cover yields

$$\sum_{j=1}^{t} Cost_j \geq \frac{C}{r} \sum_{j=1}^{t} \ell_j.$$

Since the union of the feet of the spiders contains M, $\sum_{j=1}^{t} \ell_j \geq |M| \geq \frac{5q}{12}$ by Proposition 6.5.8. Also $\sum_{j=1}^{t} Cost_j$ is exactly the cost of the spider decomposition, which is at most that of $T^*(v)$ which in turn is at most OPT_b. Substituting in the above equation and simplifying yields the Claim. □

A potential function argument

Now we are ready to complete the proof of Lemma 6.5.3. Fix an iteration i and let the set of nodes chosen in Step 9 of the algorithm in this iteration be $v_1, \ldots v_f$ in the order in which they were chosen.

Let ϕ_j denote the number of active components in the solution after choosing vertex v_j in this iteration. Thus, for instance, $\phi_0 = q$, the number of active components at the beginning of this iteration in (S, F) and $\phi_f \leq \frac{11q}{12}$. Let the number of trees merged using vertex v_j be r_j. Then we have

$$\phi_j = \phi_{j-1} - (r_j - 1) \tag{6.2}$$

Let C_j denote the cost of the subgraph added by the algorithm in the step when vertex v_j was chosen. Then by Claim 6.5.7, we have

$$r_j \geq \frac{5C_j q}{12 OPT_b} \geq \frac{5C_j \phi_{j-1}}{12 OPT_b} \tag{6.3}$$

We now use an analysis technique due to Leighton and Rao [103] to complete the proof as in Chapter 4. Substituting Equation (6.3) into (6.2) and simplifying using $r_j \geq 2$ gives

$$\phi_j \leq \phi_{j-1}(1 - \frac{5C_j}{24OPT_b}) \tag{6.4}$$

Unraveling (6.4), we obtain

$$\phi_{f-1} \leq \phi_0 \prod_{j=1}^{f-1}(1 - \frac{5C_r}{24OPT_b})$$

Taking natural logarithms on both sides and simplifying using the approximation $\ln(1+x) \leq x$, we obtain

$$\frac{24OPT_b}{5}\ln(\frac{\phi_0}{\phi_{f-1}}) \geq \sum_{j=1}^{f-1} C_j$$

Note that $\phi_0 = q$ and $\phi_{f-1} > \frac{11q}{12}$ and so we have

$$\sum_{j=1}^{f-1} C_j < 5OPT_b \ln\frac{12}{11} = O(OPT_b) \tag{6.5}$$

Note that the cost of the nodes added in this iteration is exactly the sum $\sum_{j=1}^{f} C_j$.

To complete the proof, we bound the cost of the subgraph associated with v_f, the last node chosen in this iteration. Using Claim 6.5.7 and noting that $r_f \leq q$ we have

$$C_f \leq \frac{12OPT_b}{5}$$

Using the above equation and (6.5), we have that the cost of the set of nodes added in this iteration is

$$\sum_{j=1}^{f} C_j = O(OPT_b)$$

This proves Lemma 6.5.3.

Lemmas 6.5.1, 6.5.2, 6.5.3 and 6.5.4 together prove the performance guarantee in Theorem 6.1.7.

The extension of Theorem 6.1.7 to generalized Steiner forests and proper function cut-covers is immediate and follows the same outline as above. The only additional issue is that in the proof of the performance guarantee of the cost of the subgraph added in each iteration, the optimal solution is a forest instead of a single tree. Using the fact that each tree in the forest must contain at least two active components, we can infer that this forest contains at least as many edges as half the number of active components. This observation is sufficient to prove a modified version of Claim 6.5.7 with slightly worse constants. The rest of the analysis is identical.

6.6 Algorithms under Triangle Inequality

In this section, we introduce the short-cutting technique used in proving Theorem 6.1.9. First, we review the short-cutting method introduced in [137].

Short-cutting for TSP

Rosenkrantz, Stearns and Lewis [137] introduced a simple short-cutting technique for obtaining a TSP tour of a graph with edge-costs obeying the triangle inequality. We review this method briefly. First note that the cost of a TSP tour is at least as much as that of a minimum spanning tree of the graph. We start with an MST and duplicate each edge in it. The resulting multigraph has an Euler tour since each vertex has an even degree. The Euler tour can be converted to a TSP tour by starting at an arbitrary vertex, traversing the nodes in the order of the Euler tour and including each vertex in the TSP tour at the first instance it is visited in this traversal. This corresponds to adding short-cut edges over paths composed of already visited nodes in the Euler tour. By triangle inequality, the cost of the TSP tour so obtained is no more than the cost of the Euler tour which is in turn at most twice the cost of the MST.

Deleting a single edge (say the costliest) from this tour gives a spanning tree of maximum degree two, and of total cost at most twice the MST. However, it falls short in two ways in addressing our problem in Theorem 6.1.9. First, the approximation bound cannot be improved for higher values of the degree bound b, and secondly, since the short-cuts may represent arbitrarily long paths, no bound can be proven on the bottleneck cost of the tree so obtained. We provide a short-cutting method that rectifies these shortcomings. We present below the approximation algorithm referred to in Theorem 6.1.9.

The algorithm for an approximate b-MST

Input: An undirected graph with edge-costs satisfying the triangle inequality and a degree bound $b \geq 3$.

Output: A spanning tree in which the maximum degree of any node is b.

1 Find an MST of the given graph [4, 98, 127]. Root the spanning tree at any node r of degree at least two.

2 Partition the edges of the tree into "claws", namely, sets of edges going from every internal node to its children in the tree. Sort the edges in every claw in the order of non-decreasing cost. Thus if a typical internal node v has children $v_1, v_2, \ldots, v_d$ then for $1 \leq i < d$, the costs obey $c(v, v_i) \leq c(v, v_{i+1})$.

3 We short-cut each claw locally by replacing edges from the internal node to its first $(d -$

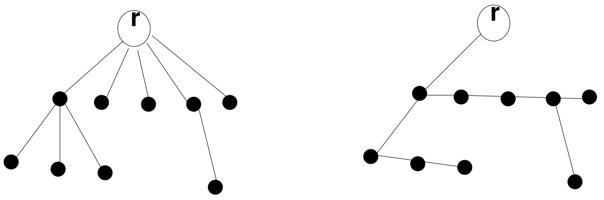


Figure 6.2: An example of the short-cutting algorithm for finding a spanning tree of small cost, degree and bottleneck cost. The edges to the children from every internal node are drawn from left to right in nondecreasing order of cost. The tree on the right results on applying th algorithm with $b = 3$.

$b + 2)$ children except the very first child, with edges between consecutive children. Thus if an internal node v has d children in the tree and $d > (b - 1)$, then replace the edges $(v, v_2), (v, v_3), \ldots, (v, v_{d-b+2})$ in the tree with the set of edges $(v_1, v_2), (v_2, v_3), \ldots, (v_{d-b+1}, v_{d-b+2})$.

4 Output the resulting spanning tree.

An example of the running of the algorithm is shown in Figure 6.2.

Proof of the b-MST Theorem

We prove that the above algorithm outputs a spanning tree with the properties in Theorem 6.1.9.

It is easy to see that the short-cutting above produces a spanning tree. Since the local short-cutting at a vertex reduces the degree of a vertex from $d + 1$ by removing exactly $d - b + 1$ edges incident on it, the degree of any node in the output spanning tree is at most b as required.

The following facts are immediate.

Fact 6.6.1 *([20]) Any minimum spanning tree of an undirected graph is also a minimum bottleneck spanning tree, namely, one in which the cost of the maximum edge is minimum.*

Observation 6.6.2 *For any $b \geq 2$, the cost of any b-bounded spanning tree is at least as much as that of the MST.*

Observation 6.6.3 *For any $b \geq 2$, the bottleneck cost of any b-bounded spanning tree is at least as much as that of the minimum bottleneck cost spanning tree.*

Let r denote the node chosen as the root by the algorithm. The algorithm partitions the edges of the MST into claws. Let T denote the tree output by the algorithm. For any set E' of edges, let $c(E')$ denote the sum of the costs of all the edges in E'. We have the following relations.

$$c(MST) = \sum_{v \ : \ v \text{ is not a leaf of the MST}} c(claw(v)).$$

For the solution T, we have

$$c(T) = \sum_{v \;:\; v \text{ is not a leaf of the MST}} [c(claw(v)) - \sum_{i=2}^{d-b+2} c(v, v_i) + \sum_{i=2}^{d-b+2} c(v_{i-1}, v_i)].$$

By triangle inequality on the costs c, we have

$$c(v_{i-1}, v_i) \leq c(v_{i-1}, v) + c(v, v_i) \leq 2c(v, v_i)$$

The last inequality follows from the way we ordered the edges in each claw in non-decreasing order of costs. Putting the above three equations together, we get the following bound on the cost of the output tree T.

$$\frac{c(T)}{c(MST)} \leq (2 - \frac{(b-2)}{(n-1)}).$$

Combining this with Observation 6.6.2 above proves the bound on the cost of the tree output by the algorithm.

We now complete proof by proving the bound of two on the bottleneck cost. By the Fact 6.6.1, the MST found in the first step of the algorithm is also an optimum bottleneck spanning tree. Since each short-cut used in forming the output tree T is made of at most two edges, the bottleneck cost of T is at most twice that of the MST. Since the bottleneck cost of any b-bounded spanning tree is at least as much as that of the bottleneck spanning tree by Observation 6.6.3, the resulting tree has bottleneck cost at most twice optimum.

The algorithm for TSPs

The algorithm used in proving Theorem 6.1.10 is based on a recursive short-cutting procedure that uses edges from the cube of the minimum spanning tree of the graph. Let T be a MST of the input graph. We denote by T^3 the set of edges between pairs of nodes of T that are at a distance of at most three from each other in T.

Lemma 6.6.4 *Let G be an edge-weighted graph with the weights obeying the triangle inequality. Let T be a spanning tree of G rooted at r and of weight $w(T)$ and let c be one of the children of r in T. Then there exist a Hamiltonian path in G from r to c of cost at most $2w(T) - w(r,c)$ in which all the edges are in T^3. Here $w(r,c)$ denoted the weight of the edge (r,c).*

Proof: The proof of the lemma is by induction on the number of nodes in G. The basis when there are two nodes in G is trivial.

To prove the induction step, assume that T is a tree on n nodes. Let the children of the root r in T be $r_1(=c), \ldots, r_k$. Consider the subtree T_i rooted at r_i. By the induction hypothesis, there

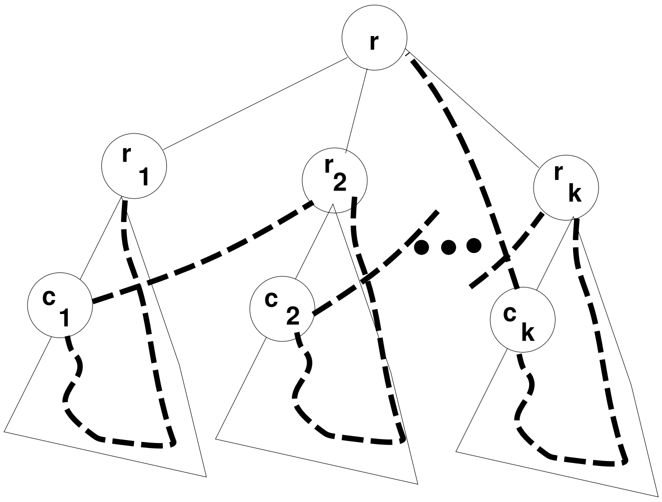


Figure 6.3: The recursive short-cutting procedure for obtaining a Hamiltonian path from the root of a tree to one of its children using only edges from the cube of the tree and having cost at most twice the cost of the tree.

exists Hamiltonian paths from each r_i to c_i of weight at most $2w(T_i) - w(r_i, c_i)$. We can now string these paths together to obtain a Hamiltonian path from $r_1 (= c)$ to r as follows. Starting from r_1, we follow the path in T_1 to c_1, followed by the edge (c_1, r_2) in T^3, continuing this way in each of $T_2, ...T_k$ and finally add the edge (r_k, r) to end in r (See Figure 6.3). The extra edges we used are (c_i, r_{i+1}) $\forall i$; all these extra edges are in T^3. Furthermore the weight of the Hamiltonian path we identified is

$$\sum_{i=1}^{k}(2w(T_i) - w(c_i, r_i)) + \sum_{i=1}^{k-1} w(c_i, r_{i+1}) + w(r, r_k).$$

By the triangle inequality,

$$w(c_i, r_{i+1}) \leq w(c_i, r_i) + w(r_i, r) + w(r, r_{i+1}).$$

Substituting above and simplifying using $w(T) = \sum_i (w(T_i) + w(r, r_i))$ finally yields that the weight of the Hamiltonian path identified is at most

$$2w(T) - w(r, r_1)$$

as desired. □

The above lemma gives a natural recursive algorithm for finding a TSP with the bounds as in Theorem 6.1.10. We can apply the lemma to a rooted MST of the input graph and find a Hamiltonian path from the root to one of its children. Adding the edge from the root to this child to the Hamiltonian path gives a TSP of cost at most twice that of the MST. The bound on the bottleneck cost follows from Observation 6.6.3 and the property that we used only edges from the cube of the MST in forming the tour. This completes the proof of Theorem 6.1.10.

Higher connectivities

Now we are ready to prove Theorem 6.1.11 in its full generality. The starting point is the TSP tour obtained in Theorem 6.1.10. Let c^* and b^* denote the cost of an MST and the optimum bottleneck cost of a spanning tree of the input graph. By Theorem 6.1.10, we can obtain a TSP tour T of cost $c(T)$ and bottleneck cost $b(T)$ such that $c(T) \leq 2c^*$ and $b(T) \leq 3b^*$. Let the vertices in this tour be numbered $v_1, v_2, \ldots, v_n$. Now, we add extra edges to this cycle as follows: Add edges joining any two vertices that are within a distance $\lceil \frac{k}{2} \rceil$ of each other in the cycle (distance in terms of number of edges: reachable using a path of at most k edges). It is not hard to see that this graph is k-vertex-connected. The degree of every node in this graph is at most $k+1$. Since each short-cut employed replaces a path of at most $\frac{k}{2}$ edges, the bottleneck cost goes up by this factor. This proves that the bottleneck cost of this subgraph is within $3 \cdot \lceil \frac{k}{2} \rceil$ of optimal.

The total cost of the graph obtained this way is $\frac{k(k+2)}{4}$ times that of the TSP tour T that we started with. This in turn is at most $\frac{k(k+2)c^*}{2}$. However, we can apply an approximate min-max relation between a MST and a packing of cuts in the graph that is derived in [2, 60] in proving a better performance guarantee of $k+2$ for the total cost. In particular, if OPT_k denotes the cost of a minimum k-connected subgraph, we show that $OPT_k \geq \frac{kc^*}{2}$. This would prove that the cost of the k-connected subgraph output by our algorithm is at most $(k+2)OPT_k$ as claimed in 6.1.11 [3]

It remains to prove that $OPT_k \geq \frac{kc^*}{2}$. We do this in the remainder of this section. For this we need to recall a Theorem from Section 2.5 in Chapter 2. Recall that a (fractional) packing of cuts is a family of cuts $\Gamma(W_1), \Gamma(W_2), \ldots, \Gamma(W_k)$, together with a rational *weight* for each cut with the following property: for each edge (u, v) of weight $w(u, v)$, the sum of the weights of all the cuts in this collection containing the edge is at most $w(u, v)$. The following is a restatement of Theorem 2.5.1 and is a consequence of the results in [2, 60].

Theorem 6.6.5 *Given an undirected graph with edge-weights, the minimum-weight spanning tree has weight at most twice the value of a maximum packing of cuts.*

Note that any k-connected spanning subgraph must have at least k edges crossing any cut since this subgraph has k disjoint connections between every pair of vertices. Thus we have the following lemma.

Lemma 6.6.6 *The weight of any k-connected subgraph is at least k times as much as the value of a maximum packing of cuts.*

Applying the above lemma to the optimum k-connected subgraph of cost OPT_k and combining with Theorem 2.5.1 above we conclude that that $OPT_k \geq \frac{kc^*}{2}$.

---

[3] For k odd, taking the ceiling gives that the total cost of the subgraph is at most $\frac{(k+1)(k+3)}{4}$ times that of the TSP tour. Using the similar argument, the final bound for this case is $\frac{(k+1)(k+3)}{k}$. We chose to present the cleaner though not completely correct bound in the statement of Theorem 6.1.11.

