Many Birds With One Stone: Multi-Objective Approximation Algorithms

R. Ravi[1]
M.V.Marathe[2]
S.S. Ravi[3]
D.J. Rosenkrantz[4]
H.B. Hunt III[5]

Technical Report No. CS-92-58
December 1992

[1]Brown University Box 1910, Providence, RI 02912
[2]University at Albany-Suny NY 12222
[3]University at Albany-Suny NY 12222
[4]University at Albany-Suny NY 12222
[5]University at Albany-Suny NY 12222

Many birds with one stone: Multi-objective approximation algorithms

R. Ravi[1,3] M. V. Marathe[2,4] S. S. Ravi[2,5] D. J. Rosenkrantz[2,6]
H.B. Hunt III[2,4]

Abstract

We study network-design problems with multiple design objectives. In particular, we look at two cost measures to be minimized simultaneously: the total cost of the network and the maximum degree of any node in the network. Our main result can be roughly stated as follows: given an integer b, we present approximation algorithms for a variety of network-design problems on an n-node graph in which 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 the minimum-cost degree-b-bounded network. These 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. The performance guarantee on the cost of the output network is nearly best-possible unless $NP = \tilde{P}$.

We also address the special case in which the costs obey the triangle inequality. In this case, we obtain a polynomial-time approximation algorithm with a stronger performance guarantee. Given a bound b on the degree, the algorithm finds a degree-b-bounded network of cost at most a constant time optimum. There is no algorithm that does as well in the absence of trianlge inequality unless $P = NP$. We also show that in the case of spanning networks, we can simultaneously approximate within a constant factor yet another objective: the maximum cost of any edge in the network, also referred to as the bottleneck cost of the network. We extend our algorithms to find TSP tours and k-connected spanning networks for any fixed k that simultaneously approximate all these three cost measures.

[1]Dept. of Computer Science, Brown University, Providence, RI 02912. email: rr@cs.brown.edu

[2]Dept. of Computer Science, University at Albany-SUNY, NY 12222. Email:{madhav,ravi,djr,hunt}@cs.albany.edu

[3]Research supported by an IBM Graduate Fellowship. Additional support provided by NSF PYI award CCR-9157620 and DARPA contract N00014-91-J-4052 ARPA Order No. 8225.

[4]Supported by NSF Grants CCR 89-03319.

[5]Supported by NSF Grants CCR 89-05296.

[6]Supported by NSF Grants CCR 90-06396.

1 Introduction

1.1 Motivation

Several fundamental problems in the design of communication networks can be modeled as finding a network obeying certain connectivity specifications. For instance, the network may be required to connect all the nodes in the graph (a spanning tree problem), a subset of the nodes in the graph (a Steiner tree problem) or may be only interconnect a set of site-pairs of nodes (a generalized Steiner forest problem). If we require the network to define a tour in which each node is visited exactly once, this corresponds to a TSP problem. The goal in such network-design problems can usually be expressed as minimizing some notion of "cost" associated with the network. This cost may reflect the price of constructing the network, or it may measure some form of vulnerability of the network, or it may even measure some critical price in using this network.

There are three classic examples of such cost measures. If we associate costs with feasible edges and nodes that can be used to build the network, then we may look for a network such that the price of construction is minimized. This is the *minimum-cost network design* problem and has been well studied [3, 1, 8, 12, 15, 19, 20, 26, 28, 21, 30]. 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 cost corresponds to the *minimum-degree network design* problem, which has also been well-studied [2, 9, 10, 27]. A cost measure that captures the price associated with using the network is the bottleneck cost which is the maximum cost of any edge in 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. Such problems are termed *bottleneck problems* and these have also received considerable attention [14, 25].

Finding a network of sufficient generality and of minimum cost with respect to each one of these measures can be shown to be NP-complete [11]. Hence much of the work mentioned above focuses on approximation algorithms for each of these problems. However, in applications that arise in real-world situations, it is often the case that the network to be built is required to minimize more than one of these cost measures simultaneously. Recent papers have identified many such problems [4, 18, 17, 31] wherein more than one objective is specified in the statement of the problem.

In this paper, we embark on a new research endeavor, namely studying the approximation of general network-design problems with multiple objectives in a unified way. We consider several combinations of these parameters and provide approximation algorithms that are nearly best-possible in terms of the performance guarantee for one of the parameters studied. We also examine the case of cost functions obeying the triangle inequality and provide better algorithms for this case. We proceed to outline our main results.

1.2 Degree-bounded minimum-cost network design

Consider the following simply stated 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.

Observation 1.1 *For any fixed rational number $R \geq 1$, finding 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 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 1.2 *There is a polynomial algorithm that, given an undirected graph G on n nodes with nonnegative costs on its edges, and a degree bound b, constructs a spanning tree whose maximum degree*

is $O(b \log \frac{n}{b})$ and whose cost is at most $O(\log \frac{n}{b})$ times that of the minimum-cost b-bounded spanning tree of G.

Our techniques are powerful enough to generalize to the case of constructing Steiner trees as well as generalized Steiner forests. Given an undirected graph and a set of *site-pairs* of nodes, define a b-bounded generalized Steiner forest for the site-pairs to be a subgraph such that the maximum degree of any node in the subgraph is at most b and there is a path in the subgraph between any site-pair. We have the following generalization of the above theorem.

Theorem 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 degree bound b, constructs an $O(b \log \frac{k}{b})$-bounded generalized Steiner forest for the site-pairs whose cost is at most $O(\log \frac{k}{b})$ 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.*

In fact, we can even go one step further and prove the same result as above for a whole class of constrained forest problems introduced by Goemans and Williamson in [12]. Using this extension, we can solve approximately for the first time the degree-bounded minimum-cost versions of a non-fixed point-to-point interconnection problem.

1.3 Extension to the node-weighted case

We can strengthen each of the 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. Even so, in this case, we achieve a performance ratio on the cost of the solution that is within a constant factor of best possible.

Note that any problem with costs on nodes and edges can be transformed to one with weights only on the nodes as follows: 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 extension below is a strict generalization of Theorem 1.2.

The case of spanning trees treated in Theorem 1.2 is less interesting in the case of node-weighted graphs since every node must be included in the output solution. This problem then reduces to computing a minimum-degree spanning tree that has been well-studied [9, 10].

We focus our attention on the more interesting case of Steiner trees. Recently, Klein and Ravi [19] have presented the first polynomial-time approximation algorithms for node-weighted Steiner trees. The performance guarantee of their approximation algorithms is logarithmic in the number of terminals specified in the problem. Using recent results on the hardness of approximation for the set-cover problem [24], they show that the performance guarantee is nearly best-possible (within a constant factor) unless $NP = \tilde{P}$. We extend the techniques of Klein and Ravi [19] to the case when the degree of the Steiner tree is also required to be bounded and derive the following analogue of Theorem 1.2.

