Many birds with one stone: Multi-objective approximation algorithms

(Extended Abstract)

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 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 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.

[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 Grant CCR 89-03319.

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

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

1 Introduction

Several 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 specified subset of the nodes in the graph (a Steiner tree problem) or to only interconnect a set of site-pairs of nodes (a generalized Steiner forest 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 represent some notion of the price of using the network.

There are some classic examples of such cost measures. If we associate costs with edges and nodes that can be used to build the network, then we may seek a network such that the price of construction is minimized. This is the *minimum-cost network design* problem and has been well studied [1, 3, 10, 12, 15, 23, 25]. A notion of cost that reflects the vulnerability of the network to single point failures 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 [7, 8, 22].

Finding a network of sufficient generality and of minimum cost with respect to either of these measures is often NP-complete [9]. 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 [5, 13]. Recent papers have identified many such problems [2, 4, 14, 17, 20, 24] wherein more than one objective is specified in the statement of the problem.

In this paper, we study approximations of general network-design problems with multiple objectives:

minimizing the total cost of the network subject to a constraint on the degree of any node in the network. We also examine the case of cost functions obeying the triangle inequality and provide algorithms with better performance guarantees in this case.

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.

Observation 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 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 of maximum degree $O(b \log \frac{n}{b})$ and of cost at most $O(\log \frac{n}{b})$ 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*, a Steiner tree for the terminals is a subgraph spanning the terminals. Agrawal, Klein and Ravi [1] 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 [1], 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 1.2 as follows.

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 of cost 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.*

Goemans and Williamson [10], building on the work of Agrawal, Klein and Ravi [1], consider a class of constrained forest problems for which the generalized Steiner forest problem is a prototypical example. Our techniques directly extend to approximating degree-bounded minimum-cost networks of this general form. However we omit a description of these extensions in this abstract.

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.

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 solution. This problem then reduces to computing a minimum-degree spanning tree that has been well-studied [7, 8].

We focus our attention on the more interesting case of Steiner trees. Recently, Klein and Ravi [15] have presented the first polynomial-time approximation algorithm for node-weighted Steiner trees. The performance guarantee of their approximation algorithm is logarithmic in the number of terminals specified in the problem. Using recent results on the hardness of approximating the set-cover problem [19], they show that the performance guarantee is nearly best-possible (within a constant factor) unless $NP = \tilde{P}$. [7] We extend the techniques of Klein and Ravi [15] 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, of maximum degree $O(b \log(\frac{k}{b}))$ and of cost at most $O(\log k)$ 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

[7] Here we use $\tilde{P}$ to mean the complexity class Deterministic Quasi-polynomial time, or DTIME[$n^{\text{polylog } n}$].

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 1.2.

As before, we can extend the above theorem to generalized Steiner forests and to more general constrained forest problems addressed in [1, 10].

An application

Theorem 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 considered by Ravi, Raghavachari and Klein in [22]. In [22], they address the problem of finding two-edge-connected spanning subgraphs and one-connected networks of a general form (introduced in [10]) such that the maximum degree is minimum. They provide slightly super-polynomial approximation algorithms for these problems. A prototypical example of the one-connected network problem 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. The techniques in [22] can be adapted to provide polynomial-time approximation algorithms with performance ratio $\Omega(n^{\epsilon})$ for any constant $\epsilon > 0$. 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 in which the maximum degree of any node is at most $O(\delta^* \log \frac{k}{\delta^*})$. 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 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 1.6 *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 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 [23] 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 1.7 *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 four times the cost of a MST and of bottleneck cost at most eight times that of a minimum bottleneck-cost spanning tree.*

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 satisfying the triangle inequality, an integer $k \geq 2$ (the node-connectivity requirement), and an integer $b \geq k+1$ (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 $4(k+1)$ times that of the minimum-cost k-connected subgraph, and the bottleneck cost of the subgraph is at most $8k$ times that of a minimum bottleneck-cost spanning tree.*

This 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 1.2. We present the algorithm for degree-bounded node-weighted networks in the following section. Finally, we outline the algorithms for the problems under triangle inequality.

2 Related work

While there has been much work [1, 7, 8, 10, 11, 12, 21, 22, 25] 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 [4] 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 extend existing approximation algorithms for center problems to the balanced versions. Lin and Vitter [17] provide 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 they 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.