6.7 Hardness Results

In this section we sketch related NP-completeness results and some simple hardness results that fall out of them. Then, we examine approximation of Steiner trees under multiple objectives.

NP-completeness

In this subsection, we prove NP-completeness of the problems we considered. Though the minimum spanning tree and the minimum bottleneck spanning tree problems are polynomially solvable (in fact coincidentally by the same algorithm!), the degree bounded versions of these problems that we consider in this paper are NP-hard. In these versions, the maximum degree of any node in the spanning tree is required to be bounded by an integer b where $2 \leq b \leq n^{1-\epsilon}$ for any constant $\epsilon > 0$.

Theorem 6.7.1 *The following problems are NP-complete:*

- *(b-bounded minimum spanning tree) Given an undirected graph on n nodes and a degree bound b such that $2 \leq b \leq n^{1-\epsilon}$ for some constant $\epsilon > 0$, find a minimum-cost spanning tree of G in which the maximum degree of any node is at most b.*
- *(b-bounded bottleneck spanning tree) Given an undirected graph on n nodes and a degree bound b such that $2 \leq b \leq n^{1-\epsilon}$ for some constant $\epsilon > 0$, find a spanning tree of G in which the maximum degree of any node is at most b and in which the maximum cost of any edge is minimum.*

Proof Sketch: First of all, it is easy to see that case $b = 2$ is hard for both the above problems, by a reduction from the Hamiltonian path problem. Given a graph in which we wish to find a Hamiltonian path, we can assign unit edge cost to all the edges in the graph and a cost of two to edges not in the graph. The resulting edge-weighted graph has a 2-bounded MST of cost $(n-1)$ if and only if the original graph is Hamiltonian. Furthermore this edge-weighted graph has a 2-bounded bottleneck spanning tree of cost one if and only if the original graph is Hamiltonian. Note that the edge-costs we assigned satisfy the triangle inequality, so the problem is hard even in the case of a graph with such a special cost function.

Now we can handle the more general case of $b = n^{1-\epsilon}$ for any fixed $\epsilon > 0$. To do this, we transform the problem of determining if there is a Hamiltonian path from vertex u to vertex v in an undirected graph to this problem as follows: For each of u and v, add $h_b - 1$ additional nodes to the graph $u_1, \ldots, u_{h_b-1}, v_1, \ldots, v_{h_b-1}$. Add edges (u, u_i) and (v, v_i) for all $1 \leq i \leq (h_b - 1)$. For every other vertex w in the graph, add $h_b - 2$ additional vertices $w_1, \ldots, w_{h_b-2}$ and edges (w, w_i) for $1 \leq i \leq (h_b - 2)$. The resulting graph has the property that any of the newly added vertices x_i has a single incident edge in the graph. Hence in any spanning tree, x_i is adjacent to the corresponding vertex x. Thus this graph has a h_b-bounded spanning tree if and only if the original graph has a Hamiltonian path from u to v. Note that the number of vertices in the resulting graph is about $h_b \cdot n$. We require $h_b = (n \cdot h_b)^\epsilon$. Solving gives $h_b = n^{\frac{1-\epsilon}{\epsilon}}$. Thus the transformation is polynomial-size for any fixed $\epsilon > 0$.

□

Hardness of approximation

Theorem 6.7.2 *Let G be an undirected graph with nonnegative costs on the edges and let R be a fixed rational number such that $R \geq 1$. The following problems are NP-hard.*

- *Given an integer $b \geq 2$ (the bound on the degree), find a spanning tree of G whose maximum degree is at most b and whose total cost is at most R times that of a minimum-cost degree-b-bounded spanning tree of G.*
- *Given an integer $b \geq 2$ (the bound on the degree), find a spanning tree of G whose maximum degree is at most b and for which the maximum cost of any edge in the tree (sometimes referred to as the bottleneck cost) is at most R times the minimum bottleneck cost of any degree-b-bounded spanning tree of G.*
- *Let the costs on the edges obey the triangle inequality and let $R < 2$. Given an integer $b \geq 2$ (the bound on the degree), find a spanning tree of G whose maximum degree is at most b and for which the bottleneck cost of the tree is at most R times the minimum bottleneck cost of any degree-b-bounded spanning tree of G.*

Proof Sketch: When the cost function does not obey the triangle inequality, the proof of the Theorem 6.7.1 can be turned into a non-approximability proof for any ratio $R \geq 1$. To do this, we can simply set the cost of any edge not in the original graph to $R(n-1)$ for the b-bounded MST problem and to R for the b-bounded bottleneck spanning tree problem. The cost of a b-bounded MST in the resulting graph is exactly $(n-1)$ if the original graph is Hamiltonian and is at least $R(n-1)$ if it is not. Similarly, the maximum cost of any edge in a b-bounded spanning tree in this graph is at most one if the original graph is Hamiltonian and is at least R otherwise. Thus any approximation algorithm with performance ratio less than R would be able to recognize if the original graph is Hamiltonian. This proves the first two parts of Theorem 6.7.2.

To prove the last part, we use the cost assignment as in the proof of Theorem 6.7.1 above that obeys the triangle inequality. Under this assignment, the maximum cost of any edge in any b-bounded spanning tree of the resulting graph is at most one if the original graph is Hamiltonian and is at least two otherwise. Hence an approximation algorithm with performance ratio less than two in this case would be able to recognize Hamiltonian graphs. This completes the proof of Theorem 6.7.2. □

Triangle inequality: One-connected extensions

In this section we prove the following results on approximating both bottleneck cost and total cost of Steiner trees with respect to the respective optima.

Proposition 6.7.3 *We can construct an undirected graph G with nonnegative costs on the edges such that the costs obey the triangle inequality and identify a subset of the nodes of G as terminals, such that the following is true: For any tree T, let $c(T)$ and $b(T)$ denote the total edge-cost and the bottleneck cost of T respectively. Let c^* and b^* represent the minimum total cost and minimum bottleneck cost of any Steiner tree of G spanning the terminals. Then for every Steiner tree T spanning the terminals,*

$$\frac{c(T) \cdot b(T)}{c^* \cdot b^*} \geq \sqrt{n}$$

where n is the number of nodes in G.

Proof: The example illustrating the Proposition is described below. The graph has n nodes and consists of two terminal vertices t_1 and t_2 with an edge of cost $\sqrt{n}$ between them. There is also a path consisting of all the remaining $n-2$ nodes between t_1 and t_2. Each of the $(n-1)$ edges in this path has unit cost. All the other edge costs in the graph are as implied by these edges and the triangle-inequality. The minimum-cost Steiner tree T_1 connecting t_1 and t_2 consists of the single edge (t_1, t_2) of cost $\sqrt{n}$. The optimal bottleneck Steiner tree T_2 consists of the path between t_1 and t_2 of the remaining $n-2$ vertices. This Steiner tree has bottleneck cost of one. Now the bottleneck cost of the tree T_1 is $\sqrt{n}$ which is $\sqrt{n}$ times the optimal value. Similarly, the tree T_2 has total cost $(n-1)$ which is about $\sqrt{n}$ times the minimum possible. It is easy to check that any other path connecting t_1 and t_2 also exhibits the trade-off implied in Proposition 6.7.3. □

We also prove a theorem to show that the bound above is algorithmically achievable within a constant factor.

Theorem 6.7.4 *Let G be an undirected graph with nonnegative costs on the edges such that the costs obey the triangle inequality, and let a subset of the nodes of G be specified as terminals. For any tree T, let $c(T)$ and $b(T)$ denote the total edge-cost and the bottleneck cost of T respectively. Let c^* and b^* represent the minimum total cost and minimum bottleneck cost of any Steiner tree of G spanning the terminals. Let R_s denote the ratio of approximation achievable for the minimum-cost Steiner tree problem by a polynomial-time* *heuristic. Then we can find in polynomial-time a Steiner tree T' spanning the terminals such that*

$$\frac{c(T') \cdot b(T')}{c^* \cdot b^*} \leq R_s\sqrt{n} \tag{6.6}$$

where n is the number of nodes in G.

Currently, the best known values of R_s are $\frac{16}{9}$ using the techniques of Zelikovsky [164] and Berman and Ramaiyer [15].

Proof: The algorithm for computing the Steiner tree which simultaneously minimizes the product of the total cost and the bottleneck cost given in the ratio (6.6) is simply this: Compute an approximately minimum-cost Steiner tree using any of the known algorithms [164, 15]. Let this heuristic guarantee a solution which comes to within R_s of the minimum-cost Steiner tree. Call this tree T_1. Let $c(T_1)$ and $b(T_1)$ denote the total edge cost and the bottleneck cost of the tree. Next we construct a Steiner tree whose bottleneck cost is optimal. To do this, we simply add edges in the graph in the increasing order of cost and stop when all the terminals are in a single connected component. Clearly any spanning tree of this component has minimum bottleneck cost. Call this tree T_2. Choose one of the two trees T_1 or T_2 that yields a lower product for the total edge cost and the bottleneck cost. Call this tree T'.

Case 1: $T' = T_1$. Since the bottleneck cost of a tree can be no more than the total edge cost of the tree, we have $c(T_1) \cdot b(T_1) \leq R_s c^* \cdot R_s c^*$. Thus

$$\frac{c(T_1) \cdot b(T_1)}{c^* \cdot b^*} \leq \frac{R_s^2 c^*}{b^*}$$

Case 2: $T' = T_2$. Since the total cost of any tree in no more than n times the bottleneck cost, we have that $c(T_2) \cdot b(T_2) \leq n \cdot b^* \cdot b^*$ and so

$$\frac{c(T_2) \cdot b(T_2)}{c^* \cdot b^*} \leq \frac{b^* \cdot n}{c^*}$$

Since we selected the tree which has a lesser product of total and bottleneck costs it follows that the worst case performance ratio is less than $R_s\sqrt{n}$ since for any $\tau \geq 1$,

$$\min(\frac{n}{\tau}, R_s \cdot \tau) \leq R_s\sqrt{n}.$$

□

This chapter completes our study of network-design problems with minimization objectives. In the next chapter, we turn to a maximization problem that arises in finding trees with a large number of pendant vertices.

Chapter 7

Maximum-leaf Spanning Trees

7.1 Introduction

In this chapter we turn to a maximization problem arising in the design of spanning trees. Given a simple, undirected graph $G = (V, E)$, suppose we wish to find a spanning tree of G with the maximum number of leaves. This problem finds applications in communication networks, circuit layouts and in other graph-theoretic problems [145]. An interesting application is mentioned in [124]: In [32], E. W. Dijkstra studied the problem of self-stabilizing a set of processors in the presence of distributed control and proposed a solution based on mutual exclusion. A variant of this solution [149] seeks a spanning tree of a graph such that the product of the degrees of all the nodes in the tree is minimum. We can use a spanning tree with the maximum number of leaves as a heuristic solution (with no guarantees) for the latter problem.

The maximum-leaf spanning tree problem is NP-complete [53]. In this chapter, we use the simple technique of local optimization to provide the first approximation algorithms for this problem. These algorithms perform a series of local-improvement steps until a local optimum is reached and output a locally optimal solution. This notion of applying local-improvement heuristics to hard optimization problems was around [30] long before the invention of NP-completeness [83]. It has been applied heuristically to solve a variety of NP-hard problems in combinatorial optimization [121]. As we mentioned in Section 1.10, a recent breakthrough in this area was the work of Fürer and Raghavachari [50], who applied this technique to the minimum-degree spanning tree problem. They showed a version of a local optimization algorithm runs in polynomial time to produce near-optimal solutions. The work in this chapter is inspired by that of Fürer and Raghavachari. We demonstrate that local optimization can be used to design polynomial-time approximation algorithms for the maximum-leaf spanning tree problem. Our algorithms output spanning trees with as many leaves as at least a constant fraction of the maximum. The results in this chapter were obtained jointly with Hsueh-I Lu, and appeared in [111].

Our algorithms perform local changes that increase the number of leaves in the resulting spanning tree. For the problem at hand, there is a natural notion of such a local change, namely, swapping a tree edge for a non-tree edge. This notion can be extended to include swapping many tree edges for an equal number of non-tree edges in a single local change operation. Our algorithms perform such local changes that increase the number of leaves until no improvement in the solution is possible and output a locally optimal solution as the approximate solution.

By varying the limit on the number of tree edges that can participate in a single local change operation, we derive a series of approximation algorithms. Increasing this limit corresponds to allowing for more powerful local improvement steps. Thus, the higher this limit is, the higher is the running time of the corresponding algorithm. A local change operation involving at most k tree edges is termed a *k-change*. Thus k-changes are costlier to implement than $(k-1)$-changes. Intuitively we expect that an algorithm using k-changes would perform better than an algorithm using only $(k-1)$-changes. In this chapter, we corroborate this intuition and prove that local improvement algorithms that use 1-changes and 2-changes have performance ratios of 5 and 3 respectively. The following are the main results in this chapter.

Theorem 7.1.1 *There is an $O(n^4)$ algorithm using 1-changes for finding a spanning tree of any simple, undirected graph G on n nodes such that the number of leaves in this spanning tree is at least as many as a fifth of the number of leaves in any spanning tree of G.*

Theorem 7.1.2 *There is an $O(n^7)$ algorithm using 2-changes for finding a spanning tree of any simple, undirected graph G with n nodes such that the number of leaves in this spanning tree is at least as many as a third of the number of leaves in any spanning tree of G.*

We prove that the running time of the approximation algorithm using k-changes increases with k as follows.

Lemma 7.1.3 *The running time of the approximation algorithm using k-changes on a graph with n nodes is $O(n^{3k+1})$.*

The pattern exhibited in Theorems 7.1.1 and 7.1.2 lead us to conjecture that there is a trade-off between the running times and the performance ratios exhibited in the series of algorithms we have defined.

Conjecture 7.1.4 *The performance of the approximation algorithm using k-changes is strictly better than that of the algorithm using $(k-1)$-changes.*

In the next section, we survey related work. Then we describe the approximation algorithms after introducing relevant definitions. In the following section, we illustrate our proof technique by providing a rough bound on the performance guarantee. In the subsequent sections, we prove Theorems 7.1.1 and 7.1.2. We close this chapter with some remarks on related problems.

7.2 Related Work

The technique of local optimization has been applied to provide heuristic solutions for a variety of hard problems in combinatorial optimization. Chapter 19 of [121] surveys a few applications of this technique. Some notable examples are its applications to the graph partitioning [84] and the Traveling Salesperson problems [102]. A number of complexity results relating to the paradigm of local optimality and the time complexity of computing a locally optimal solution are presented in [80, 120]. In this work, we are interested in the application of local search techniques to design efficient approximation algorithms, namely, those that run in polynomial time and provide provably good solutions with values close to that of the optimum.

Fürer and Raghavachari [50] showed such an application of local search to the problem of computing a spanning tree of a given graph whose maximum degree is minimum. This is termed the minimum-degree spanning tree problem and is NP-complete [53]. Fürer and Raghavachari show that local optimization can be applied to produce spanning trees and even Steiner trees of nearly minimum degree. We showed in Chapter 5 how to generalize their technique to obtain approximation algorithms for a variety of minimum-degree network-design problems. In this chapter, we add to the list of problems that are amenable to good approximate solutions by local-search methods.

Previous work on finding spanning trees with many leaves have focused on graphs with minimum degree at least k for some fixed $k \geq 3$. For such graphs, good lower bounds on the number of leaves achievable in a spanning tree are derived in [67, 92, 124, 145]. These lower bounds are typically proved algorithmically by constructing a spanning tree with the desired number of leaves. Thus these algorithms can be interpreted as approximation algorithms for the maximum-leaf spanning tree problem for special classes of graphs in which the minimum degree of a node is at least k. However, to the best of our knowledge, no previous approximation algorithms for this problem are known in the general case that we consider in this chapter. The best-known lower bound for the number of leaves achievable in a spanning tree of a n-node graph with minimum degree k is $(1 - b\ln k/k)n$ where b is any constant exceeding 2.5 and k is sufficiently large [92]. The best lower bounds for the cases $k = 3$ and 4 are $n/4$ and $2n/5$ respectively [92].

There has also been work on polynomial-time solutions to the problem of determining if a given graph has a spanning tree with at least k leaves for fixed k. The first such algorithm was due to Fellows and Langston [42]. The running time of their algorithm was improved by Bodlaender [17].