Theorem 1.4 *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 degree bound b, constructs a Steiner tree spanning all the terminals whose maximum degree is $O(b \log(\frac{k}{b}))$ and whose cost is at most $O(\log k)$ times that of the minimum-cost b-bounded Steiner tree of G spanning all the terminals.*

As before, we can extend the above theorem to generalized Steiner forests and to even more general constrained forest problems addressed in [12]. We omit these extensions in this abstract.

We note that the above theorem subsumes the result of Klein and Ravi [19] since it contains their result as a special case when the degree of any node in the solution is allowed to be unbounded. Though we generalize their result, the performance guarantee on the cost of the solution that we prove matches that proved in [19] within a constant factor. Using the observation in their paper, it follows that the performance guarantee of our solution in terms of the total cost is nearly best-possible unless $NP = \tilde{P}$.

1.4 An application: approximating minimum-degree generalized Steiner forests

Theorem 1.3 has an important application. We can use this theorem to provide the first polynomial-time approximation algorithm for a class of minimum-degree forest problems considered by Ravi, Raghavachari

and Klein in [27]. A prototypical example of a problem in this class 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 whose maximum degree is minimum. The best approximation ratio for this problem achievable using the techniques in [27] is $\Omega(n^{\epsilon})$ for some constant $\epsilon > 0$. The running time of the algorithm achieving this ratio is $O(n^{\frac{1}{\epsilon}})$. We can improve the approximation factor achievable in polynomial-time for this problem by an application of Theorem 1.3.

Theorem 1.5 *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 whose maximum degree is at most $O(\log \frac{k}{\delta^*})$ times the minimum possible. Here k represents the number of nodes of G that are sites and δ^* is the minimum-degree of any generalized Steiner forest for the site-pairs.*

1.5 Approximations under triangle inequality

One way to circumvent the difficulty exhibited in Observation 1.1 is to consider cost-functions on the edges with more structure. In this direction, we turn to the case of cost-functions on the edges obeying the triangle inequality. This is a reasonable assumption in the design of communication networks where the costs typically measure geographic distances between the nodes. In this case, we present approximations that strictly conform to the degree restrictions in the input problem and approximate the bottleneck cost of the output network as well.

We introduce an extension of the short-cutting technique of Rosenkrantz, Stearns and Lewis [28] for the TSP problem to approximate the total cost, the maximum degree as well as the bottleneck cost of spanning networks.

Theorem 1.6 *[The b-MST Theorem] There is a polynomial-time algorithm that, given an undirected graph with edge costs obeying the triangle inequality and an integer $b \geq 3$, outputs a spanning tree whose degree is at most b and whose total cost is at most $(2 - \frac{(b-2)}{(n-1)})$ times optimum and whose bottleneck cost is at most twice optimum.*

If we insist on a Hamiltonian path (i.e., require $b = 2$), then the simple TSP heuristic in [28] provides a solution that approximates the total cost of the spanning tree by a factor $2(1 - \frac{1}{n})$ in an n-node graph, but provides no guarantee on the bottleneck cost. However, we can tailor our short-cutting technique to obtain a TSP with small total cost as well as small bottleneck cost.

Theorem 1.7 *There is a polynomial-time algorithm that, given a undirected graph with edge costs obeying the triangle inequality, outputs a TSP tour whose total cost is at most four times optimum and whose bottleneck cost is at most eight times optimum.*

We note that the techniques of Parker and Rardin [25] and subsequently Hochbaum and Shmoys [14] apply to approximate the bottleneck cost of the TSP within a factor of two of the optimum. Similarly, the techniques of Rosenkrantz, Stearns and Lewis [28] apply to approximate the total cost of the TSP within twice the optimum. However, the theorem above is the first of its kind on approximating both these cost measures for the TSP simultaneously even under triangle inequality.

We can extend Theorem 1.6 to higher-connected networks as follows.

Theorem 1.8 *There is a polynomial-time algorithm that, given an undirected graph with edge costs obeying the triangle inequality, an integer $k \geq 2$ (the edge-connectivity requirement), and an integer $b \geq k$ (the degree bound), 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 $4(k+2)$ times optimum, and the maximum cost of any edge in the subgraph is at most $4k$ times optimum.*

This theorem is proved by using short-cuts that induce higher-connected graphs. We can extend the results in Theorem 1.6 to more general one-connected networks. Thus, for instance, we can obtain approximation algorithms for degree-bounded minimum-cost Steiner trees and for degree-bounded minimum bottleneck Steiner trees. These extensions are straightforward and are detailed in the Appendix.

In the next section, we first discuss related work. In Section 3, we provide a proof of Theorem 1.2 as warm-up. Then we sketch the proof of its extensions and its application to prove Theorem 1.5. Then

we move on to present the algorithm and proof of Theorem 1.4 in the following section. After this, we present the results for the case of cost functions obeying triangle inequality. In the appendix we include some simple NP-hardness results and also some observations on finding approximately minimum Steiner trees under multiple objectives.

2 Related work

2.1 Multi-objective approximations

There has been very little work on approximating problems with many simultaneous objective functions. One important contribution to this area is by Bar-Ilan and Peleg [4]. In this work, they provide approximation algorithms for problem of allocating network centers wherein each center is allowed to service only a bounded number of nodes. Wong [31] examines 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. This result is quite similar in flavor to Observation 1.1 wherein we are required to conform to the degree requirement and wish to approximate only the total cost. Klein, Agrawal, Ravi and Rao [18] provide 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 [17] provide an algorithm for finding a spanning tree whose weight is at most a constant times that of the MST such that the distance in this tree from the root is at most a constant times the distance in the input graph.

Iwainsky et al. [16] formulated a version of the minimum-cost Steiner problem with an additional cost based on node-degrees. Duin and Volgenant [6] formulate the degree-bounded Steiner tree problem motivated by practical considerations.

2.2 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 [9, 10], Steiner trees [2, 10], generalized Steiner forests and more general one-connected networks as well as two-edge-connected spanning networks [27].

A lot of research effort has been dedicated to the study of minimum-cost network problems: e.g., spanning trees [3, 20, 26], TSPs [21, 8], Steiner trees [15, 30], generalized Steiner trees [1] and even more general one-connected networks [12].

Bottleneck problems have been investigated in [14, 25].

3 Approximating both the degree and cost: the edge-weighted case

In this section, we sketch a proof of Theorem 1.2. First, we provide some background and address a problem which comes up as a subproblem in our solution.