7.3 Definitions

In this section, we introduce some definitions that we use throughout this chapter. The following definitions are relevant to any spanning tree of the input simple graph G. Henceforth we shall use the term degree of a node to refer to its degree in the tree under consideration. Thus leaves are just nodes of degree one. A node is *internal* if it is not a leaf. We call a node of degree greater than two a *high-degree node*. A leaf is *special* if it is adjacent in the tree to a high-degree node. All other leaves are termed *normal*. Thus normal leaves are exactly the leaves that are adjacent to nodes of degree two in the tree. A maximal tree path containing only nodes of degree two is called a *2-path*. Its length is the number of nodes in it. A 2-path is *short* if its length is one; otherwise it is *long*. We shall sometimes refer to an edge $e = (u, v)$ as an edge *between* u and v. We say a node v is *enclosed* by a non-tree edge e if the node v is not an endpoint of e and is in the unique tree cycle closed by e.

We use $\lambda(T)$ to denote the number of leaves in a spanning tree T. We omit the T where it is understood. We use n_i to denote the number of nodes of degree i. We define $N_i = \sum_{j \geq i} n_j$, the number of nodes with degree at least i. Thus for instance, N_3 is the number of high-degree nodes, and $N_1 = n$, the total number of nodes in the graph. The number of short and long 2-paths are denoted by P_ℓ and P_s, respectively. Let λ_s and λ_n be the number of special and normal leaves, respectively.

Contracting all the 2-paths in a tree to single edges and bounding the number of internal nodes (each resulting internal node has degree at least 3) in this contracted tree by the number of leaves, we arrive at the following observation.

Observation 7.3.1 *The number of high-degree nodes in a tree is at most the number of leaves in the tree minus two. In other words,* $N_3 \leq \lambda - 2$.

By discarding special leaves along with their incident edges from a tree, contracting all 2-paths to single edges and bounding the number of contracted edges in the resulting tree by the number of nodes, we have the following.

$$P_\ell + P_s \leq N_3 + \lambda_n - 1. \tag{7.1}$$

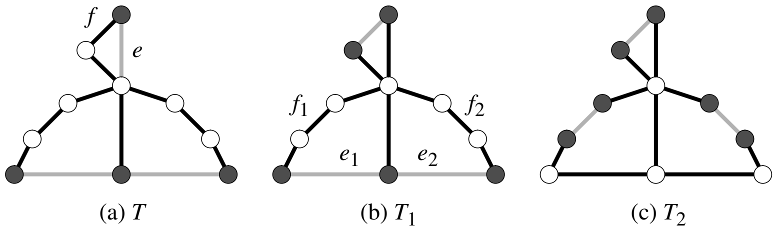
From Equation 7.1 and Observation 7.3.1, we have the following.

Observation 7.3.2 *The number of 2-paths in any tree is at most twice the number of leaves in the tree, i.e.,*

$$P_\ell + P_s \leq 2\lambda - 3.$$

7.4 The Algorithm

In this section, we formally define k-changes and describe the approximation algorithms.



(a) T (b) T_1 (c) T_2

Figure 7.1: Examples of local improvements. In (a), the tree T is shown in dark lines and the non-tree edges are dotted. The change $(e; f)$ is a 1-improvement to T resulting in T_1 with one more leaf as shown in (b). T_1 does not admit any 1-improvement but admits the 2-improvement $(e_1, e_2; f_1, f_2)$ to give T_2 shown in (c).

Edge Changes, Improvements and LOTs

Our algorithm starts with an arbitrary spanning tree T of the given graph G. Let $e = (u, v)$ be an edge in $G - T$ and let f be an edge in the unique path connecting u and v in T. We call making e a tree edge and f a non-tree edge an *(edge) change* $(e; f)$ with respect to T. The spanning tree obtained by applying change $(e; f)$ to T is denoted by $T(e; f)$. If $T(e; f)$ has more leaves than T, then we call the change $(e; f)$ an *improvement*.

We can generalize the notion of performing single edge swaps to allow for multiple edge swaps in a single change step. Suppose $e_0 = e$, $f_0 = f$, and $T_0 = T$. Let T_{i+1} denote $T_i(e_i; f_i)$, where $(e_i; f_i)$ is a change with respect to T_i for $0 \leq i < j$. We use $T(e_0, e_1, \ldots, e_{j-1}; f_0, f_1, \ldots, f_{j-1})$ to denote T_j. We define

$$(e_0, e_1, \ldots, e_{j-1}; f_0, f_1, \ldots, f_{j-1})$$

to be a *k-change* with respect to T, for any $k \geq j$. A k-change with respect to a spanning tree T is a *k-improvement* if the resulting spanning tree on applying the k-change has more leaves than T.

Note that a k-change is stronger than k 1-changes. It is a batched version of any sequence of k 1-changes. In Figure 7.1, we illustrate a spanning tree T that admits a 1-improvement. On applying the 1-improvement we get a spanning tree T_1 that does not admit a 1-improvement but has a 2-improvement.

A *k-locally optimal tree* (k-LOT) of a given graph G is a spanning tree that does not admit any k-improvement. By definition, an i-LOT is also a j-LOT for any $j \leq i$. Since a spanning tree has exactly $n - 1$ edges, an optimal spanning tree with the maximum number of leaves is just a $(n - 1)$-LOT.

```
Input  A simple undirected graph $G$ and an integer $k \geq 1$.
Output  A $k$-LOT $T$ of $G$.
  1. Find a spanning tree $T$ of $G$.
  2. While there exists a $k$-improvement with respect to $T$ do
       Let $T$ be the tree obtained by applying the improvement to $T$.
```

Figure 7.2: The approximation algorithm.

The Algorithm and Termination

Our idea is simply to use k-LOTs to approximate the maximum-leaf spanning tree for small values of k.

The approximation algorithm is shown in Figure 7.2. We start with an arbitrary spanning tree and keep applying k-improvements to the current spanning tree until this spanning tree is a k-LOT. Clearly, the second step dominates the time complexity. Whether a spanning tree is a k-LOT can be decided in polynomial time for constant k, since a spanning tree cannot have more than $\binom{\binom{n}{2}-n+1}{k} \times \binom{n-1}{k}$ k-changes. The algorithm itself has no more than $n-3$ iterations because the number of leaves should be at least 2 to begin with and at most $n-1$ in the end and this number increases with every iteration. Therefore, the running time to obtain a k-LOT is $O(n^{3k+1})$ which is a polynomial in n for constant k. This proves Lemma 7.1.3.

Our major concern is the performance guarantee of this approximation algorithm. In other words, how much less can the number of leaves in a k-LOT be than that of the optimum? In the following sections we shall prove that the performance ratios of 1-LOTs and 2-LOTs are 5 and 3, respectively. However, we believe that 2-LOTs have a better performance ratio than what we prove.

7.5 Proof of a Rough Bound

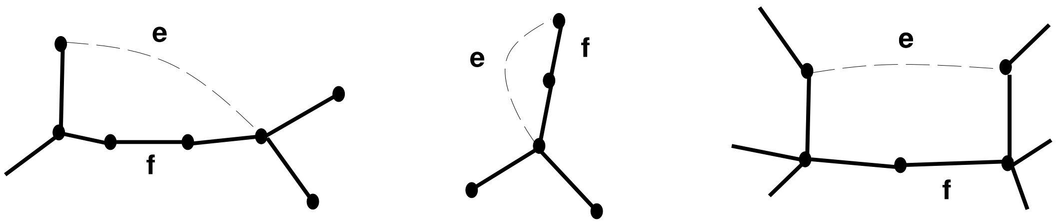
In this section, we illustrate our proof method by proving a rough bound on the performance ratio of 1-LOTs. We begin by examining a few basic properties that are useful in proving the bounds.

Basic Properties

By definition, a 1-LOT has the following properties (See Figure 7.3).

P1 In a 1-LOT T, any cycle formed by adding a non-tree edge between a leaf and an internal node does not enclose two adjacent nodes of degree two in T.

P2 In a 1-LOT T, there is no non-tree edge between a normal leaf and an internal node.



violates P1 violates P2 violates P3

Figure 7.3: Illustrating properties P1, P2 and P3 for 1-LOTs. In the above figure, dark edges represent tree edges and the dashed edge is a non-tree edge violating one of the properties P1, P2 or P3. In all the above figures, a 1-improvement $(e; f)$ can be identified.

P3 In a 1-LOT T, any cycle formed by adding a non-tree edge between two internal nodes does not enclose a node of degree two in T.

The following lemma shows that P1, P2 and P3 together are sufficient conditions for 1-optimality.

Lemma 7.5.1 *A tree is a 1-LOT of a given graph if it satisfies P1, P2 and P3.*

Proof We prove the contrapositive and show that if a spanning tree is not a 1-LOT then it violates one of P1, P2 or P3. If a spanning tree is not a 1-LOT then there exists a 1-improvement for this spanning tree. It is easy to see that any non-tree edge incident on two leaves cannot be involved in any 1-improvement. So any non-tree edge involved in a 1-improvement must be an edge between an internal node and a leaf or an edge between two internal nodes. In the first case, the tree violates either property P1 or P2 and in the second case, it violates P3. □

Any 2-path in a spanning tree partitions the remaining nodes of the tree into two pieces on its removal. We refer to the sets of nodes representing these two pieces as the two *sides* of the 2-path. Using Lemma 7.5.1, it is easy to derive the following.

Lemma 7.5.2 *In a 1-LOT T, only the first two nodes on a long 2-path, counting from a particular side, have non-tree edges to nodes in that side. Furthermore, all non-tree edges between nodes in two sides of a long 2-path must have both their endpoints as leaves in T.*

Lemma 7.5.3 *In a 1-LOT, there is no non-tree edge between two nodes in the same long 2-path.*

The Strategy: Proving a Rough Bound

In this subsection, we illustrate the basic strategy used in proving Theorem 7.1.1 by proving the following weaker version of it. Recall that $\lambda(T)$ denotes the number of leaves in a tree T.

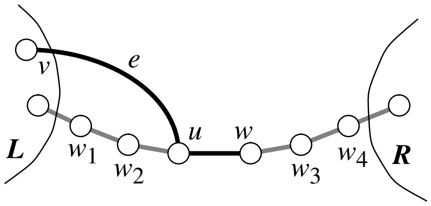


Figure 7.4: Proof of Lemma 7.5.5 by contradiction: The dark edges are edges of T' and the dashed edges are edges of T. If there are five leaves of T' in path $\mathcal{P}$, then the middle node must be connected in T' to one of the two sides via a non-tree edge e as shown above. This edge e violates the assumption that the tree T is a 1-LOT by Lemma 7.5.2.

Theorem 7.5.4 *Let G be a given simple undirected graph. If T^* is a maximum-leaf spanning tree of G and T is a 1-LOT of G, then $\lambda(T^*) \leq 10 \cdot \lambda(T)$.*

Consider any spanning tree T' of G. We bound the number of leaves of T' that are (a) leaves in T, (b) high-degree nodes in T, and (c) nodes in 2-paths in T independently. Note that these three categories account for every node in G. The total number of leaves in T is $\lambda(T)$ by definition. Also, the total number of high-degree nodes in T is at most $\lambda(T)$ by Observation 7.3.1. Also, we can bound the *number* of 2-paths in T by $2\lambda(T)$ using Observation 7.3.2. Below, we bound the number of leaves of T' in each 2-path of T by four. Thus the total number of leaves in T' is at most $\lambda(T) + \lambda(T) + 2\lambda(T) \cdot 4 = 10\lambda(T)$. Since this bound holds for any tree T' it also holds for T^* in particular and our theorem is proved. It remains to bound the number of leaves of T' in any 2-path in T by four. We do this next.

Lemma 7.5.5 *For any spanning tree T' of G and any 2-path $\mathcal{P}$ of a 1-LOT T, the number of leaves of T' in $\mathcal{P}$ is at most four.*

Proof The proof is by contradiction. Assume for a contradiction that there are no less than five leaves of T' in $\mathcal{P}$. Let five of them be w_1, w_2, w, w_3, and w_4, where w is the middle one along $\mathcal{P}$, and w_1 and w_2 are on the same side of w (See Figure 7.4). Note that by Lemma 7.5.3 there are no non-tree edges between nodes in $\mathcal{P}$. Let v be the closest node to w in T' that is not in $\mathcal{P}$. Let the last edge in the path from w to v in T' be $e = (u, v)$ where $u \in \mathcal{P}$. Since w is the middle leaf in T' along $\mathcal{P}$, the cycle closed by e contains at least two nodes of degree two, i.e., either w_1 and w_2 or w_3 and w_4. Without loss of generality, let these two nodes be w_1 and w_2. Since they are both in $\mathcal{P}$, the nodes between them in $\mathcal{P}$, if any, also have degree two in T. This implies that the cycle closed by e encloses at least two adjacent nodes of degree two which contradicts the fact that T is a 1-LOT by Lemma 7.5.2. □

7.6 Performance Guarantee of 1-LOTs

In this section we prove the following restatement of Theorem 7.1.1.

Theorem 7.6.1 *Let G_0 be a given simple undirected graph. If T_0^* is a maximum-leaf spanning tree of G_0 and T_0 is a 1-LOT of G_0, then $\lambda(T_0^*) \leq 5 \cdot \lambda(T_0)$.*

Augmentation

In order to prove Theorem 7.6.1, we augment G_0 together with T_0 as follows. For every normal leaf in T_0 adjacent to a short 2-path, we subdivide the tree edge incident on the normal leaf by inserting a *pseudo-node* of degree 2 in this edge. We have the following lemma.

Lemma 7.6.2 *Let G and T be the augmented versions of G_0 and T_0 respectively. If T_0 is a 1-LOT of G_0 then T is a 1-LOT of G. Furthermore, $\lambda(T) = \lambda(T_0) \leq \lambda(T_0^*) \leq \lambda(T^*)$ where T_0^* is a maximum-leaf spanning tree of G_0 and T^* is a maximum-leaf spanning tree of G.*

Proof We prove that T is a 1-LOT of G by contradiction. Assume for the sake of a contradiction that T is not a 1-LOT of G. Then, there exists a 1-improvement involving addition of a non-tree edge e. Since T_0 is a 1-LOT of G_0, a pseudo-node v must be enclosed by e in T. Note that no non-tree edge is incident on v, so e must be incident on the normal leaf adjacent to v. Since T_0 is a 1-LOT and one of the endpoints of e is a normal leaf, by Property P2, we infer that the other endpoint of e should be a leaf as well. It is easy to see that no non-tree edge incident on two leaves can participate in a 1-improvement. This contradicts the fact that edge e yields a 1-improvement. Thus T is a 1-LOT.

Now we consider the relations among $\lambda(T)$, $\lambda(T_0)$, $\lambda(T_0^*)$, and $\lambda(T^*)$. The equality is trivial since augmentation does not change the number of leaves. The first inequality follows from the definition of T_0^*. The last inequality is proved by constructing a spanning tree of G with at least $\lambda(T_0^*)$ leaves. To achieve this goal, we start with T_0^*. For each pseudo-node v to be inserted, if the unique edge whose subdivision resulted in v is in T_0^*, we subdivide this edge and insert the pseudo-node v in T_0^*. Otherwise, let the neighbors of v in G be v_1 and v_2. To augment T_0^* to include v, we simply attach v to v_1. In either case, we finally obtain a spanning tree of G with at least $\lambda(T_0^*)$ leaves. □

Partition into Regions

For the sake of proof, based on a 1-LOT T, we partition the nodes in G depending on whether they occur in a long 2-path in T or not. We further partition the nodes of G that are not in any long 2-path in T into *regions*. These regions are exactly the connected components of T obtained on removing the nodes in long 2-paths from T. A region is *special* if it contains at least one special leaf of T; otherwise it is *normal*. Let us use S and R to stand for the number of special and normal regions, respectively. The degree of a region is the number of long 2-paths incident on it. Note that as a result of augmentation, each normal leaf is a single-node normal region by itself. So a normal region is either a single leaf or a set of high-degree nodes interconnected via short 2-paths. We term these two kinds of normal regions *external* and *internal*, respectively.

Lemma 7.6.3 *For an augmented 1-LOT T, we have the following relations.*

$$P_\ell = S + R - 1 \tag{7.2}$$

$$N_3 + \lambda_n = P_\ell + P_s + 1 \tag{7.3}$$

Proof Equality (7.2) is trivial. If we regard T as a tree of regions by contracting long 2-paths to single edges, the number of nodes (i.e. regions) is $S + R$ and the number of edges (i.e. long 2-path) is P_ℓ. The equality holds because the the number of edges of a tree is equal to the number of nodes minus one.

Inequality (7.3) is not difficult either. By contracting both short and long 2-paths to single edges and discarding special leaves, we obtain a tree whose nodes are high-degree nodes and normal leaves. Since the number of nodes in this tree is exactly the number of edges plus one, we have

$$N_3 + \lambda_n = P_\ell + P_s + 1,$$

which proves (7.3). □

The following lemma is a consequence of Lemma 7.5.1.

Lemma 7.6.4 *Let u be a node in a region $\mathcal{R}$ of T. There is no non-tree edge between u and v if either v is in an internal normal region other than $\mathcal{R}$, or v is in a long 2-path that $\mathcal{R}$ is not adjacent to.*

Proof We prove by contradiction. In both cases, the node v is an internal node and the non-tree edge (u, v) encloses all the nodes of some long 2-path of T. Hence this edge violates property P1 contradicting the local optimality of T. □

Accounting for Leaves in Long 2-paths

The key observation which is useful in proving the tighter bound in Theorem 7.6.1 is that not every long 2-path in T contributes four leaves in any spanning tree as implied by Lemma 7.5.5. The following lemma is a refined version of Lemma 7.5.5 and is proved similarly.

Lemma 7.6.5 *Let T' be a spanning tree of G and $\mathcal{P}$ be a long 2-path of a 1-LOT T of G. Suppose $\mathcal{P}$ is incident on two regions $\mathcal{L}$ and $\mathcal{R}$ in T. The number of nodes in $\mathcal{P}$ which are leaves in T' is at most*

1. *four, if $\mathcal{L}$ and $\mathcal{R}$ are both special,*
2. *three, if only one of $\mathcal{L}$ and $\mathcal{R}$ is special, and*
3. *two, if $\mathcal{L}$ and $\mathcal{R}$ are both normal.*

Proof The case in which both regions are special follows directly from Lemma 7.5.5. The proofs for the other two cases are similar to that of Lemma 7.5.5.

In the third case, both of $\mathcal{L}$ and $\mathcal{R}$ are normal. Assume that there are no less than three nodes in $\mathcal{P}$ that are leaves of T'. Let us say, three of them are w_1, w and w_2, where w is the middle one, and $\mathcal{L}$ and w_1 are on the same side of w. Note that by Lemma 7.5.3 there are no non-tree edges between nodes in $\mathcal{P}$. Let v be the closest node to w in T' that is not in $\mathcal{P}$. Let the last edge in the path from w to v in T' be $e = (u, v)$ where $u \in \mathcal{P}$. Since w is the middle leaf in T' along $\mathcal{P}$, the cycle closed by e contains at least one node whose degree is two in T, i.e. either w_1 or w_2. Without loss of generality, let this node be w_1. Since T is a 1-LOT, by properties P1, P2 and P3, v must be a special leaf of T. This means that v cannot be in $\mathcal{L}$ or $\mathcal{R}$. Thus v is in some other region. By Lemma 7.6.4, this is a contradiction and the third case is proved.

Let us consider the second case. Assume without loss of generality that $\mathcal{L}$ is special and $\mathcal{R}$ is normal. Assume that there are no less than four nodes of $\mathcal{P}$ that are leaves in T'. Let us say, these four nodes are w_1, w_2, w, and w_3, where $\mathcal{R}$ and w_3 are on the same side of w, and $\mathcal{L}$, w_1, and w_2 are on the other side of w. For the same reason as above, w must be connected to a node v not in $\mathcal{P}$ through a non-tree connection $e = (u, v)$ where u is a node in $\mathcal{P}$. As before, v must be a special leaf in T. By Lemma 7.6.4, v must be in either in $\mathcal{R}$ or in $\mathcal{L}$. But by Property P2, v cannot be in $\mathcal{R}$ since this region is normal, so v must be in $\mathcal{L}$. Now the cycle closed by e contains at least two adjacent nodes of degree two in T. This is a violation of property P1 and hence contradicts the fact that T is a 1-LOT by Lemma 7.5.1. This prove the second case. □

We can think of the above lemma as providing an upper bound on the number of leaves in T' in any long 2-path $\mathcal{P}$ of T in the following way. Assume that a normal region contributes a charge of 1 and a special region a charge of 2 to the upper bound for any incident long 2-path. Then the upper bound on the number of leaves in T' in any long 2-path $\mathcal{P}$ is exactly the sum of the charge contributions of both its adjoining regions.

Invalid Regions and Better Bounding

Even the careful counting of the leaves of T' in long 2-paths in the above lemma is not sufficient to prove Theorem 7.1.1. We need some additional results. For example, consider an internal normal region. To connect any node in such a region in T' to any region outside, we must use one of the long 2-paths incident on it since there are no non-tree edges going from this region to any other region. This implies that this normal region cannot charge a contribution of 1 to *all* its incident long 2-paths. To capture this we define the notion of an invalid region.

Definition: A region of T is *invalid* in T' if it is either an internal normal region or it is a special region and every special leaf in this region is also a leaf in T'. We now identify long 2-paths for which the bounds on the number of leaves of T' is more stringent than those stated in Lemma 7.6.5. Let $\mathcal{P}$ be a long 2-path of T. We call $\mathcal{P}$ *deficient* in T' if the number of nodes in $\mathcal{P}$ that are leaves in T' is at most two less than the upper bound stated in Lemma 7.6.5.

Lemma 7.6.6 *Let $k(\geq 2)$ be the number of regions of T that are invalid in T'. Then there are at least $k-1$ distinct long 2-paths of T that are deficient in T'.*

Proof We start with a few definitions.

Definition: Let $\mathcal{R}$ be a region that is invalid in T'. Define the *core* of $\mathcal{R}$ to be the set of nodes in $\mathcal{R}$ excluding special leaves in it if any. Thus, for instance, the core of an invalid normal region is the whole region. The *adjacency* of a region $\mathcal{R}$ denoted $adj(\mathcal{R})$ is defined as follows.

$$adj(\mathcal{R}) = \bigcup_{\text{long 2-paths } \mathcal{P} \text{ incident on } \mathcal{R}} \{\text{the node in } \mathcal{P} \text{ closest to } \mathcal{R} \text{ in } T\}.$$

We now define a notion of neighborhood for each invalid region $\mathcal{R}$.

$$nbrhood(\mathcal{R}) = core(\mathcal{R}) \cup adj(\mathcal{R}).$$

We use the properties of a locally optimal tree to investigate the structure of the edges in the graph going out of the neighborhood of an invalid region. Define

$$\Gamma(\mathcal{R}) = \{\text{all edges in } G \text{ between } nbrhood(\mathcal{R}) \text{ and } V - (\mathcal{R} \cup adj(\mathcal{R}))\}.$$

For a normal invalid region, the only edges out of the region in the graph must be in the above edge set. For special invalid regions, since the special leaves of T are also leaves in T', a path in T' from the core of the region to the outside must use edges of $\Gamma(\mathcal{R})$. This is summarized below.

Observation 7.6.7 *Let $\mathcal{R}$ be a region of T invalid in T'. Then the path in T' between any node $x \in nbrhood(\mathcal{R})$ and $y \in V - (\mathcal{R} \cup adj(\mathcal{R}))$ must use an edge in $\Gamma(\mathcal{R})$.*

We also have the following proposition about the edges in $\Gamma(\mathcal{R})$.

Proposition 7.6.8 *Let $\mathcal{R}$ be an invalid region in T. Then every edge $e \in \Gamma(\mathcal{R})$ has an endpoint in $adj(\mathcal{R})$.*

Proof Since $nbrhood(\mathcal{R}) = core(\mathcal{R}) \cup adj(\mathcal{R})$, every edge $e \in \Gamma(\mathcal{R})$ has an endpoint in either $core(\mathcal{R})$ or $adj(\mathcal{R})$. Suppose one of the endpoints of e is in $core(\mathcal{R})$. The other endpoint must be in $V - (\mathcal{R} \cup adj(\mathcal{R}))$ by the definition of $\Gamma(\mathcal{R})$. For every possible location of this other endpoint of e, we can show that e admits a local improvement. This contradicts the fact that T is locally optimal. $\square$

Next we characterize non-tree edges that have an endpoint in long 2-paths. By an *extreme* node of a 2-path $\mathcal{P}$, we mean a node in one of the two ends of $\mathcal{P}$.

Observation 7.6.9 *Let T be a 1-LOT and let $\mathcal{P}$ be a long 2-path of size at least three in T. Let $u \in \mathcal{P}$ be a node that is not one of the two extreme nodes in $\mathcal{P}$. If (u, v) is a non-tree edge, then u must be adjacent to an extreme node u' of $\mathcal{P}$ and v must be a special leaf in a region $\mathcal{R}$ such that $u' \in adj(\mathcal{R})$.*

Proof Suppose u is not adjacent to an extreme node of $\mathcal{P}$. Note that $v \notin \mathcal{P}$ by Lemma 7.5.3. Since u is not adjacent to an extreme node of $\mathcal{P}$, the edge (u, v) encloses at least two consecutive nodes of degree two in $\mathcal{P}$. This violates property $P1$ contradicting that T is a 1-LOT. Therefore, u must be adjacent to an extreme node u' of $\mathcal{P}$.

Since u is adjacent to an extreme node in a 2-path, and (u, v) is a non-tree edge of a locally optimal tree T, by properties P2 and P3, v cannot be an internal node or a normal leaf in T. Thus v must be a special leaf in T. Suppose v is in a region $\mathcal{R}$ such that $u' \notin adj(\mathcal{R})$. Then the edge (u, v) encloses at least two consecutive nodes of degree two in the path between $\mathcal{R}$ and u' in T. This violates property P1 contradicting the local optimality of T. $\square$

We illustrate typical non-tree edges in a 1-LOT in Figure 7.5.

Now we proceed with the proof of Lemma 7.6.6. Pick a vertex in the core of one of the $k(\geq 2)$ invalid regions as the root vertex r and direct all edges of T' towards this vertex. Consider a directed subtree T'' of this tree rooted at r that is edge-minimal with respect to the following property: for each invalid region $\mathcal{R}$ not containing r, there is a directed path in the subtree T'' from some node in $core(\mathcal{R})$ to r. By the edge-minimality property, the leaves of T'' are nodes in the core of some invalid region $\mathcal{R}$ not containing r. Furthermore, the edge minimality property also implies that the tree T'' contains exactly one node in $core(\mathcal{R})$ for each invalid region $\mathcal{R}$ not containing r. Thus T'' has at most $k-1$ leaves.

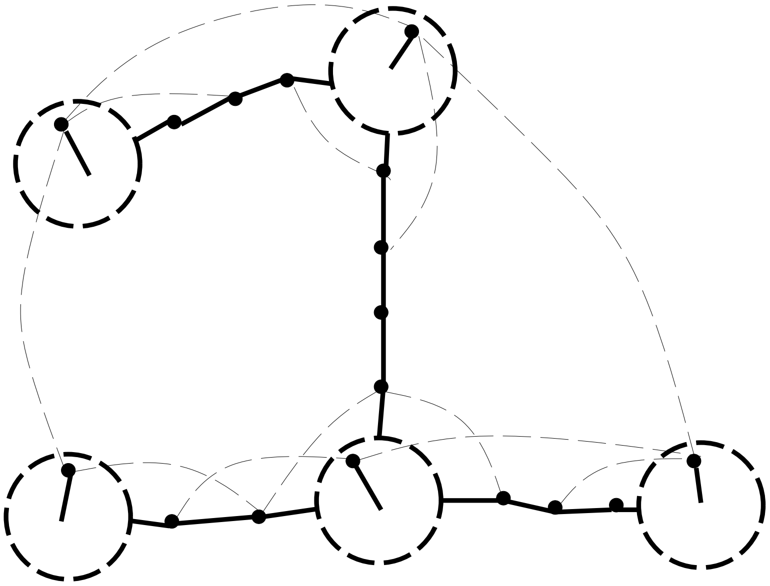


Figure 7.5: Typical non-tree edges in a 1-LOT are show as dashed lines. The tree edges are dark while the regions are shown as circles with thick dashed edges. The pendant nodes inside a regions are special leaves within the region.

For each invalid region $\mathcal{R}$ not containing r, let $v_{\mathcal{R}}$ be the node in $core(\mathcal{R})$ in T''. Consider the path in T'' from $v_{\mathcal{R}}$ to r. Let $a_{\mathcal{R}}$ be the first node along this path that is in $\mathcal{R}' \cup adj(\mathcal{R}')$ for some $\mathcal{R}' \neq \mathcal{R}$.

Claim 7.6.10 *There is a long 2-path $\mathcal{P}$ incident on $\mathcal{R}$ and $\mathcal{R}'$. Furthermore, $\mathcal{P}$ is deficient in T'.*

Proof The proof essentially uses the local optimality of T. Suppose for a contradiction that $\mathcal{R}$ and $\mathcal{R}'$ are not adjacent to the same 2-path in T. By Observation 7.6.7, the path from $v_{\mathcal{R}}$ to $a_{\mathcal{R}}$ in T' must use an edge $e = (x, y)$ in $\Gamma(\mathcal{R})$. By Proposition 7.6.8, $x \in adj(\mathcal{R})$. Let $\mathcal{P}$ be the long 2-path containing x and suppose $\mathcal{P}$ is incident on $\mathcal{R}$ and some other region $\mathcal{S}$.

Suppose $y = a_{\mathcal{R}}$. Then $a_{\mathcal{R}} \in \mathcal{R}' \cup adj(\mathcal{R}')$ for some $\mathcal{R}' \neq \mathcal{R}$. Suppose $a_{\mathcal{R}} \in adj(\mathcal{R}')$. Then by property P3 and Lemma 7.5.3, the edge $(x, a_{scrr}) \in \mathcal{P}$ and $\mathcal{P}$ has size two. Also in this case $\mathcal{R}' = \mathcal{S}$. Otherwise $a_{\mathcal{R}} \in \mathcal{R}'$. By property P3, the node $a_{\mathcal{R}}$ must be a special leaf in $\mathcal{R}'$. By Lemma 7.6.4, the 2-path $\mathcal{P}$ must be incident on $\mathcal{R}'$. Thus $\mathcal{R}' = \mathcal{S}$ in this case also.

Now suppose that $y \neq a_{\mathcal{R}}$. Then since y is not in any region, it must in a 2-path. By Lemma 7.5.3, the edge (x, y) cannot be a nontree edge and so y must be the node adjacent to x in $\cap$. Note that in this case $\mathcal{P}$ has size at least three. We follow the path in T' from y to $a_{\mathcal{R}}$. Let the last edge in this path be $(z, a_{\mathcal{R}})$. Since all the nodes in this path from y to z are in 2-paths, by Lemma 7.5.3, all the edges between them in this path are edges of $\mathcal{P}$ as well. If $a_{\mathcal{R}} \in \mathcal{P}$, then the last edge

$(z, a_{\mathcal{R}})$ is in $\mathcal{P}$ as well and $\mathcal{R}' = \mathcal{S}$. Otherwise, $(z, a_{\mathcal{R}})$ is a non-tree edge and by Lemma 7.5.2 and Observation 7.6.9, $a_{\mathcal{R}}$ must be a special leaf in the region $\mathcal{S}$. Thus $\mathcal{R}' = \mathcal{S}$ in this case as well.

We now show that $\mathcal{P}$ is deficient in T'. Suppose that $a_{\mathcal{R}}$ is a special leaf in $\mathcal{R}'$. Then $\mathcal{R}'$ is a special region. In this case, since $a_{\mathcal{R}}$ is on a path in T'' between $v_{\mathcal{R}}$ to the root r and T'' is a directed version of T', the node $a_{\mathcal{R}}$ is internal in T'. Thus $\mathcal{R}'$ is not an invalid region in T'. Note that the path between $v_{\mathcal{R}}$ and $a_{\mathcal{R}}$ in T' contains all but one node of $\mathcal{P}$. Thus at most one node of $\mathcal{P}$ is a leaf in T'. Since $\mathcal{R}'$ is a special region, the allowed number of leaves in $\mathcal{P}$ by Lemma 7.6.5 is at least three. Since T' contains at most node of $\mathcal{P}$ as a leaf, this 2-path is deficient in T'.