3.1 Background

In this section, we provide some background material needed to understand the working of our algorithm and our proofs. First, we recall a tree decomposition theorem. Then we look at a variant of a degree constrained subgraph problem which we use as a subroutine in our solutions.

3.1.1 A tree decomposition theorem

The following tree decomposition theorem from [19] will be used in bounding the cost of the solution obtained by our algorithms.

Claim 3.1 *Let T 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 T are edge-disjoint.*

3.1.2 The Degree Constrained Subgraph (DCS) problem

We describe a DCS problem in this section since we use an algorithm for this problem as a subroutine in our solutions. 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 any) such that the degree of any vertex v in this subgraph is $f(v)$. This problem is sometimes referred to as the f-factor problem [23], and is known to be polynomially solvable [23, 7, 29]. The following more general variant of this problem is also known to be polynomially solvable using matching techniques [23].

Definition: We denote the degree of a node v in a subgraph H by $deg_H(v)$.

Fact 3.2 *[23] (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 $deg_H(v) = 1$ for all vertices $v \in T$, $0 \leq deg_H(v) \leq b$ for all vertices $v \in S$, and $deg_H(v) \equiv 0 \pmod 2$ for all $v \in S$.*

The b-bounded even DCS problem described above is a generalization of the famous T-join problem [7]. Note that when b is allowed to be unbounded, then the b-bounded even DCS problem reduces to the T-join problem. A proof of Fact 3.2 can be provided using a transformation to a min-cost perfect matching problem as is done for the T-join problem in [23, 7]. Using results on the structure of T-joins, it is easy to prove the following property of any subgraph H satisfying the conditions of Fact 3.2.

Proposition 3.3 *Let H be any subgraph satisfying the conditions in Fact 3.2. 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 will use the above proposition in the proof of the performance guarantee.

3.2 The spanning tree case: Proof of Theorem 1.2

In this section, we prove Theorem 1.2 by describing the algorithm referred to in the theorem. We shall use b to denote the degree bound specified in the problem. We also use OPT_b to denote the minimum cost of any b-bounded spanning tree of the input graph.

3.2.1 Overview

The algorithm follows the same skeletal outline as that of Fürer and Raghavachari [9] 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 $O(\log \frac{n}{b})$ phases 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 phase, we add edges between the components thus 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 $O(\log \frac{n}{b})$ phases in all. We also ensure that in each phase, the degree of any node in the graph increases by at most $O(b)$ 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 1.2.

3.2.2 The Algorithm

Input: An undirected graph $G = (V, E)$ and a degree bound $b \leq n - 1$ where $n = |V|$.
Output: A spanning tree in which the maximum degree is $O(b \log \frac{n}{b})$ and whose total cost is $O(OPT_b \log \frac{n}{b})$.

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 Add a node in G_i for every node in V and one node for every $C_j \in \mathcal{C}$. Thus the node set V_i of G_i is $V \cup \mathcal{C}$.

6 Add the following edges in G_i: Between all the nodes of V in each connected component $C_j \in \mathcal{C}$, add edges of zero-cost to form a clique.
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 $c(e)$, add four edges in G_i: let $u \in C_u \in \mathcal{C}$ and $v \in C_v \in \mathcal{C}$. Note that since $e \in E'$, we have $C_u \neq C_v$. We add 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 add an extra dummy node z and add zero-cost edges between z and every $C_j \in \mathcal{C}$.

8 Find a $2b$-bounded even DCS of G_i setting $S = V$ and $T = \mathcal{C}$ (or $T = \mathcal{C} \cup \{z\}$ if $|\mathcal{C}|$ is odd). This can be done using the algorithm referred to in Fact 3.2.

9 $F := F \cup$ the set of edges of G found by the DCS algorithm in Step 8.

10 $i := i + 1$.

11 Now the number of connected components of (V, F) is $O(b)$. Contract each of these connected components to single nodes and find an MST of these nodes in the original graph. Add the edges of these MST to the set F.

12 Output a spanning tree of (V, F).

Note that at each iteration, only the edges of $E' \subseteq E$ are represented in G_i and so these are the only edges chosen as a solution to the DCS problem (other than the zero-cost edges). However, the solution to the DCS problem may choose one or more copies of an edge $e \in E'$. In this case, in Step 9, we include only one copy of this edge in the set F.

3.2.3 Performance Guarantee

We prove the performance guarantee using a series of lemmas.

Lemma 3.4 *The total number of iterations of the above algorithm is* $O(\log \frac{n}{b})$.

Proof: We prove the above lemma using a potential function argument. Define the potential at each iteration of the algorithm to be the number of components in the set $\mathcal{C}$ at the beginning of the iteration. We claim that at each iteration, this number reduces by a factor of half. To show this, we use Proposition 3.3 to derive a pairing of the components in $\mathcal{C}$ such that there are edge-disjoint paths between the pairs in the DCS solution subgraph found in Step 8 of the iteration. It is easy to convert these paths in G_i to paths in (V, F) of the same cost between corresponding pairs of components. Thus after adding the DCS solution found in this iteration, each component merges with at least one other component. Hence the total number of components in the resulting graph reduces by a factor of half. Since there are n components to begin with and we stop and compute a MST when the number of connected components falls to $O(b)$, there are $O(\log \frac{n}{b})$ iterations in all. □

Lemma 3.5 *The degree of the spanning tree output by the above algorithm is* $O(b \log \frac{n}{b})$.

Proof: The lemma is proved using the constraint on the degree of the nodes in the DCS solution formed at each iteration of the algorithm. At any iteration i except the last, the contribution to the degree of any node in the solution subgraph F is at most $2b$ due to edges incident on its copy in G_i and at most 1 due to an edge incident on the component in $\mathcal{C}$ to which the node belongs. Hence the degree of any node of V due to edges in the DCS formed in G_i is at most $2b + 1$. This is also an upper bound on the degree of the subgraph induced on the node set V by the edges in this DCS solution. The degree added to any node using the edges in the last edges is $O(b)$ since we add a tree spanning this many nodes. Since the number of iterations is $O(\log \frac{n}{b})$ by Lemma 3.4, the degree of any node in the solution subgraph F in Step 12 is $O(b \log n)$. Hence any spanning tree of (V, F) also has maximum degree $O(b \log \frac{n}{b})$. □

Lemma 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 this iteration. Since we find a minimum-cost DCS at Step 8 of this iteration and this is also the cost of the edges of G added to F in this iteration, this set of edges has cost at most OPT_b.

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. Contract all the nodes of T^* in the same connected component in $\mathcal{C}_i$ to a single supernode to form a graph $T^*(i)$ on the node set $\mathcal{C}_i$. This operation preserves connectivity and since T^* is connected, $T^*(i)$ is a connected graph as well. Let T_i be any spanning tree of $T^*(i)$. Then T_i is a spanning tree on the nodes $\mathcal{C}_i$ and has cost at most OPT_b since all the edges in T_i are edges of T^*. But the degree of any node in T_i may be much larger than b as a result of contraction. However, we shall show that we can use T_i to derive a solution to the DCS problem in G_i having small degree at the nodes of V. In particular, we prove the following claim.

Claim 3.7 *Using the spanning tree T_i on the node set $\mathcal{C}_i$, we can construct a solution of cost at most that of T_i to the DCS problem set up in Step 8.*

Proof: Assume for the sake of simplicity that $|\mathcal{C}_i|$ is even[7]. Applying Claim 3.1 to the tree T_i with all the nodes (representing components in $\mathcal{C}_i$) 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 now convert these paths into a feasible solution to the DCS problem as follows:

(i) First, we consider 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 in $\mathcal{P}$ joined by an edge (u, v) of T^*. Since $C_u \neq C_v$, the edge $e = (u, v) \in E'$. So by the construction in Step 6 of the algorithm, there is an edge (C_u, C_v) of cost $c(e)$ in G_i. In this case, we add this edge (C_u, C_v) of cost $c(e)$ to the DCS solution. We do this for every pair of components joined by a single edge in T_i.

(ii) Then we 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 of components in $\mathcal{P}$. Let the path between them in T_i be $(C_x, C_1), (C_1, C_2), \ldots, (C_q, C_x)$. 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 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. Now, we define a path $P_{x,y}$ in G_i between C_x and C_y as follows.

$$P_{x,y} = (C_x, v_{1,in}), (v_{1,in}, v_{1,out})[\text{ if } v_{1,in} \neq v_{1,out}](v_{1,out}, v_{2,in}), (v_{2,in}, v_{2,out})[\text{ if } v_{2,in} \neq v_{2,out}] \ldots (v_{q,out}, C_y)$$

It is easy to verify that all the edges used in any path of the above form were included in the graph G_i in Step 6. Also note that all edges of the form $(v_{j,in}, v_{j,out})$ are of zero cost since both the endpoints of any such edge are in the same component in $\mathcal{C}_i$. We form such a path $P_{x,y}$ between every pair of components in the pairing $\mathcal{P}$ between which the path in T_i has more than one edge. We then take the modulo two sum of all these paths [8] and add this to the DCS solution.

It is now routine to verify that the cost of the solution constructed this way is at most that of the set of edge-disjoint paths in T_i from which it was constructed. This in turn is at most OPT_b.

Each vertex $v \in V \subset V_i$ has degree at most b in T^* and we add at most one extra edge of zero cost for every edge of T^* incident on it in part (ii) of the construction above. Furthermore, taking the modulo two sum of these paths may only reduce the degree of any node in V. Thus the degree of any node in V in the constructed solution is at most $2b$. To verify that the degree of each node in V is even in the solution constructed, note that any node in this set is only used as an internal node in the set of paths identified in (ii) above. Since we used a modulo two sum of the paths in (ii), the degree of these nodes due to the set of edges added in the solution is even.

This completes the proof of Claim 3.7. □

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

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

This also completes the proof of Lemma 3.6. □

By Lemma 3.5, the degree of the output spanning tree is at most $O(b \log \frac{n}{b})$. The cost of the spanning tree is at most that of the solution subgraph F from which the tree is extracted in Step 12. This subgraph is the union of the DCS solutions obtained at different iterations. The cost of the DCS solution at each iteration is at most OPT_b by Lemma 3.6 and there are $O(\log \frac{n}{b})$ iterations by Lemma 3.4. Hence the bound on the cost follows. This proves Theorem 1.2.

4 Extensions to generalized connectivities and an application

In this section, we show how to extend Theorem 1.2 proved in Section 3 to more general one-connected networks like generalized Steiner trees.

4.0.4 Proof of Theorem 1.3

We sketch the algorithm for the generalized Steiner case. The algorithm follows the same outline as the algorithm in Section 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 must also modify the algorithm for the sake of proof as follows: If in an iteration, there are q active components, we add $\lfloor \frac{q}{3} \rfloor$ dummy nodes that have zero cost edges to all the active components in the auxiliary graph G_i. As a result, only $\frac{2q}{3}$ of the active components may be paired using the DCS solution found. However, this 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. So the performance guarantee of the algorithm remains unaffected.

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 3.6 and Claim 3.7) in each iteration. In the proof of Lemma 3.6, we start with the optimal generalized Steiner forest and contract the active components to single nodes. This may leave a forest where the q components are in $\frac{q}{3}$ trees with three active components each. Note that we can apply Claim 3.1 only to trees with an even number of nodes, so in this case, a single node remains in each tree. This node must be paired with a dummy node so as to exhibit a feasible solution to the DCS problem set up in G_i. This is the reason why we use so many dummy nodes in G_i. Since each active component must be part of a tree with at least one other active component, it is easy to see that adding $\frac{q}{3}$ dummy nodes always suffice to exhibit such a solution to the DCS problem set up in G_i from the optimal solution.

In the last iteration, when the number of active components falls to $O(b)$, we run the approximation algorithm for minimum-cost generalized Steiner forest of Agrawal, Klein and Ravi [1] 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 [1]. 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 1.3.

4.0.5 Proof of Theorem 1.5

Now we show how to apply Theorem 1.3 to approximate the problem of finding a minimum-degree generalized Steiner forest in an unweighted graph. 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 run the algorithm in Theorem 1.3 on the graph G with these weights and with the same set of site-pairs, but we run it for different values of the degree bound starting 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 1.3 outputs a tree of degree at most $O(b^* \log k)$ where k is the number of nodes in site-pairs. Furthermore, the cost of this tree is at most $O(n \log n)$ 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. From the discussion above, we know that when the degree bound is actually set to b^* we will obtain a generalized Steiner forest using edges of G of degree $O(b^* \log \frac{n}{b^*})$. This proves Theorem 1.5. Note that we may speed things up by doing a binary search on the values of the degree bound to converge on the smallest value such that the output solution uses only original edges in G.

5 Approximating both the degree and cost: the node-weighted case

The algorithms presented for minimum-degree edge-weighted network design do not extend directly to handle node-costs. In particular, we cannot model the subproblem at each phase as a DCS problem. Instead, we choose a series of greedy subgraphs to add to the solution. In this section, we turn to this node-weighted version of network design and prove Theorem 1.4.