Now suppose $a_{\mathcal{R}}$ is in $adj(\mathcal{R}')$. In this case, the path between $v_{\mathcal{R}}$ and $a_{\mathcal{R}}$ in T' contains all nodes in $\mathcal{P}$ and so no node in this 2-path is a leaf in T'. Thus $\mathcal{P}$ is deficient in this case as well.

□

For each invalid region $\mathcal{R}$ other than the one containing r, we identify a long 2-path $\mathcal{P}$ as above using the path in T'' from $v_{\mathcal{R}}$ to r.

Claim 7.6.11 *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two long 2-paths identified using two distinct invalid regions $\mathcal{R}_1$ and $\mathcal{R}_2$. Then $\mathcal{P}_1$ and $\mathcal{P}_2$ are distinct as well.*

Proof Suppose $\mathcal{P}_1 = \mathcal{P}_2$. Then these must both be a 2-path $\mathcal{P}$ between $\mathcal{R}_1$ and $\mathcal{R}_2$. Since $\mathcal{P}$ is a long 2-path incident on two invalid regions, by the definition of deficiency, no node of $\mathcal{P}$ is a leaf in T'. Also note that all the edges in T with both endpoints in $\mathcal{P}$ are in T'' in this case. When $\mathcal{P} = \mathcal{P}_1$ was identified using $\mathcal{R}_1$, these edges are directed away from $\mathcal{R}_1$ towards $\mathcal{R}_2$ in T''. Also when $\mathcal{P} = \mathcal{P}_2$ was identified using $\mathcal{R}_2$, these edges must be directed the other way in T'', a contradiction. Thus $\mathcal{P}_1$ and $\mathcal{P}_2$ must be distinct. □

Thus we have identified $k - 1$ deficient long 2-paths in T'. This completes the proof of Lemma 7.6.6.

□

Now we prove our final bound on the number of leaves of any spanning tree T' that are nodes in long 2-paths in a 1-LOT T. The bound depends on S, λ_n, the number of normal leaves which are still leaves in T', and the number of invalid special regions.

Lemma 7.6.12 *For any spanning tree T', the number of leaves of T' that are nodes of long 2-paths in T is at most*

$$4 \cdot S + 3 \cdot \lambda_n - \lambda_n^* - 2 \cdot S^*,$$

where λ_n^* is the number of normal leaves of T which are also leaves in T', and S^* is the number of special regions of T that are invalid in T'.

Proof For any $i \geq 1$, let S_i and R_i denote the number of special and normal regions of degree i respectively. Then we have $S = \sum_{i \geq 1} S_i$ and $R = \sum_{i \geq 1} R_i$. If we regard T as a tree of regions, the sum of degrees of regions is equal to two times the number of long 2-paths. Namely, $\sum_{i \geq 1} i \cdot S_i + \sum_{i \geq 1} i \cdot R_i \leq 2 \cdot P_\ell$.

We interpret Lemma 7.6.5 using the notion of charge contributions of regions to incident long 2-paths. Using this interpretation, the total number of leaves of T' in long 2-paths of T is at most $2 \cdot \sum_{i \geq 1} i \cdot S_i + \sum_{i \geq 1} i \cdot R_i \leq 4 \cdot P_\ell - \sum_{i \geq 1} i \cdot R_i$, by the previous inequality.

Note that a single normal leaf is also a normal region.

Claim 7.6.13 *If a normal leaf of T is also a leaf in T', then this normal region cannot contribute any charge in the scheme above to its incident long 2-path.*

Proof Formally, we show that if $\mathcal{L}$ is a normal leaf with incident long 2-path $\mathcal{P}$, and if $\mathcal{R}$ is the other region adjacent to $\mathcal{P}$, then the number of leaves of T' in $\mathcal{P}$ is one less than that claimed in Lemma 7.6.5. This can be proved by looking at the two cases when $\mathcal{R}$ is normal and special. As before, the proof is by contradiction where we assume more leaves in T' than claimed and prove a contradiction to the 1-optimality of T. □

By the above claim, we can subtract one for each such region that is also a leaf in T'. Thus we arrive at the upper bound on the number of leaves of

$$4 \cdot P_\ell - \sum_{i \geq 1} i \cdot R_i - \lambda_n^* = 4 \cdot (R + S - 1) - \sum_{i \geq 1} i \cdot R_i - \lambda_n^*$$

by (7.2).

Note that the number of invalid regions in T' is precisely sum of the number of internal normal regions and invalid special regions. The number of such regions is $R - R_1 + S^*$. By Lemma 7.6.6 and the definition of deficiency, we can then subtract at least $2 \cdot (R - R_1 + S^* - 1)$ from the bound derived above. Thus the final upper bound on the number of leaves in T' from long 2-paths in T is

$$4 \cdot (R + S) - \sum_{i \geq 1} i \cdot R_i - \lambda_n^* - 2 \cdot (R - R_1 + S^*).$$

Simplifying using $R = \sum_{i \geq 1} R_i$ and $R_1 = \lambda_n$ as shown below proves the lemma.

$$\begin{aligned} 4 \cdot (R + S) & - \textstyle\sum_{i \geq 1} i \cdot R_i - \lambda_n^* - 2 \cdot (R - R_1 + S^*) \\ = & \ 4 \cdot S + 2 \cdot R - \textstyle\sum_{i \geq 1} i \cdot R_i + 2 \cdot R_1 - \lambda_n^* - 2 \cdot S^* \\ \leq & \ 4 \cdot S + 3 \cdot R_1 - \lambda_n^* - 2 \cdot S^*. \end{aligned}$$

□

Now we prove Theorem 7.6.1.

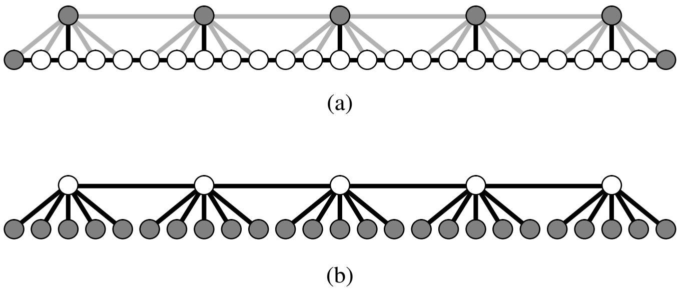


Figure 7.6: (a) An example of 1-LOT. (b) The maximum-leaf spanning tree of the graph in (a).

Proof of Theorem 7.6.1

We allow all high-degree nodes in T and all nodes in short 2-paths of T to be leaves in T^*. Using Lemma 7.6.12 with $T' = T^*$, we infer that the number of leaves in T^* on long 2-paths of T is at most $4S + 3\lambda_n - \lambda_n^* - 2S^*$. The remaining nodes in G are just special leaves and normal leaves. Since exactly λ_n^* normal leaves are still leaves in T^*, and at least $S - S^*$ special leaves are not leaves in T^*, the number of leaves in T^* is bounded by

$$N_3 + P_s + (4S + 3\lambda_n - \lambda_n^* - 2S^*) + (\lambda_s - (S - S^*) + \lambda_n^*).$$

Note that $S \leq \lambda_s$ because each special region contains at least one special leaf. From (7.2) and (7.1), we get $S + P_s \leq N_3 + \lambda_n - R$. It follows that

$$\begin{aligned}
\lambda(T^*) &\leq N_3 + \lambda_s + 3\lambda_n + 3S + P_s \\
&\leq (N_3 + 3\lambda_s + 3\lambda_n) + (N_3 + \lambda_n - R) && \text{(using } S \leq \lambda_s\text{)} \\
&= (2N_3 + 3\lambda_s + 3\lambda_n) + (R_1 - R) \\
&\leq 2N_3 + 3\lambda_s + 3\lambda_n && \text{(since } R_1 \leq R\text{)} \\
&\leq 2n_1 + 3n_1 && \text{(since } \lambda_s + \lambda_n \text{ is just } n_1\text{)} \\
&= 5\lambda(T).
\end{aligned}$$

By Lemma 7.6.2, we know

$$\frac{\lambda(T_0^*)}{\lambda(T_0)} \leq \frac{\lambda(T^*)}{\lambda(T)} \leq 5.$$

Therefore Theorem 7.6.1 is proved. □

We conclude this section by presenting an example of 1-LOT such that the bound of 5 in Theorem 7.1.1 is approachable arbitrarily closely. It is easy to verify that the tree in Figure 7.6-(a) is a 1-LOT. Since the structure is symmetric and repetitive, we only need to verify three different kinds of non-tree edges as candidates for 1-improvements. Since these three types of non-tree edges violate none of the properties P1, P2 or P3, the tree is a 1-LOT. The maximum-leaf spanning tree for this graph is shown in Figure 7.6-(b). Clearly, the longer this 1-LOT is, the closer to five the ratio is.

7.7 Performance Guarantee of 2-LOTs

In this section we prove the following restatement of Theorem 7.1.2.

Theorem 7.7.1 *Let G be a given simple undirected graph. If T^* is a maximum-leaf spanning tree of G and T is a 2-LOT of G, then $\lambda(T^*) \leq 3 \cdot \lambda(T)$.*

In the following discussion, we use G to denote the given undirected graph. Let T be a 2-LOT of G, T' a fixed spanning tree of G, and T^* a maximum-leaf spanning tree of G. Furthermore, a 2-path of T is termed *external* if it is adjacent to a leaf of T; otherwise, it is termed *internal*. We do not use augmentation as we did in the previous section. With these notations and definitions, we shall show that for any fixed T', $\lambda(T')$ is at most $3\lambda(T)$. Clearly, this implies Theorem 7.7.1.

Marking and Better Bounds

As in the proof of Theorem 7.6.1, we get a better performance ratio by bounding the number of leaves of T' in long and short 2-paths. As before, we bound the number of leaves of T' that are in the following three accounts: (a) leaves in T, (b) high-degree nodes in T, and (c) nodes in 2-paths in T, independently. The number of leaves in T' from the first two categories is at most $2\lambda(T)$ as before.

To bound the number of leaves of T' in 2-paths in T, we exhibit a *marking scheme* that marks leaves of T as follows. We only mark leaves in T that are internal nodes in T'. Thus marked leaves represent a slack in the account (a). Moreover, we show that we can mark as many leaves as the number of leaves of T' that are nodes of any long 2-path or external short 2-path of T. Thus the number of leaves of T' that are nodes from these two categories is also bounded by the slack in account (a). The remaining nodes of T are those in internal short 2-paths. The number of such paths is at most $N_3 \leq \lambda(T)$. Thus the total number of leaves in T' is at most $3\lambda(T)$ as claimed. It remains to prove the following lemma.

Lemma 7.7.2 *Let T be a 2-LOT of G and T' be any fixed spanning tree of G. If there are ℓ leaves of T' in long 2-paths and external short 2-paths of T, then we can mark at least ℓ leaves of T.*

To prove the above lemma we need a few preliminaries.

Second connections, cross connections, and normal connections

The marking proceeds by identifying three kinds of edges in $T' - T$. We regard T as a tree of regions as we did for the analysis of 1-LOT. For a long 2-path $\mathcal{P} = (v_1, v_2, \ldots)$, an edge (v, v_2) in $T' - T$ is called a *second connection*, whenever v is in the region adjacent to $\mathcal{P}$ on the side of v_1.

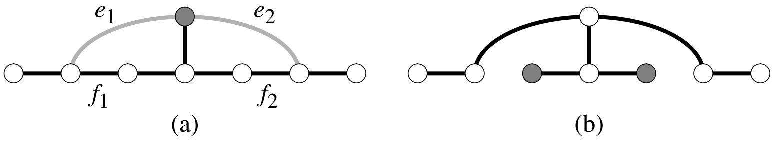


Figure 7.7: If e_1 and e_2 are two second connections sharing a special leaf as an endpoint, then $(e_1, e_2; f_1, f_2)$ is a 2-improvement.

By properties P2 and P3, we know that v must be a special leaf in T. An edge in $T' - T$ is called a *cross connection* if its endpoints are in different regions of T. By Lemma 7.5.2, both endpoints must be leaves in T. An edge (u_1, u_2) in $T' - T$ is termed a *normal connection* if u_1 is a normal leaf in T. Then by Property P2, since T is also a 1-LOT, u_2 must be a leaf in T.

Why we need the right connections...

The rough idea in using these three connections is to mark leaves of T that are endpoints of such connections. We illustrate this with an example. Let $\mathcal{P} = (v_1, v_2, \ldots)$ be a long 2-path, and let v_1 be a leaf of T'. Let the node other than v_2 that is adjacent to v_1 in T be v_0. Suppose the path P' connecting v_0 and v_2 in T' contains a second connection (w, v_2) for $\mathcal{P}$. In this case, we mark the node w that is a leaf in T but not in T' for the node v_1 that is a leaf in T'. Similarly, we use cross connections and normal connections to mark leaves of T that are not leaves in T' for leaves of T' in long 2-paths and external short 2-paths. The hard part of the proof is showing that we do not re-mark a leaf of T in this way. For example, it is not hard to see that two different second connections cannot mark the same leaf as described above. If such a re-marking of a leaf occurs, we identify a 2-improvement involving the two non-tree edges incident on the re-marked leaf. This is illustrated in Figure 7.7.

Saturation and Covering

In this subsection we define the notions of *saturation* and *covering*. Both these notions are used to account for leaves of T' that are nodes in long 2-paths or external short 2-paths of T.

Definition: A leaf v of T' in a long 2-path of T is *saturated by a second connection* e whenever v is enclosed by e and the endpoint of e that is a special leaf of T is not a leaf in T'. We call a leaf in T' *unsaturated* (by second connection) if it is not saturated by any second connection. The proof of the following lemma is similar to that of Lemma 7.5.5.

Lemma 7.7.3 *For a 2-LOT T and a fixed spanning tree T' of G, a long 2-path of T contains at most two unsaturated leaves of T'.*

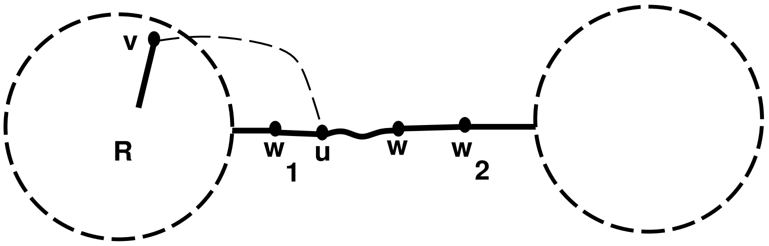


Figure 7.8: A long 2-path of a 2-LOT contains at most two unsaturated leaves. The path between the middle leaf w and one of the two other leaves, say w_1, must contain a non-tree edge (u, v) where v is a special leaf in a region adjacent to the long 2-path. This edge saturates w_1.

Proof Assume for a contradiction that there is a long 2-path $\mathcal{P}$ containing three unsaturated leaves of T', i.e. w_1, w, and w_2, where w is the middle one. Since w is in the middle and w_1 is a leaf in T', the unique tree path connecting w and w_1 in T' must contain an edge $e = (u, v)$ where $u \in \mathcal{P}$ and $v \notin \mathcal{P}$. Note that $\mathcal{P}$ is of size at least three and also that u is not an extreme node of $\mathcal{P}$. From Observation 7.6.9, we have that u must be adjacent to an extreme node u' of $\mathcal{P}$ and v must be a special leaf in a region $\mathcal{R}$ such that $u' \in adj(\mathcal{R})$. Since w_1 and w_2 lie on two different sides of u in $\mathcal{P}$, one of them must be u'. Assume without loss of generality that $w_1 = u'$ (See Figure 7.8). Now since v lies on the path in T' between u and w_1, it is not a leaf in T' and so the edge (u, v) is a second-connection saturating w_1, contradicting that w_1 is unsaturated. □

Definition: Let w be an external short 2-path[1] of T which is also a leaf in T'. Let w_1 be the normal leaf adjacent to w. We say w is *saturated by a normal connection* (v, w_1) if the unique tree path connecting w_1 and w in T' contains (v, w_1). The following is a typical proof illustrating how saturation and covering works.

Lemma 7.7.4 *Each external short 2-path in T that is a leaf in T' is saturated by a normal connection. Furthermore, at least one endpoint of this normal connection is internal in T'.*

Proof Let w be an external short 2-path in T that is a leaf in T' and let u be the normal leaf in T adjacent to w. Consider the path in T' from u to w. The first edge (u, z), say, in this path is a non-tree edge with the normal leaf u as an endpoint. Hence this edge is a normal connection and saturates w. Note that $z \neq w$ and so z is an internal node in T'. □

We classify long 2-paths of T into three types: a long 2-path in T is of type 0, 1, or 2 if it contains 0, 1, or 2 unsaturated leaves, respectively.