5.0.6 Outline

The algorithm maintains a node-disjoint set of trees containing all the terminals. Initially, each terminal is in a tree by itself. As in Section 3, the algorithm works in $O(\log(\frac{k}{b}))$ phases. However, in each phase, instead of a DCS problem 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 active components 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.

5.0.7 The Algorithm

Input: An undirected graph $G = (V, E)$ with a cost $c(v)$ on each node v, a subset D of the nodes called terminals and a degree bound b.

Output: A Steiner tree spanning D in which the maximum degree is $O(b \log(\frac{|D|}{b}))$ and whose total cost is $O(OPT_b \log |D|)$.

1 Initialize the set of nodes in the solution $S := D$, the set of terminals, the set of edges in the solution subgraph $F := \emptyset$, and the iteration count $i := 1$.

2 Define a component in (S, F) to be *active* if it contains at least one terminal and does not contain at least one terminal.

3 While there are active components in (S, F)

4 Let $\mathcal{C}$ be the set of active components of (S, F). Let $\mathcal{C} = \{C_1 \ldots, C_q\}$. Let $G' := G$.

5 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.)

6 Let V' be the nodes in G'.

7 Construct an auxiliary complete bipartite graph G_i from G' as follows.

8 Add a node in G_i for every node in V' and one node for every $C_j \in \mathcal{C}$. Thus the node set V_i of G_i is $V' \cup \mathcal{C}$.

Assign costs to nodes in V' as follows. If $v \in S$, then set $c(v) = 0$. Otherwise, the cost of the node is as specified in the input graph G.

9 For every $v \in V'$ and $C_j \in \mathcal{C}$, there is an edge (v, C_j) in G_i. This edge is assigned cost $c(v, C_j)$ equal to that of a shortest path from v to any node in C_j in G' using node-costs. In computing these shortest paths we ignore the costs of the end-points. Thus for example, if $v \in C_u$ or if $(v, u) \in G'$ and $u \in C_u$, then the edge (v, C_u) has zero cost.

10 Find a node $v \in V'$ in the graph G_i minimizing the ratio

$$\min_{2 \le r \le b} \min_{\{C_1, \ldots, C_r\} \subseteq \mathcal{C}} \frac{c(v) + \sum_{j=1}^{r} c(v, C_j)}{r} \tag{1}$$

11 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 all the active components in $\cup_{j=1}^{r} C_j$ to a single component.
Update $\mathcal{C}$ by deleting all the merged components and adding the newly-formed component if it is active.

12 For every node $v \in V'$, if the degree of the node using edges added in this iteration i 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'.

13 The number of active components in (S, F) is now at most $\frac{11q}{12}$. So we move onto the next iteration setting $i := i + 1$.

14 Output (S, F) as the solution.

It is easy to implement Step 10. For each node v, define the *quotient cost* of v to be the minimum value of (1), taken over all subsets of the the current components. 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 distance $c(v, C_j)$ from v. 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 $2 \le j \le b$. 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 10 in polynomial time.

Next, we show that both the degree and the cost of all nodes in the solution output by the algorithm are not much more than those of a minimum-cost b-bounded Steiner tree.

5.0.8 Performance guarantee

We prove the performance guarantee using a series of lemmas.

Lemma 5.1 *The number of iterations of the algorithm is $O(\log(\frac{k}{b}))$ where $k = |D|$.*

The number of active components in the solution goes down by a constant factor in each iteration. Since there are k active components to begin with and $O(b)$ components in the beginning of the last iteration, the above lemma follows. The bound on the degree and the cost of the output solution follows from the following lemmas.

Lemma 5.2 *At any given iteration of the algorithm, the degree of any node due to edges added to the solution in this iteration is at most $O(b)$.*

This lemma directly follows from the way the algorithm works: In every iteration except the last, any node is deleted from consideration in the graph G' when its degree lies between $2b$ and $3b$ at Step 12. Note that the degree can never overshoot $3b$ since the degree of any node in the subgraph added in Step 11 is at most b. In the last iteration, the degree of any node increases by at most $O(b)$ since the subgraph added is acyclic and connects $O(b)$ components.

Now we turn to the cost of the subgraph added in an iteration. We shall prove the following lemma.

Lemma 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 at most $O(OPT_b)$, where OPT_b is the minimum-cost of a b-bounded Steiner tree spanning the terminals.*

Lemma 5.4 *The cost of the set of nodes added to the solution in the last iteration is at most $O(OPT_b \log b)$, where OPT_b is the minimum-cost of a b-bounded Steiner tree spanning the terminals.*

We prove Lemma 5.3 in the remainder of this section. Lemma 5.4 is proved similarly.

5.0.9 Spider decompositions

We employ the notion of spider decompositions introduced in [19] in showing that the each node chosen in Step 10 has small quotient cost with respect to the optimal solution.
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 node-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 5.5 (Klein and Ravi [19]) *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.*

5.0.10 Outline of the proof

We prove an averaging lemma and use this in conjunction with a potential function argument to prove Lemma 5.3. Fix an iteration i 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 fom G' in Step 12. We call these nodes *saturated* since they are saturated in terms of their degree. We first prove the following upper bound on the number of saturated nodes in an iteration.

5.0.11 Few nodes are saturated in an iteration

Claim 5.6 *At iteration i of the algorithm, the number of saturated 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 12, each saturated node has degree at least $2b$. Hence the sum of the degrees of all the saturated nodes is greater than q. The idea of the proof is to show that a large portion of this degree contributes to merging the q active components in the beginning of this iteration, using the fact that the subgraph added in this iteration is acyclic. We elaborate on this idea below.

Construct an auxiliary graph $H(i)$ for this iteration i as follows. 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 $\{C_1, C_2, \ldots C_q\} \cup V$. We now add all the edges added to the solution in this iteration in $H(i)$ as follows. Consider a set of such edges added in Step 11. These edges form a tree whose leaves are nodes in active components of the solution currently. A leaf can thus be a node in one of the active components $\{C_1, \ldots, C_q\}$ or in a component formed subsequently in this iteration by a merge. Consider the case of a leaf that is a node x in a component $C_x \in \{C_1, \ldots, C_q\}$ such that this component participates in a merge for the first time in this iteration. In this case, apart from adding the set of edges added in Step 11 to $H(i)$, we also add an edge in $H(i)$ from this leaf node $x \in V$ to the component node representing C_x. It is not hard to show that the graph $H(i)$ formed this way is acyclic and that the leaves of each tree in $H(i)$ are nodes in the set $\{C_1, \ldots, C_q\}$.

The set of saturated nodes in this iteration 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 assumed for a contradiction that there are more than $\frac{q}{2b}$ saturated nodes in this iteration. We shall 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 saturated nodes by $H_1, \ldots, H_z$. Let d_j denote the number of saturated 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 saturated node is at least $2b$ and so the degrees of all the saturated nodes in H_j sum to at least $2bd_j$. At most $2d_j$ of these edges are on paths between the d_j saturated nodes in H_j. 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 by construction, 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 made of edges added in this iteration, 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 and so this is a contradiction. Thus the total number of saturated nodes in iteration i beginning with q active components in the solution is at most $\frac{q}{2b}$. □

5.0.12 An averaging lemma

Let v be a node chosen in Step 10 in this iteration and let C denote the cost of the subgraph added subsequently in Step 11. Let this subgraph merge r trees. We prove the following claim.

Claim 5.7

$$r \geq \frac{5Cq}{12OPT_b} \tag{2}$$

Proof: Let T^* be a minimum-cost Steiner tree. Let $C_1, \ldots, C_p$ be the current 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. Note that $T^*(v)$ is connected and contains all supernodes.

Now consider deleting all edges incident to nodes in $V - V'$ in $T^*(v)$. These are nodes that were deleted from consideration in forming G' and G_i since they were saturated in degree. The number of such nodes is at most $\frac{q}{2b}$ by Claim 5.6 above. 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. Let M be the subset of supernodes that are in such subtrees with two or more supernodes. What we argued above is summarized in the following.

Proposition 5.8

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

We apply Theorem 5.5 to each of the subtrees of $T^*(v)$ with at least two supernodes to obtain a spider decomposition of M. The cost of all spiders in the decomposition is at most that of $T^*(v)$ which is in turn at most OPT_b.

To estimate the quotient cost of the node v chosen by the algorithm with respect to that induced by each spider in the decomposition, we must further decompose the spider decomposition so that the centers of the spiders are real nodes and thus have degree at most b. We do this next. Let $c_1, \ldots, c_s$ be the centers of the spiders in the decomposition. If c_j is a real node, then the degree of this node in T^* and hence in the decomposition is at most b. However, if the center is a supernode, we can partition the spider into many spiders, each centered at a real node contained in this supernode. The arms of the spider are partitioned among the real nodes based on the real node to which the arm is incident to in the supernode. Note that the spiders obtained this way are still nontrivial and the union of their feet contains the active components in M. Each of the spiders obtained this way now has degree at most b. 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 leaves, 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, each ℓ_j is at least two. 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 vertex v_j plus the sum of distances to these ℓ_j active components in the bipartite graph G_i be $Cost_j$. Hence the quotient cost of v_j in G_i is at most $\frac{Cost_j}{\ell_j}$.

Since the algorithm chooses a vertex of minimum quotient cost in G_i, 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

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

The sum $\sum_{j=1}^{t} \ell_j$ of the number of nodes of M in each of the spiders is at least $\frac{5q}{12}$ by Proposition 5.8. Also $\sum_{j=1}^{t} Cost_j$ is exactly the cost of the spider decomposition, which is at most OPT_b. Substituting in the above equation and simplifying yields the Claim. □

5.0.13 A potential function argument

Now we are ready to complete the proof of Lemma 5.3. Fix an iteration i and let the set of nodes chosen in Step 10 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

$$\phi_j = \phi_{j-1} - (r_j - 1) \tag{3}$$

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 5.7, we have

$$r_j \geq \frac{5C_j q}{12OPT_b} \geq \frac{5C_j \phi_{j-1}}{12OPT_b} \tag{4}$$

We now use an analysis technique due to Leighton and Rao [22] to complete the proof. Substituting Equation (4) into (3) and using the inequality $r_i \leq 2(r_i - 1)$, we get

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

Unraveling (5), 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

$$\sum_{j=1}^{f-1} C_j < 5OPT_b \ln \frac{12}{11} = O(OPT_b) \tag{6}$$

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 5.7 and noting that $r_f \leq q$ we have

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

Using the above equation and (6), 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 5.3.

Lemmas 5.1, 5.2, 5.3 and 5.4 together prove the performance guarantee in Theorem 1.4.

6 Algorithms under triangle inequality

In this section, we introduce the short-cutting technique used in proving Theorem 1.6. First, we review the short-cutting method introduced in [28].

6.1 Short-cutting for TSP

Rosenkrantz, Stearns and Lewis [28] 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 in the TSP tour over paths composed of already visited nodes in the Euler tour. By triangle inequality, the cost of the TSP tour so obtained can be shown to be 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 1.6. 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 problems. We present below the approximation algorithm referred to in Theorem 1.6.

6.2 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. This can be done by using any of the well-known algorithms [20, 26, 3].

2 Root the spanning tree at an arbitrary node r. Now every node except the root has a unique parent.

3 Partition the edges of the tree into "claws", namely, sets of edges going from every node that is not a leaf to its children in the tree. Sort the edges in every claw in the order of non-decreasing cost. Thus a typical node v has children $v_1, v_2, \ldots, v_d$ where d is one less than the degree of v in the tree (d equals the degree of node if $v = r$, the root). If we denote the cost of an edge (x, y) by $c(x, y)$, then we have for $1 \leq i < d$, the costs obey $c(v, v_i) \leq c(v, v_{i+1})$ as a result of our ordering.

4 We short-cut each claw locally as follows. For each node v that is not a leaf, we do the following.

5 If v has d children in the tree and $d > (b-1)$, then replace the edges $(v, v_2), \ldots, (v, v_{d-b+2})$ in the tree by the set of edges $(v_1, v_2), (v_2, v_3), \ldots, (v_{d-b+1}, v_{d-b+2})$.

6 Output the resulting spanning tree.

6.3 Proof of the b-MST Theorem

We now sketch a proof of Theorem 1.6. It is not hard to see that the short-cutting above produces a b-spanning tree. To bound the cost of the tree, we decompose the cost of the MST into the sum of the costs of the claws centered at each internal node of the MST. For any such claw of degree d we delete about $(d-b)$ edges incident on it and add edges between the children instead in Step 5. The cost of these added edges can be shown to be at most twice the cost of the deleted edges using triangle inequality. Using the fact that we deleted only the cheapest $(d-b)$ edges in each claw, we can prove the bound on the cost of the output tree.

To prove the bound on the bottleneck cost, observe that the MST is also a minimum bottleneck cost spanning tree. Since we used short-cuts that replace a path of length at most two in the MST, the bottleneck cost of the resulting tree is at most twice optimum. □

Next, we describe how to obtain spanning networks that are two-edge-connected and have small total cost, degree and bottleneck cost. Then we extend this algorithm to prove Theorem 1.7. Theorem 1.7 proves the case of $k = 2$ in Theorem 1.8. However, we use the TSP tour obtained in Theorem 1.7 to prove Theorem 1.8 in its full generality.