Definition: Let $\mathcal{P}$ be a long 2-path with two unsaturated leaves w_1 and w_2. We say $\mathcal{P}$ is *covered by a cross connection* (v_1, v_2) if the unique tree path connecting w_1 and w_2 in T' contains (v_1, v_2) and v_1 and v_2 are in different sides of $\mathcal{P}$.

---

[1] Note that an external short 2-path is nothing more than a single node. Hence the notation.

Lemma 7.7.5 *Each long 2-path of type 2 in T is covered by a cross connection. Furthermore, both endpoints of this covering cross connection are internal in T'.*

Proof Let $\mathcal{P}$ be a long 2-path with two unsaturated leaves w_1 and w_2.

Suppose w_1 and w_2 are not adjacent in $\mathcal{P}$. Then there is a node w in $\mathcal{P}$ between w_1 and w_2. Then as in the proof of Lemma 7.7.3, the path in T' between w and w_1 must contain an edge saturating either w_1 or w_2 contradicting that both these nodes are unsaturated.

Therefore w_1 and w_2 must be adjacent in T. Then the path between w_1 and w_2 in T' must contain an edge (u, v) where u and v are in different sides of $\mathcal{P}$. By the local optimality of T, both u and v must be leaves in two different regions of T. Thus (u, v) is a cross connection and covers $\mathcal{P}$.

Since the edge (u, v) is in the path in T' connecting w_1 and w_2, both u and v are internal in T'. □

Definition: Suppose $\mathcal{P}$ is a long 2-path with one unsaturated leaf w in T'. Let w_1 and w_2 be the nodes adjacent to w in T. We say $\mathcal{P}$ is *covered by a cross connection* (v_1, v_2) if the unique tree path connecting w_1 and w_2 in T' contains (v_1, v_2) where v_1 and v_2 are on different sides of $\mathcal{P}$. We have the following lemma about covering.

Lemma 7.7.6 *Each long 2-path of type 1 in T is covered by a cross connection. Furthermore, at least one endpoint of this cross connection is internal in T'.*

Proof Assume for a contradiction that there is a long 2-path $\mathcal{P}$ with one unsaturated leaf w which is not covered by any cross connection. Let w_1 and w_2 be the nodes adjacent to w in T and let P' be the unique tree path connecting them in T'. If $P' = (w_1, w_2)$, then w_1 and w_2 are both leaves in T by Property P2 and Lemma 7.5.3. This contradicts the fact that $\mathcal{P}$ is a long 2-path. Since w is unsaturated, P' cannot contain any second connection enclosing (and thus saturating) w. So P' contains only nodes and edges in $\mathcal{P}$. This contradicts the fact that w is a leaf in T'. Therefore each long 2-path of type 1 is covered by at least one cross connection.

Note that both neighbors of w in T cannot be leaves in T. Hence both endpoints of the cross connection covering w cannot be leaves in T'. Thus at least one of the endpoints of this cross connection is internal in T'[2]. □

Marking

Now we describe the marking scheme used to prove Lemma 7.7.2. We mark nodes that are leaves in T but internal in T'.

---

[2] This may actually be the case, e.g., when one of the neighbors of w in T is a normal leaf and the cross connection covering w leaves w as a leaf in T'.

- For each leaf in T' saturated by a second connection, mark the endpoint of the second connection that is a special leaf of T. (Note that by definition of a second connection, this special leaf is an internal node in T'.)
- For an external short 2-path that is a leaf in T', mark an endpoint of a normal connection that saturates this short 2-path such that this endpoint is internal in T'. (Lemma 7.7.4)
- For each long 2-path of type 2, mark the endpoints of a cross connection which covers this long 2-path. (Lemma 7.7.5)
- For each long 2-path is of type 1, mark an endpoint of a cross connection that covers this long 2-path such that this endpoint is internal in T'. (Lemma 7.7.6)

In the following subsection, we shall show that there is only one case when a leaf in T is marked twice.

Sharing of Connections

We say two distinct edges *share* a node if this node is an endpoint of both edges. Using the fact that T is a 2-LOT, it is easy to disallow the following cases of sharing.

Lemma 7.7.7 *Two normal connections saturating two external short 2-paths cannot share a leaf in T.*

Proof Assume for a contradiction that (v, u) and (v, w) are two normal connections saturating external short 2-paths u_1 and w_1, respectively. Since $u_1 \neq w_1$, we have that $((v, u), (v, w); (u, u_1), (w, w_1))$ is a 2-improvement in T. Namely, applying this edge replacement to T will make u_1 and w_1 leaves and make only v a non-leaf while both u and w remain leaves. This contradicts the fact that T is a 2-LOT. □

We prove the following lemmata in a similar manner.

Lemma 7.7.8 *Two second connections cannot share a special leaf in T.*

Proof Assume for a contradiction that (v, u) and (v, w) are two second connections, where v is a special leaf in T. Since $u \neq w$, we have that u and w must be in two different long 2-paths of T. Let u_1 and w_1 be the nodes in these two long 2-paths which are enclosed by (v, u) and (v, w), respectively. Then we have a 2-improvement $((v, u), (v, w); (u, u_1), (w, w_1))$ in T, since applying this edge replacement to T will make u_1 and w_1 leaves and make only v a non-leaf. This contradicts the fact that T is a 2-LOT. □

Lemma 7.7.9 *Two cross connections covering two long 2-paths cannot share a leaf in T.*

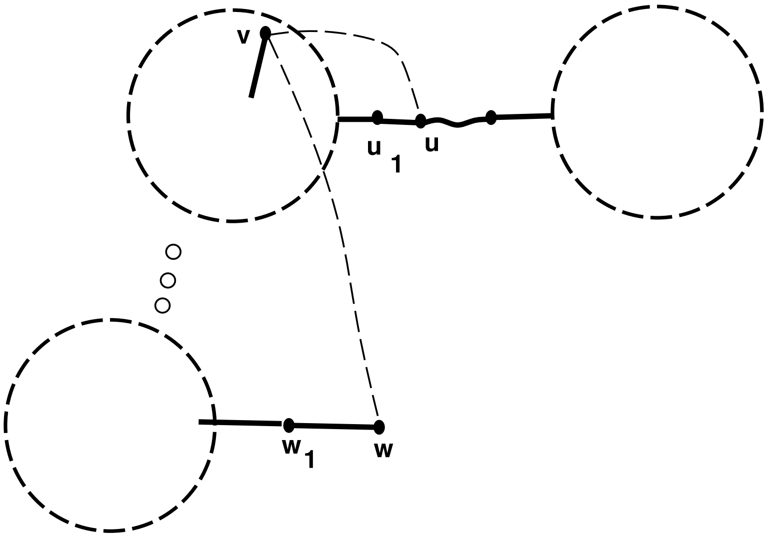


Figure 7.9: A second connection (v, w) and a normal connection (v, u) cannot share a leaf v as an endpoint. If they do, a 2-improvement $((v, u), (v, w); (u, u_1), (w, w_1))$ can be identified.

Proof Assume for a contradiction that (v, u) and (v, w) are two cross connections covering long 2-paths $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively and v is a leaf in T. Let u_1 and u_2 be two adjacent nodes in $\mathcal{P}_1$, w_1 and w_2 be two adjacent nodes in $\mathcal{P}_2$. Since $\mathcal{P}_1 \neq \mathcal{P}_2$, $((v, u), (v, w); (u_1, u_2), (w_1, w_2))$ is a 2-improvement in T. Namely, applying this edge replacement to T will make u_1, u_2, w_1, w_2 leaves and make only u, v, and w non-leaves. This contradicts the fact that T is a 2-LOT. □

Lemma 7.7.10 *A second connection and a normal connection cannot share a leaf in T.*

Proof Assume for a contradiction that (v, u) is a second connection and (v, w) a normal connection. Let u_1 be the node of degree two enclosed by (v, u), and let w_1 be the node of degree two adjacent to w in T and enclosed by (v, w). Since $u \neq w$, $((v, u), (v, w); (u, u_1), (w, w_1))$ is a 2-improvement in T. Namely, applying this edge replacement to T will make u_1 and w_1 leaves and make only v a non-leaf while w remains a leaf (See Figure 7.9). This contradicts the fact that T is a 2-LOT. □

Lemma 7.7.11 *A cross connection and a normal connection cannot share a leaf in T.*

Proof Assume for a contradiction that (v, u) is a cross connection and (v, w) a normal connection. Let $\mathcal{P}$ be the long 2-path covered by (v, u) and (u_1, u_2) be two adjacent nodes in $\mathcal{P}$. Let w_1 be the node of degree two enclosed by (v, w). Since w_1 is not in $\mathcal{P}$, $((v, u), (v, w); (u_1, u_2), (w, w_1))$ is a 2-improvement in T. Namely, applying this edge replacement to T will make u_1, u_2, and w_1 leaves and make only v and u non-leaves while w remains a leaf. This contradicts the fact that T is a 2-LOT. □

The following lemma describes the special conditions under which a special leaf in T can be marked twice.

Lemma 7.7.12 *A cross connection (v, u) covering a long 2-path $\mathcal{P}$ can share a special leaf v in T with a second connection (v, w) only if (a) w is in $\mathcal{P}$, (b) $\mathcal{P}$ is of length 2, and (c) u is not a leaf in T'.*

Proof (a) Suppose w_1 is the node in $\mathcal{P}$ enclosed by (v, w) and u_1 and u_2 are two adjacent nodes in $\mathcal{P}$. Assume for a contradiction that w is not in $\mathcal{P}$. There is a 2-improvement $((v, u), (v, w); (u_1, u_2), (w, w_1))$ in T. Namely, applying this edge replacement to T will make u_1, u_2, and w_1 leaves but make only u and v non-leaves. This contradicts the fact that T is a 2-LOT. Therefore w must be in $\mathcal{P}$.

(b) Assume for a contradiction that $\mathcal{P}$ is longer than two. Then $((v, u), (v, w); (w, w_1), (w, w_2))$ is a 2-improvement for T, where w_1 and w_2 are the two neighbors of w in T. Namely, applying this edge replacement to T will make w_1, w, and w_2 leaves and make only v and u non-leaves (See Figure 7.10). This contradicts the fact that T is a 2-LOT. Therefore $\mathcal{P}$ is of length two.

(c) Assume for a contradiction that u is a leaf in T'. Note that $\mathcal{P}$ is covered by (v, u). Then by the definition of covering, (v, u) must be in the unique tree path in T' between the two neighbors of w in T. This implies that u must be adjacent to w. Thus u must be a normal leaf adjacent to the 2-path $\mathcal{P}$ of length two made of w and w_1. But then $((v, u), (v, w); (w_1, w), (w, u))$ is a 2-improvement in T (See Figure 7.11). Namely, applying this edge replacement to Twill make w_1 and w leaves but make only v a non-leaf while u remains a leaf. This contradicts the fact that T is a 2-LOT.

Therefore this lemma is proved. □

Proof of Lemma 7.7.2 By the above lemmata we know that a leaf v in T will be marked more than once (at most twice, actually – see Lemma 7.7.12) only if it is shared by a cross connection $e = (v, u)$ covering a long 2-path $\mathcal{P}$ of length two and type 1, and a second connection $f = (v, w)$ enclosing a node w_1 in $\mathcal{P}$. Note that u is a special leaf in T. We show that the endpoint u of e cannot be marked. Let us assume for a contradiction that u is marked. By Lemma 7.7.12, since u is a special leaf and an endpoint of the cross connection $e = (v, u)$, the node u can only be marked for a leaf of T' which is saturated by a second connection e_1. By Lemma 7.7.12, e_1 is exactly

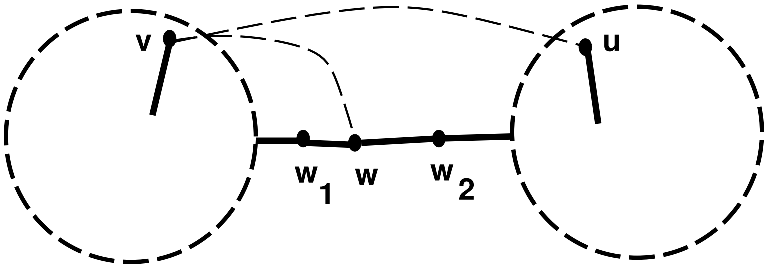


Figure 7.10: Suppose the cross connection (v, u) covering a long 2-path $\mathcal{P}$ shares a leaf v in T with a second connection (v, w). Then w is in $\mathcal{P}$ by a of Lemma 7.7.12. If $\mathcal{P}$ has size three or more, then $((v, u), (v, w); (w, w_1), (w, w_2))$ is a 2-improvement for T.

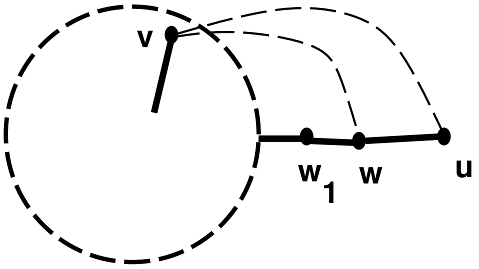


Figure 7.11: Suppose the cross connection (v, u) covering a long 2-path $\mathcal{P}$ shares a leaf v in T with a second connection (v, w). Then w is in $\mathcal{P}$ by a of Lemma 7.7.12. By part b, we have that $\mathcal{P}$ is of size two. Suppose u is also a leaf in T'. Then u must be a normal leaf adjacent to $\mathcal{P}$ where $\mathcal{P}$ is made of w and w_1. In this case $((v, u), (v, w); (w_1, w), (w, u))$ is a 2-improvement.

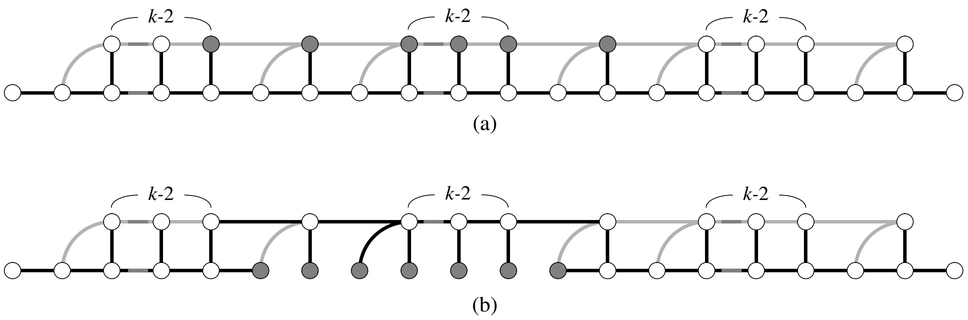


Figure 7.12: (a) An example of k-LOT with performance ratio $\frac{k+1}{k-1}$. This tree is not a $(k+1)$-LOT since the tree in (b) can be obtained by applying a $(k+1)$-improvement.

(u, w_1) which should saturate the node in $\mathcal{P}$ other than w. Since both nodes in $\mathcal{P}$ are saturated by second connections, $\mathcal{P}$ cannot be covered by any cross connection. This contradicts the fact that $\mathcal{P}$ is covered by e.

Since u is not marked and is an internal node in T' by part (c) of Lemma 7.7.12, we can shift the one of the two markings from v to u. After this shift, each node is marked exactly once and we have marked as many leaves of T as there are leaves in T' among nodes in long 2-paths and external short 2-paths. This proves Lemma 7.7.2. □

7.8 Closing Remarks

We believe that the performance ratio of 2-LOTs is better than what we prove and conjecture the following.

Conjecture 7.8.1 *The performance guarantee of 2-LOTs is 2.5.*

Papadimitriou and Yannakakis [122] identified a class of NP-hard optimization problems interreducible to one another using approximation-preserving reductions and thus took a step towards classifying NP-complete problems with respect to hardness of approximating them. They called this class MAX SNP. Recent work [6] has shown that problems complete for this class do not permit a fully polynomial-time approximation scheme unless $P = NP$. It is an intriguing open problem to determine if the maximum-leaf spanning tree problem is complete for MAX SNP or if it does allow a $(1+\epsilon)$-approximation. Along this direction, it would be interesting to examine if we can use k-LOTs to derive such a polynomial approximation scheme. We have discovered a series of examples of graphs in which the performance ratio of a k-LOT can be as bad as $\frac{k+1}{k-1}$ for each $k \geq 3$. We present a typical such graph in Figure 7.12. However, we have no proof that this is the worst possible performance of k-LOTs on any graph.