6.4 The algorithm for 2-edge-connected spanning networks

The algorithm for 2-edge-connected case works in two phases. In the first phase we construct a minimum spanning tree for the given graph. Next we use our b-MST algorithm to obtain a spanning tree of degree at most 3. The cost of this tree is no more than twice that of the MST. We then augment the tree using edges in the square of this tree to get a 2-edge connected subgraph. We show that this augmentation may increase the degree by at most one at each vertex.

Input: An undirected graph G with edge-costs satisfying the triangle inequality and a degree bound $b \geq 4$.

Output: A spanning 2-edge-connected subgraph in which the degree of any node is at most 4.

1 Find a 3-bounded spanning tree using algorithm in Section 6. Denote this tree by T_1. Note that every vertex that is not a leaf has either one or two children. We may assume without loss of generality that the root has two children.

2 Initialize the solution to be the tree T_1. Consider the vertices of the tree starting from the root in a breadth-first fashion.

3 **Case 1:** The vertex v has two children.
Let the children of the vertex v be v_1 and v_2. We add the edge (v_1, v_2) to the solution.

Case 2: The vertex v has a single child v_c in T_1. In this case add the edge (v_p, v_c), where v_p denotes the parent of v in T_1 and delete the edge (v, v_p).

4 Output the resulting subgraph.

Claim 6.1 *The subgraph obtained at the end of the algorithm is 2-edge connected and has maximum degree 4.*

Proof Sketch: First we prove that the degree of any node is at most 4 in the output subgraph. Since we started with T_1 which is 3-bounded, the degree of any node in T_1 is at most 3. During the course of the algorithm, we may add edges between nodes in Step 3. There are two cases. If the edge is added in Case 1, it increases the the degree of both its endpoints by 1. This is allowable since the degree may be 4 in the final subgraph. If the edge is added in Case 2, then note that we delete an edge incident to v_p on adding the edge (v_c, v_p) so this does not change the degree of v_p. However this may increase the degree of v_c by 1. But in this case, note that v_c has no sibling in the tree T_1 and so we do not add an extra edge to it in Case 1. So this increase is allowed for vertex v_c. Thus, the maximum degree of the output subgraph is at most 4.

We show that the output subgraph is 2-edge-connected by induction on the breadth-first levels of T_1. Since we can assume that the root has at least two children, the base case is the root and its two children joined by an edge and thus they form a cycle which is 2-edge-connected. Assume that after the algorithm has worked on all the nodes until level i, the solution constructed on these nodes is 2-edge-connected. We prove that the algorithm working on the nodes in level $i + 1$ maintains this property. There are two cases based on the running of the algorithm.

Case 1: If the vertex in level i has two children in the tree T_1, we add an edge between its children forming a cycle connecting the vertex and its children. This 2-edge-connects the two children in level $i+1$ with their parent in level i and hence with all nodes in levels i and below by the induction hypothesis.

Case 2: In this case we deleted an edge from the parent of the vertex v in level $i + 1$ to its grandparent in T_1, but added an edge from v to the grandparent. The vertex v can shown to have two edge-disjoint paths to its parent in the resulting graph. One of these is a direct edge added to its parent and the other path is using the added edge to its grandparent concatenated with a path from the grandparent

to the parent. The latter path exists since the graph until level i is two-edge-connected by the induction hypothesis and we deleted a single edge from this graph. Thus in this case also, the node v in level $i+1$ is two-edge-connected to all the nodes in levels i and below. □

We note that the subgraph produced by the algorithm above has a very nice structure. Namely, every edge has degree either two or four. Thus it can be decomposed into a set of cycles that intersect only at nodes of degree four. Moreover the cycles are connected to each other in a tree-like fashion. Thus, the degree-four nodes are cut-vertices in the solution. We shall use this structure to build a TSP tour by short-cutting this graph.

Theorem 6.2 *The cost of the output subgraph is no more than four times the minimum spanning tree on the same set of nodes and the cost of the maximum edge is also no more than four times the optimum.*

Proof Sketch: We first construct a 3-bounded MST T_1 whose cost is at most twice the cost of an MST of the input graph and whose bottleneck cost is twice optimum by Theorem 1.6. Then we use edges in the square of this tree to form the final output solution. We can show that the sum of the costs of the edges added minus the costs of the deleted edges is at most the cost of the tree T_1 itself and so the output subgraph has cost at most $2T_1$. This is at most four times the cost of the MST. Since the edges added to T_1 were edges in the square of T_1, the bottleneck cost of the output subgraph is at most twice that of T_1 which is at most four times optimum as claimed. □

6.5 TSP by short-cutting

We note that the 2-edge-connected subgraph output by the algorithm above has a very nice structure. Namely, every edge has degree either two or four. Thus it can be decomposed into a set of cycles that intersect only at nodes of degree four. Moreover, the cycles are connected to each other in a tree-like fashion, i.e., every pair of these cycles intersect in at most one degree-four node. These degree-four nodes are cut-vertices in the solution. We shall use this structure to build a TSP tour by short-cutting this graph.

We perform a local short-cut to reduce the degree of each of these degree-four nodes to two. For a degree-four vertex v, let v_1 and v_2 be two of its descendants in T_1 to which there are edges in the 2-edge-connected solution. Also, v is part of a cycle not using any of its descendants in T_1. Let the neighbors of v in this cycle be u_1 and u_2. We delete two of the edges, $(u_1, v), (v, v_1)$ incident on v and add the short-cut edge (u_1, v_1) instead. By triangle inequality, we have not increased the cost of the resulting subgraph, but we may have increased the bottleneck cost by a factor of two. Doing this local short-cuts at all the degree-four vertices yields a TSP tour with the bounds as claimed in Theorem 1.7.

6.6 Higher connectivities

Now we are ready to prove Theorem 1.8 in its full generality. The starting point is the TSP tour obtained in Theorem 1.7. 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 1.7, we can obtain a TSP tour T of cost $c(T)$ and bottleneck cost $b(T)$ such that $c(T) \leq 4c^*$ and $b(T) \leq 8b^*$. 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 $\frac{k}{2}$ of each other in the cycle (distance in the graph-theoretic sense: reachable using a path of at most k edges). Now it is easy to see that this graph is k-connected. The degree of every node in this graph is k. Since each shortcut 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 $\frac{k}{2} \cdot 8 = 4k$ of optimal.

The cost of the graph obtained this way is $\frac{k(k+2)}{2}$ times that of the TSP tour T that we started with. This in turn is at most $2k(k+2)c^*$. However, we can apply an approximate min-max relation between a MST and a packing of cuts in the graph that is derived in [12] in proving a better performance guarantee of $4(k+2)$ 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 $4(k+2)OPT_k$ as claimed in 1.8.