We mention in passing that the directed version of the problem we considered in which where we seek an arborescence rooted at some fixed node with the maximum number of leaves is also

NP-complete. This can be easily deduced using a reduction from the undirected problem. We note that locally optimal algorithms like those we presented for the undirected case perform badly in the directed case.

Consider the closely related minimum-leaf spanning tree problem where we seek a spanning tree with the minimum number of leaves. It is easy to see that this problem is NP-complete by a simple reduction from the Hamiltonian path problem. It is natural to ask if techniques similar to ours can be used to approximate this problem as well. We can use recent results on hardness of approximating the longest path problem [81] to infer that the minimum-leaf spanning tree problem is hard to approximate. The key observation is that a spanning tree on n nodes with at most ℓ leaves contains a path with at least $2n/\ell$ edges. This can be shown by considering an Euler tour of the tree and estimating the longest length of a path between any two consecutive leaves in the tour. A Hamiltonian graph has a spanning tree with at most two leaves. Any polynomial-time approximation algorithm with a multiplicative performance guarantee of f can be used to obtain a spanning tree of a Hamiltonian graph with at most $2f$ leaves. By our observation above, this yields a path in the graph of size at least n/f. The results of Karger, Motwani and Ramkumar [81] imply that f is at least $2^{\Omega(\log^{1-\epsilon} n)}$ for any $\epsilon > 1$ unless $\tilde{P} = NP$[3].

---

[3] Recall that $\tilde{P}$ stands for the complexity class Deterministic Quasi-polynomial time, or DTIME[$n^{\text{polylog } n}$].

Chapter 8

Conclusions and Open Issues

We have investigated several important problems arising in the design of networks and provided approximation algorithms. In the course of proving the performance guarantee of our algorithms, we have demonstrated the efficacy of three important techniques: the primal-dual technique, local optimization, and the solution-based decomposition method. Some of the challenging open problems that remain are to discover more problems for which each of these techniques will be applicable and to isolate the underlying structure of problems that makes such a technique relevant to it. We list below several specific open issues that arose in the course of our work.

- While the result on finding bridge-connected Steiner subgraphs in 2 has been generalized and studied extensively [52, 61, 90, 154], the result on the biconnected case in this chapter is the only one known to date. In recent work with D. Williamson [133, 155], we have applied the primal-dual method to derive a $2H(k)$-approximation algorithm for the minimum-cost k-vertex-connected subgraph problem where $H(k)$ is the k^{th} harmonic number. It is an intriguing open problem to apply the primal-dual method to general node-survivability problems.

- While the phased approach introduced in Chapter 3 has been taken to its limit [52, 61, 154] in the recent spate of work, it is unclear whether the performance guarantees achieved in these papers are best-possible. An important issue that remains is whether there is a more direct, one-phased approach that yields better performance guarantees for the problems addressed therein.

- We described in Chapter 4 two important variants of the Steiner problem that are at least as hard to approximate as the set-cover problem: the directed Steiner problem and the group Steiner problem. Finding logarithmic approximation algorithms for these two variants are challenging open problems.

- Though the algorithms we present in Chapter 5 provided very good approximations, their running times were quasipolynomial. We believe that our techniques in this chapter can be used to obtain polynomial-time approximation algorithms with the same performance guarantees. This is an important open problem resulting from this work.

- In Chapter 6, we studied problems in network design involving multiple objectives and provided the first approximation algorithms for many of them. The most important open problem resulting from this work is to tighten up the performance ratios in all our results. In this context, it would be interesting to investigate if the primal-dual method can be applied to provide such better guarantees and at the same time, provide a more unified framework for multi-objective problems.

- An important issue in the study of multi-objective approximation algorithms is to examine if there exists algorithms in which performance ratios achievable for different objectives can be traded off for one another.

- In another direction, it would be find ways in which the recent non-approximability results for single-objective problems can be generalized to related multi-objective problems.

- In Chapter 7, we presented a series of local optimization algorithms, and conjectured (See Conjecture 7.1.4) that algorithms using more powerful local steps perform strictly better. It would be interesting to resolve this conjecture.

- An intriguing open problem raised in Chapter 7 is to determine if the maximum-leaf spanning tree problem is complete for MAX SNP or if it does allow a $(1+\epsilon)$-approximation scheme. It would be interesting to examine if there is any relationship between finding such a scheme and Conjecture 7.1.4. Thus examining if k-LOTs for higher values of k provide such performance guarantees is an important direction for future work.

- More specifically, one could attempt to resolve Conjecture 7.8.1 that proposes that 2-LOTs have a performance bound of 2.5 rather than 3 as we prove in Chapter 7.

- There is a natural weighted version of the maximum-leaf spanning tree problem where nodes are assigned nonnegative weights and the goal is to find a spanning tree in which the leaves have maximum total weight. No approximation algorithms are known for this problem. It would be useful to understand if this problem can be attacked using a variant of the local optimization technique we used for the unweighted version.

The area of approximation algorithms has seen explosive growth over the years when this thesis has taken shape, and provided much-needed pragmatic answers to the class of impossible-to-solve

NP-complete problems. We leave the reader with the optimism that this growth will be matched in the years ahead.

Bibliography

[1] A. Agrawal, "Network Design and Network Cut Dualities: Approximation algorithms and applications," Ph. D. Thesis, TR-CS-91-60, Dept. of Computer Science, Brown University, 1991.

[2] A. Agrawal, P. Klein and R. Ravi, "When trees collide: an approximation algorithm for the generalized Steiner tree problem on networks," *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*(1991), pp. 134-144.

[3] A. Agrawal, P. Klein and R. Ravi, "How tough is the minimum-degree Steiner tree? A new approximate min-max equality," Technical Report CS-91-33, Brown University (1991).

[4] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The design and Analysis of Computer Algorithms*, Addison Wesley, Reading MA., 1974.

[5] S. Arora, and S. Safra, "Probabilistic checking of proofs," *Proc. of the 33rd Annual IEEE Symp. on Foundations of Computer Science*, (1992) pp. 2-13.

[6] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, "Proof verification and the hardness of approximation problems," *Proc. of the 33rd Annual IEEE Symp. on Foundations of Computer Science*, (1992) pp. 14-23.

[7] A. Balakrishnan, and N. R. Patel, "Problem reduction methods and a tree generation algorithm for the Steiner network problem," Unpublished paper (1985).

[8] J. Bar-Ilan, and D. Peleg, "Approximation algorithms for selecting network centers (Preliminary version)," *LNCS 519, Proceedings, 2nd Workshop, WADS '91,* Algorithms and Data Structures series, Springer-Verlag.

[9] R. Bar-Yehuda, D. Geiger, J. Naor, and R. M. Roth, "Approximation algorithms for the cycle-cover problem with applications to constraint satisfaction and Bayesian inference," (manuscript) July 1993.

[10] D. Bauer, S.L. Hakimi, and E. Schmeichel, "Recognizing tough graphs is NP-hard," *Discrete Applied Mathematics, vol. 28* (1990), pp. 191-195.

[11] J. E. Beasley, "An SST-based algorithm for the Steiner problem in graphs," Techn. Rep., Dept. of Man. Science, Imperial College, London (1985).

[12] D. Beinstock, and O. Marcotte, "On a network design problem that is intractable on trees," *Mathematics of Operations Research, vol. 15* (1990), pp. 530-544.

[13] M. Bellare, S. Goldwasser, C. Lund, and A. Russell, "Efficient probabilistically checkable proofs," *Proc. of the 25th Annual ACM Symp. on the Theory of Computing* (1993), pp. 294-304.

[14] P. Berman, personal communication, 1991.

[15] P. Berman and V. Ramaiyer, "Improved approximations for the Steiner tree problem," *Proc., 3rd Annual ACM-SIAM Symposium on Discrete Algorithms* (1992), pp. 325-334.

[16] A. Blum, "Some tools for approximate 3-coloring," *Proceedings of the 31st Annual IEEE Conference on Foundations of Computer Science* (1990), pp. 554-562.

[17] H. L. Bodlaender, "On linear time minor tests and depth first search," *Proceedings of Workshop on Algorithms and Data Structures 1989*, pp. 577-590.

[18] R. Boppana, and M. Halldòrsson," Approximating maximum independent sets by excluding subgraphs," *Proc., 2nd Scandinavian Workshop on Algorithmic Theory (SWAT)* (1990), pp.13-25.

[19] W. M. Boyce, "An improved program for the full Steiner tree problem," *ACM Transactions on Mathematical Software, vol. 3* (1977), pp. 359-385.

[20] P. M. Camerini, "The min-max spanning tree problem and some extensions," *Inform. Proc. Lett.,* Vol. 7, No. 1, pp. 10-14, (1978).

[21] S. K. Chang, "The generation of minimal trees with a Steiner topology," *Journal of the ACM, vol. 19* (1972), pp. 669-711.

[22] N. Christofides, "Worst-case analysis of a new heuristic for the traveling salesman problem," Technical Report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA. (1976).

[23] F. R. K. Chung, and R. L. Graham, "A new bound for Euclidean Steiner minimum trees," *Ann. N.Y. Acad. Sci., vol. 440* (1985), pp. 328-346.

[24] F. R. K. Chung, and F. K. Hwang, "A lower bound for the Steiner tree problem," *SIAM J. Appl. Math., vol. 34* (1978), pp. 27-36.

[25] V. Chvátal, "A greedy heuristic for the set-covering problem," *Math. of OR* Vol. 4, No. 3, pp. 233-235 (1979).

[26] Chvatal V., "Tough graphs and Hamiltonian circuits," *Discrete Math. 5* (1973), pp. 215-228.

[27] E. J. Cockayne, and D. G. Schiller, "Computation of Steiner minimal trees," in *Combinatorics*, D. A. Welsh, and D. R. Wodall (eds.) (1972).

[28] C. J. Colbourn, *The combinatorics of network reliability*, Oxford University Press (1987).

[29] C. R. Coullard, A. Rais, R. L. Rardin, and D. K. Wagner, "Linear-time algorithms for 2-connected Steiner subgraph problems on special classes of graphs," *Networks, Vol. 23*, (1993), pp. 195-205.

[30] G. A. Croes, "A method for solving traveling-salesman problems," *OR 6*, No. 6, (Nov.-Dec. 1958), pp. 791-812.

[31] G. B. Dantzig, "Linear programming and extensions," *Princeton University Press, Princeton, NJ* (1963).

[32] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Comm. ACM 17*, 11 (Nov. 1974), pp. 643-644.

[33] S. E. Dreyfus, and R. A. Wagner, "The Steiner problem in graphs," *Networks, vol. 1* (1971), pp. 195-207.

[34] D. Z. Du, and F. K. Hwang, "A new lower bound for the Steiner ratio," *Trans. Amer. Math. Soc., vol. 278* (1983), pp. 137-148.

[35] D. Z. Du, and F. K. Hwang, "An approach for proving lower bounds: solution of Gilbert-Pollak's conjecture on Steiner ratio," *Proceedings of the 31st Annual IEEE Conference on Foundations of Computer Science* (1990), pp. 76-85.

[36] C. W. Duin, an A. Volgenant, "Some generalizations of the Steiner problem in graphs," *Networks, 17*, pp. 353-364, (1987).

[37] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal on Mathematics, vol. 17* (1965), pp. 449-467.

[38] J. Edmonds and E. L. Johnson, "Matching, Euler tours and the Chinese postman," *Math. Programming, vol. 5* (1973), pp. 88-124.

[39] C. El-Arbi, "One heuristique pour le problem de l'arbre de Steiner," *R.A.I.R.O. Operations Research, vol. 12* (1978), pp. 207-212.

[40] K. P. Eswaran, and R. E. Tarjan, "Augmentation problems," *SIAM Journal on Computing, Vol. 5* (1976), pp. 653-665.

[41] U. Feige, S. Goldwasser, L. Lovász, and S. Safra, "Approximating clique is almost NP-complete," To appear in the *Proceedings of the 32nd Annual IEEE Conference on Foundations of Computer Science* (1991).

[42] M. R. Fellows, and M. A. Langston, "On well-partial-order theory and its applications to combinatorial problems of VLSI design," Technical Report CS-88-188, Dept. of Computer Science, Washington State Univ., 1988.

[43] C. M. Fiduccia, and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *19th IEEE Design Automation Conference* (1982) , pp. 175-181.

[44] L. R. Ford, Jr., and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, New Jersey (1962).

[45] H. Frank, and I. T. Frisch, "Communication, Transmission, and Transportation Networks," Addison-Wesley (1971).

[46] Greg N. Fredrickson, and Joseph Ja'Ja', "Approximation algorithms for several graph augmentation problems," *SIAM Journal on Computing, Vol. 10, No. 2* (1981), pp. 270-283.

[47] Greg N. Fredrickson, and Joseph Ja'Ja', "On the relationship between the biconnectivity augmentation and Traveling Salesman problems", *Theoretical Computer Science 19* (1982), pp. 189-201.

[48] A. M. Frieze, G. Galbiati, and F. Maffioli, "On the worst-case performance of some algorithms for the asymmetric traveling salesman problem," *Networks, 12*, pp. 23-39, (1982).

[49] Martin Fürer and Balaji Raghavachari, "An $\mathcal{NC}$ approximation algorithm for the minimum degree spanning tree problem," Proceedings of the 28th Annual Allerton Conference on Communication, Control and Computing (1990), pp. 274-281.

[50] Martin Fürer and Balaji Raghavachari, "Approximating the minimum-degree spanning tree to within one from the optimal degree," *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms* (1992), pp. 317-324.

[51] M. Fürer and B. Raghavachari, "Approximating the minimum-degree Steiner tree to within one of optimal", TR CS-92-13, Pennsylvania State University, June 1992.

[52] H. N. Gabow, M. X. Goemans, and D. P. Williamson, "An efficient approximation algorithm for the survivable network design problem," in *Proceedings, the third Integer Programming and Combinatorial Optimization Conference* (1993), pp 57-74.

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

[54] N. Garg, V. Vazirani, and M.Yannakakis, "Approximate max-flow min-(multi) cut theorems and their applications," *Proc., 25th Annual ACM Symp. on Theory of Computing,* (1993), pp. 698-707.

[55] N. Garg, V. Vazirani, and M.Yannakakis, "Primal-dual approximation algorithms for integral flow and multicut in trees, with applications to matching and set cover," *Proc. of the 20th Internatnl. Colloq. on Automata, Langs. and Prog.* (1993).

[56] B. Gavish, "Topological design of centralized computer networks - formulations and algorithms," *Networks 12* (1982), pp. 355-377.

[57] E. N. Gilbert, and H. O. Pollak, "Steiner minimal trees," *SIAM Journal on Applied Mathematics, vol. 16* (1968), pp. 1-29.

[58] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning,* Addison-Wesley (1989).

[59] M. X. Goemans and D. J. Bertsimas, "Survivable networks, linear programming relaxations and the parsimonious property," *OR 216-90, Center for Operations Research, MIT* (1990); An extended abstract appears as "On the parsimonious property of connectivity problems," in *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms* (1990), pp. 388-396.

[60] M. X. Goemans, and D. P. Williamson, "A general approximation technique for constrained forest problems," *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms* (1992), pp. 307-316.

[61] M. Goemans, A. Goldberg, S. Plotkin, D. Shmoys, É. Tardos, and D. Williamson, "Improved approximation algorithms for network design problems," (manuscript) July 1993.

[62] R. L. Graham, and F. K. Hwang, "Remarks on Steiner minimal trees," *Bull. Inst. Math. Acad. Sinica, vol. 4* (1976), pp. 177-182.

[63] J. R. Griggs, D. J. Kleitman, and A. Shastri, "Spanning trees with many leaves in cubic graphs," *Journal of Graph Theory*, 13 (1989), pp. 669-695.

[64] M. Grötschel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag (1988).

[65] M. Grötschel, and C. L. Monma, "Integer polyhedra arising from certain network design problems with connectivity constraints," *SIAM J. on Discrete Math.*, Vol 3, No. 4 (1990), pp. 502-523.

[66] M. Grötschel, C. L. Monma, and M. Stoer, "Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints," to appear, *Oper. Res.,* (1992).

[67] J. R. Griggs, D. J. Kleitman, and A. Shastri, "Spanning trees with many leaves in cubic graphs," *J. Graph Theory*, 13 (1989), pp. 669-695.