Given a graph G, each cut in the graph 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 *cocycle* 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* 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) of cost $w(u, v)$, the sum of the weights of all the cuts in this collection containing the edge is at most $w(u, v)$. The *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. The following theorem is a consequence of the results in [1, 12].

Theorem 6.3 *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.*

The algorithm in [12] greedily finds a packing of cuts and simultaneously builds a minimum spanning tree of weight at most twice the value of this packing.

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 6.4 *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 6.3 above we conclude that that $OPT_k \geq \frac{kc^*}{2}$.

7 Conclusions and Open problems

We have introduced several problems in network design involving multiple objectives and provided the first approximations for many of them. The most important open problems resulting from this work is to tighten up the performance ratios in Theorem 1.2 for both the cost measures in the edge-weighted case and the bound on the degree of the network in Theorem 1.4. It would also be interesting to investigate the limits of approximability for such problems and trade-offs in performance ratios for the different objective functions achievable by approximation algorithms.

8 Acknowledgements

We would like to acknowledge helpful conversations with P. N. Klein, B. Raghavachari, V. S. Ramakrishnan and S. Sairam.

References

[1] 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.

[2] A. Agrawal, P. Klein and R. Ravi, "How tough is the minimum degree Steiner tree? An approximate min-max equality (complete with algorithms)", TR-CS-91-49, Brown University (1991).

[3] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The design and Analysis of Computer Algorithms*, Addison Wesley, Reading MA., 1974.

[4] 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.

[5] 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.

[6] C. W. Duin, an A. Volgenant, "Some generalizations of the Steiner problem in graphs," *Networks, 17*, pp. 353-364, (1987).

[7] J. Edmonds, and E. L. Johnson, "Matching, Euler tours and the Chinese postman", *Math. Prog. 5*, (1973), pp. 88-124.

[8] 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).

[9] M. Fürer and B. 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.

[10] M. Fürer and B. 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.

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

[12] 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.

[13] M. Grötschel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag (1988).

[14] 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).

[15] F. K. Hwang and D. S. Richards, "Steiner tree problems," *Networks*, Vol. 22, No. 1, pp. 55-90 (1992).

[16] A. Iwainsky, E. Canuto, O. Taraszow, and A. Villa, "Network decomposition for the optimization of connection structures," *Networks, 16*, pp. 205-235, (1986).

[17] S. Khuller, B. Raghavachari, and N. Young, "Balancing Minimum Spanning and Shortest Path Trees," Tech. report CS-92-20, Penn. State Univ., August 1992 (To appear in the *Fourth Annual ACM-SIAM Symposium on Discrete Algorithms* (1993).

[18] 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.

[19] P. Klein and R. Ravi, "A nearly best-possible approximation for node-weighted Steiner trees," submitted to the *Integer Programming and Combinatorial Optimization Conference III (1993).*

[20] 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.

[21] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. Shmoys (eds.), *The Traveling Salesman Problem*, Wiley, Chichester, (1985).

[22] 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.

[23] L. Lovász and M. D. Plummer, *Matching theory*, Akadémiai Kiadó, Budapest (1986).

[24] C. Lund and M. Yannakakis, "On the Hardness of Approximating Minimization Problems," (manuscript), July 1992.

[25] R. G. Parker, and R. L. Rardin, "Guaranteed performance heuristic for the bottleneck traveling salesman problem," *Oper. Res. Lett. 6*, pp. 269-272, (1982).

[26] R.C. Prim, "Shortest connection networks and some generalizations", *Bell System Tech Journal*, 36(6), pp. 1389-1401, 1957.

[27] R. Ravi, B. Raghavachari, and P. N. Klein, "Approximation through local optimality: Designing networks with small degree," to appear in *Proceedings, Twelfth Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, December 1992, New Delhi, India.

[28] D. J. Rosenkrantz, R. E. Stearns and P. M. Lewis II, *An analysis of several heuristics for the traveling salesman problem*, *SIAM J. Computing*, 6(3), pp. 563-581, 1977.

[29] 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).

[30] P. Winter, "Steiner problem in networks : a survey," *BIT 25* (1985), pp. 485-496.

[31] R. T. Wong, "Worst case analysis of network design problem heuristics," *SIAM. J. Alg. Disc. Meth.*, 1., pp. 51-63, (1980).

[32] A. Z. Zelikovsky, "The 11/6-approximation algorithm for the Steiner problem on networks," *Information and Computation*, in press.

9 The Appendix

Here, we sketch related NP-completeness results and some simple hardness results that fall out from the reductions. Then, we examine approximation of Steiner trees under multiple objectives.

9.1 Hardness results

In this section, we prove NP-completeness of the problems we considered. We also prove hardness results about approximating some of these problems within certain factors under the triangle inequality and otherwise.

9.2 Degree-bounded problems are NP-complete

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 9.1 *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 an arbitrary fixed bound b. 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 $(b-1)$ additional nodes to the graph $u_1, \ldots, u_{b-1}, v_1, \ldots, v_{b-1}$. Add edges (u, u_i) and (v, v_i) for all $1 \leq i \leq (b-1)$. For every other vertex w in the graph, add $b-2$ additional vertices $w_1, \ldots, w_{b-2}$ and edges (w, w_i) for $1 \leq i \leq (b-2)$. 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 b-bounded spanning tree if and only if the original graph has a Hamiltonian path from u to v. □

9.3 Hardness of approximation

Theorem 9.2 *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 in the previous section 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. These cost assignments may not satisfy the triangle inequality. 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 9.2.

To prove the last part, we use the cost assignment as in the proof of Theorem 9.1 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 9.2. □

9.4 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 respective to the respective optima.

Proposition 9.3 *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.

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

Theorem 9.4 *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*

$$\frac{c(T') \cdot b(T')}{c^* \cdot b^*} \leq R_s \sqrt{n} \tag{7}$$

where n is the number of nodes in G.

Currently, the best known values of R_s are $\frac{11}{6}$ using the techniques of Zelikovsky [32] or something even less than that using the extensions of Berman and Ramaiyer [5].

9.4.1 Proof of Proposition 9.3

The example illustrating the Proposition 9.3 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 by 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 9.3.

9.4.2 Proof of Theorem 9.4

The algorithm for computing the Steiner tree which simultaneously minimizes the product of the total cost and the bottleneck cost given in the ratio (7) is simply this: Compute an approximately minimum-cost steiner tree using any of the known algorithms in the literature [32, 5]. 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 $c(T_1) \cdot b(T_1) \leq R_s c^* \cdot R_s c^*$. And so we get

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

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 $\tau \geq 1$,

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