[68] S. L. Hakimi, "Steiner's problem in graphs and its implications," *Networks, vol. 1* (1971), pp. 113-133.

[69] Mark D. Hansen, "Approximation algorithms for geometric embeddings in the plane with applications to parallel processing problems," *Proceedings of the 30th Annual IEEE Conference on Foundations of Computer Science* (1989), pp. 604-609.

[70] M. Held, and R. M. Karp, "The traveling salesman problem and minimum spanning trees," *Operations Research, vol. 18* (1970), pp. 1138-1162.

[71] M. Held, and R. M. Karp, "The traveling salesman problem and minimum spanning trees: part II," *Math. Prog., vol. 1* (1971), pp. 6-25.

[72] D. S. Hochbaum, and D. B. Shmoys, "An unified approach to approximation algorithms for bottleneck problems," *JACM*, Vol. 33, No. 3, pp. 533-550, (July 1986).

[73] F. K. Hwang, "On Steiner minimal trees with rectilinear distance," *SIAM Journal on Applied Mathematics, vol. 30(1)* (1976), pp. 104-114.

[74] F. K. Hwang and D. S. Richards, "Steiner tree problems," *Networks*, Vol. 22, No. 1, pp. 55-90 (1992).

[75] A. Iwainsky, E. Canuto, O. Taraszow, and A. Villa, "Network decomposition for the optimization of connection structures," *Networks, 16*, pp. 205-235, (1986).

[76] A. Jain, "Probabilistic analysis of an LP relaxation bound for the Steiner problem in networks," *Networks, vol. 19* (1989), pp. 793-801.

[77] D. S. Johnson, "The NP-completeness column, an ongoing guide," *Journal of Algorithms, vol. 5* (1984), pp. 147-160.

[78] D. S. Johnson, "The NP-completeness column, an ongoing guide," *Journal of Algorithms, vol. 6* (1985), pp. 434-451.

[79] D. S. Johnson, "Approximation algorithms for Combinatorial Problems," *J. Computer System. Sci.*, 9, pp. 256-278 (1974).

[80] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, "How easy is local search?," *J. Comp. Syst. Sci.*, 37(1988), pp. 79-100.

[81] D. Karger, R. Motwani, and G. D. S. Ramkumar, "On approximating the longest path in a graph," to appear in *Proceedings, Workshop on Algorithms and Data Structures 1993.*

[82] N. Karmakar, "A new polynomial-time algorithm for linear programming," *Combinatorica, vol. 4*, pp. 373-395 (1984).

[83] R. M. Karp, "Reducibility among combinatorial problems," in R. E. Miller and J. W. Thatcher (eds.), *Complexity of Computer Communications.* Plenum Press, New York (1972), pp. 85-103.

[84] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *BSTJ* 49 (1970), pp. 291-308.

[85] S. Khuller, B. Raghavachari, and N. Young, "Balancing Minimum Spanning and Shortest Path Trees," *Proc., Fourth Annual ACM-SIAM Symposium on Discrete Algorithms* (1993).

[86] S. Khuller, and R. Thurimella, "Approximation algorithms for graph augmentation," *Journal of Algorithms*, Vol 14 (2) (1993), pp. 214-225.

[87] S. Khuller, and U. Vishkin, "Biconnectivity approximations and graph carvings", *Proc., 24th Annual ACM Symp. on Theory of Computing,* (1993).

[88] S. Khuller, U. Vishkin, and N. Young, "A primal-dual parallel approximation technique applied to weighted set and vertex cover," *Proceedings, the third Integer Programming and Combinatorial Optimization Conference* (1993), pp. 333-342.

[89] P. Klein, A. Agrawal, R. Ravi, and S. Rao, "Approximation through multicommodity flow," in *Proc. 31th Annual Symposium on Foundations of Computer Science* (1990), pp. 726-737.

[90] P. Klein, and R. Ravi, "When cycles collapse: A general approximation technique for constrained two-connectivity problems," *Proceedings, the third Integer Programming and Combinatorial Optimization Conference* (1993), pp. 39-56.

[91] P. N. Klein, and R. Ravi, "A nearly best-possible approximation for node-weighted Steiner trees," *Proceedings, the third Integer Programming and Combinatorial Optimization Conference* (1993), pp. 323-332.

[92] D. J. Kleitman and D. B. West," Spanning trees with many leaves," *SIAM J. Disc. Math.* Vol. 4, No. 1, (February 1991), pp. 99-106.

[93] P. Korhonen, "An algorithm for transforming a spanning tree into a Steiner tree," *Proc. 9th Int. Math. Progr. Symp.* (Budapest 1976), pp. 349-357.

[94] G. Kortsarz, and D. Peleg, "Approximation algorithms for minimum time broadcast," manuscript (March 1992).

[95] G. Kortsarz, and D. Peleg, "On choosing a dense subgraph," to appear in *Proc., 34th Annual IEEE Symp. on the Foundations of Computer Science* (1993).

[96] L. Kou, G. Markowsky and L. Berman, "A fast algorithm for Steiner trees," *Acta Informatica, vol. 15* (1981), pp. 141-145.

[97] J. Krarup, "The generalized Steiner problem," unpublished note (1978).

[98] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem", *Proc. American Mathematical Society*, 7(1), pp. 48-50, 1956.

[99] E. L. Lawler, "Combinatorial Optimization: Networks and Matroids," Holt, Rinehart and Winston, New York (1976).

[100] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. Shmoys (eds.), *The Traveling Salesman Problem*, Wiley, Chichester, (1985).

[101] C.- L. Li, S. T. McCormick, and D. Simchi-Levi, "The point-to-point delivery and connection problems: Complexity and algorithms," *Disc. Appl. Math.*, 36 (1992), pp. 267-292.

[102] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the Traveling-Salesman Problem," *Oper. Res.* 21 (1973), pp. 498-516.

[103] F. T. Leighton and S. Rao, "An approximate max-flow min-cut theorem for uniform multicommodity flow problems with application to approximation algorithms," *Proceedings of the 29th Annual IEEE Conference on Foundations of Computer Science* (1988), pp. 422-431.

[104] J. K. Lenstra, D. B. Shmoys, and É. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Mathematical Programming A*, 46 (1990), pp. 259-271.

[105] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall Inc. (1981).

[106] J.-H. Lin and J. S. Vitter, "ϵ-approximations with minimum packing constraint violation," *Proc., 24th Annual ACM STOC* (1992), pp. 771-782.

[107] L. Lovàsz,, "On the ratio of optimal integral and fractional covers," *Disc. Math.,* **13**, pp. 383-390 (1975).

[108] L. Lovàsz, *Combinatorial Problems and Exercises*, New York: North Holland, (1979).

[109] L. Lovász, and M. D. Plummer, "Matching Theory," Akademiai Kiadó, Budapest (1986).

[110] C. Lund and M. Yannakakis, "On the Hardness of Approximating Minimization Problems," *Proc., 25th Annual ACM Symp. on Theory of Computing*, (1993), pp. 286-293.

[111] Hsueh-I Lu, and R. Ravi, "The Power of Local Optimization: Approximation Algorithms for Maximum-Leaf Spanning Tree," in *Proceedings, Thirtieth Annual Allerton Conference on Communication, Control and Computing* (October 1992), pp. 533-542.

[112] K. Mehlhorn, "A faster approximation algorithm for the Steiner problem in graphs," *Information Processing Letters, vol. 27(3)* (1988), pp. 125-128.

[113] Z. A. Melzak, "On the problem of Steiner," *Canad. Math. Bull., vol. 4* (1961), pp. 143-148.

[114] M. Minoux, "Network synthesis and optimum network design problems: models, solution methods and applications," *Networks, vol. 19* (1989), pp. 313-360.

[115] J. S. B. Mitchell, C. Piatko, and E. M. Arkin, "Computing a shortest k-link path in a polygon", *Proc., 33rd Annual IEEE FOCS* (1992), pp. 573-582.

[116] C. L. Monma, and C. W. Ko, "Methods for designing survivable communication networks,"*NATO Advanced Research Workshop*, Denmark, (1989).

[117] C. L. Monma, B. S. Munson, and W. R. Pulleyblank, "Minimum-weight two-connected spanning networks," *Math. Programming, 46* (1990), pp. 153-171.

[118] C. L. Monma, and D. F. Shallcross, "Methods for designing communication networks with certain two-connectivity survivability constraints", *Operations Research, 37*, pp. 531-541, (1989).

[119] G. L. Nemhauser, and L. A. Wolsey, *Integer and Combinatorial Optimization*, Wiley Interscience series in Disc. Math. and Optimization (1988).

[120] C. H. Papadimitriou, A. A. Schäffer, and M. Yannakakis, "On the complexity of local search (Extended Abstract)," in *Proceedings of the twenty-second annual ACM Symposium on the Theory of Computing*, pp. 438-445 (1990).

[121] C. H. Papadimitriou, and K. Steiglitz, "Combinatorial Optimization," Prentice-Hall Inc. (1982).

[122] C. H. Papadimitriou, and M. Yannakakis, "The complexity of restricted minimum spanning tree problems," *Lecture Notes in Computer Science 71* (1979), pp. 460-470.

[123] R. G. Parker, and R. L. Rardin, "Guaranteed performance heuristic for the bottleneck traveling salesman problem," *Oper. Res. Lett. 6*, pp. 269-272, (1982).

[124] C. Payan, M. Tchuente, and N. H. Xuong, "Arbres avec un nombres maximum de sommets pendants, *Discrete Math.*, 49 (1984), pp. 267-273.

[125] J. Plesnik, "A bound for the Steiner tree problem in graphs," *Math. Slovaca, vol. 31* (1981), pp. 155-163.

[126] S. Plotkin, and É. Tardos, "Improved bounds on the max-flow min-cut ratio for multicommodity flows," *Proc., 25th Annual ACM Symp. on Theory of Computing*, (1993), pp. 691-697.

[127] R.C. Prim, "Shortest connection networks and some generalizations", *Bell System Tech Journal*, 36(6), pp. 1389-1401, 1957.

[128] R. Ravi, A. Agrawal, and P. Klein, "Ordering problems approximated: single-processor scheduling and Interval graph computation," in *Proceedings of the 18th International Colloquium on Automata, Languages and Processing '91* (1991), LNCS 510, pp. 751-762.

[129] 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," *Proc., 25th Annual ACM Symposium on the Theory of Computing* (1993), pp. 438-447.

[130] R. Ravi, and P. N. Klein, "When cycles collapse: A general approximation technique for constrained two-connectivity problems," Technical Report TR-CS-92-30, Brown University, (June 1992).

[131] R. Ravi, B. Raghavachari, and P. N. Klein, "Approximation through local optimality: Designing networks with small degree," *Proceedings, Twelfth Annual Conference on Foundations*

of Software Technology and Theoretical Computer Science, December 1992, LNCS 652, pp. 279-290.

[132] R. Ravi, R. Sundaram, M. V. Marathe, D. J. Rosenkrantz, and S. S. Ravi, "Spanning trees short or small," (manuscript) July 1993.

[133] R. Ravi and D. Williamson, "Approximation algorithms for minimum-cost k-vertex-connected subgraph," (manuscript), July 1993; a version appears in D. Williamson's thesis [155].

[134] V. J. Rayward-Smith, "The computation of nearly minimal Steiner trees in graphs," *Internat, J. Math. Ed. Sci. Tech.*, 14 (1), pp. 15-23 (1983).

[135] V. J. Rayward-Smith and A. Clare, "On finding Steiner vertices", *Networks*, 16, pp. 283-294 (1986).

[136] G. Reich and P. Widmayer, "Beyond Steiner's problem: A VLSI oriented generalization," in *Proc., 15th International Workshop on Graph-Theoretic Concepts in Computer Science*, pp. 196-210, Castle Rolduc, (1989).

[137] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, "An analysis of several heuristics for the traveling salesman problem," *SIAM Journal on Computing, vol. 6* (1977), pp. 563-581.

[138] A. Schrijver, "Min-max results in Combinatorial Optimization," in *Mathematical programming: The state of the art*, Eds. A. Bachem, M. Grötschel, and B. Korte, pp. 439-500, Springer (1983).

[139] A. Segev, "The node-weighted Steiner tree problem," *Networks, vol. 17* (1987), pp. 1-17.

[140] D. B. Shmoys and E. Tardos, "Scheduling unrelated parallel machines with costs," *Proc., 4th Annual ACM-SIAM SODA* (1993), pp. 448-454.

[141] M. L. Shore, L. R. Foulds, and P. B. Gibbons, "An algorithm for the Steiner problem in graphs," *Networks, vol. 12* (1982), pp. 323-333.

[142] J. M. Smith, "An $O(n \log n)$ heuristic for Steiner minimal tree problems on the Euclidean metric," *Networks, vol. 11* (1981), pp. 23-39.

[143] J. M. Smith, D. T. Lee, and J. S. Liebman, "An $O(n \log n)$ heuristic algorithm for the rectilinear Steiner minimal tree problem," *Eng. Optimization, vol. 4* (1980), pp. 179-192.

[144] K. Steiglitz, P. Weiner, and D. J. Kleitman, "The design of minimum-cost survivable networks," *IEEE Trans, on Circuit Theory*, CT-16, 4, pp. 455-460, (1969).

[145] J. A. Storer, "Constructing full spanning trees for cubic graphs," *Inform. Process. Lett.* 13 (1981), pp.8-11.

[146] G. F. Sullivan, "Approximation algorithms for Steiner tree problems," Tech. Rep. 249, Dept. of Computer Science, Yale Univ. (1982).

[147] H. Suzuki, T. Akama and T. Nishizeki, "Finding Steiner forests in planar graphs," *1st Ann. ACM-SIAM Symp. on Disc. Alg.* (1990), pp. 444-453.

[148] H. Takahashi and A. Matsuyama, " An approximate solution for the Steiner problem in graphs," *Math. Japonica, vol. 24* (1980), pp. 573-577.

[149] M. Tchuente, "Sur l'auto-stabilisation dans un réseau d'odinateurs," *R. A. I. R. O. Informatique Théorique 15,* (1) (1981), pp. 47-66.

[150] A. Tanenbaum, *Computer Networks*, Prentice-Hall Inc. (1981).

[151] P. J. M. van Laarhoven, and E. H. L. Aarts, "Simulated Annealing: Theory and Practice," Kluwer Academic Publishers, Dordrecht, Holland (1987).

[152] V. G. Vizing, "On an estimate of the chromatic class of a p-graph" (in Russian), *Diskretnyĭ Analiz, vol. 3* (1964), pp. 11-12.

[153] T. Watanabe, Y. Higashi, and A. Nakamura, "Graph augmentation problems for a specified set of vertices," *Proc. SIGAL'90, LNCS 450*, pp. 378-387.

[154] D. P. Williamson, M. X. Goemans, M. Mihail, and V. Vazirani, "A primal-dual approximation algorithm for generalized Steiner network problems," *Proc. of the 25th Annual ACM Symp. on the Theory of Computing* (1993), pp. 708-717.

[155] D. P. Williamson, "On the design of approximation algorithms for a class of graph problems," Ph. D. Thesis (August 1993), Department of Computer Science, MIT.

[156] P. Winter, "Steiner problem in networks : a survey," *Networks* (1987), pp. 129-167.

[157] P. Winter, "Generalized Steiner problem in outerplanar graphs,", *BIT 25* (1985), pp. 485-496.

[158] P. Winter, "Generalized Steiner problem in series-parallel networks," *J. Algorithms 7,* (1986), pp. 549-566.

[159] Pawel Winter, "An algorithm for the Steiner problem in the Euclidean plane," *Networks, vol. 15* (1985), pp. 323-345.

[160] L. Wolsey, personal communication, May 1993.

[161] R. T. Wong, “Worst-case analysis of network design problem heuristics,” *SIAM J. Alg. Disc. Math., vol. 1* (1980), pp. 51-63.

[162] R. T. Wong, “A dual ascent approach for Steiner tree problems on a directed graph,” *Math. Program., vol. 28*, (1984), pp. 271-287.

[163] Y. F. Wu, P. Windmayer and C. K. Wong, “A faster approximation algorithm for the Steiner problem in graphs,” *Acta Informatica, vol. 23* (1986), pp. 321-331.

[164] A. Z. Zelikovsky, “The 11/6-approximation algorithm for the Steiner problem on networks,” *Algorithmica*, 9, pp. 463-470.