Other researchers have addresses multi-objective approximation algorithms for problems arising in areas other than network design. Agrawal, Klein and Ravi [2] 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 [14] provide 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 [24] study 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 present an approximation algorithm for minimizing the makespan of the schedule. Mitchell, Piatko and Arkin [20] study bicriteria optimization problems arising in computational geometry.

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

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

3.1 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 any) such that the degree of any vertex v in this subgraph is $f(v)$. This problem is also referred to as the f-factor problem [18], and is known to be polynomially solvable [6, 18]. The following variant of this problem is also known to be polynomially solvable using matching techniques [18]. Denote the degree of a node v in a subgraph H by $deg_H(v)$.

Fact 3.1 *[18] (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 $deg_H(v) \equiv 0 \pmod 2$.*

The b-bounded even DCS problem described above is a generalization of the T-join problem [6]. When b is allowed to be unbounded, this problem reduces to the T-join problem.

We now recall a tree decomposition result proved in [15].

Claim 3.2 *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.*

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 3.3 *Let H be any subgraph satisfying the conditions in Fact 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.

3.2 The approximation algorithm for b-MST

We now describe the algorithm referred to in Theorem 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 [7] 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, 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 $O(\log \frac{n}{b})$ iterations in all. We also ensure that in each iteration, 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.

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.

3.3 Performance Guarantee

We prove the performance guarantee using a series of lemmas.

At each iteration, applying Proposition 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 3.4 *The total number of iterations of the above algorithm is $O(\log \frac{n}{b})$.*

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

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

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 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.[8] Applying Claim 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) First, 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

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

$(u, v) \in E'$. 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) 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 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 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 [9] 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 3.6. □

Combining Lemmas 3.4, 3.5 and 3.6 gives Theorem 1.2.

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

In this section, we present the algorithm for node-weighted networks and prove Theorem 1.4.

4.1 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 3, the algorithm works in $O(\log(\frac{k}{b}))$ iterations. However, in each iteration, instead of a DCS problem we follow the approach of Klein and Ravi [15]: 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.

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

The Algorithm

```
1  Initialize $S$ to be the set of terminals, and $F$ to be the empty set.
2  Repeat while there are active components in $(S, F)$
3    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 $G' := G$.
4    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.)
5      Let $V'$ be the nodes in $G'$.
6      Construct an auxiliary complete bipartite graph $H$ from $G'$ as follows.
7        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 $G$.
8        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
         $$\min_{2 \leq r \leq b+1} \; \min_{\{C_1, \ldots, C_r\} \subseteq \mathcal{C}} \frac{c(v) + \sum_{j=1}^{r} c(v, C_j)}{r}$$
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 by Klein and Ravi in [15]. 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.

4.2 Performance guarantee

We prove the performance guarantee using a series of lemmas. As in Section 3, the following two lemmas can be easily proved.

Lemma 4.1 *The number of iterations of the algorithm is $O(\log(\frac{k}{b}))$ where k is the number of terminals.*

Lemma 4.2 *At any iteration of the algorithm, the degree of any node due to edges added in this iteration is at most $O(b)$.*

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

Lemma 4.3 *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 $O(OPT_b)$.*

Lemma 4.4 *The cost of the set of nodes added to the solution in the last iteration is at most $O(OPT_b \log b)$.*

We prove Lemma 4.3 in the remainder of this section. Lemma 4.4 is proved similarly.

4.3 Proof of Lemma 4.3

We prove an averaging lemma and use this in conjunction with a potential function argument due to Leighton and Rao [16] to prove Lemma 4.3. 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 4.5 *At any iteration of the algorithm, the number of nodes deleted from G' is at most $\frac{q}{2b}$.*

Proof Sketch: 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 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. 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. □

Spider decompositions

We employ the notion of spider decompositions introduced in [15] in showing that the each node chosen in Step 9 has small quotient cost with respect to the optimal solution.

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.

Theorem 4.6 (Klein and Ravi [15]) *Let G be a connected graph, and let M be a subset of its nodes such that $|M| \ge 2$. 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 4.7
$$r \ge \frac{5Cq}{12OPT_b} \tag{1}$$

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 4.5. 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 $p \ge \frac{11q}{12}$, at least $\frac{5q}{12}$ supernodes are in such trees.

Proposition 4.8 *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 4.6 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 $b+1$. It is easy to 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 spider. 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 from T^*, 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 4.8. 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 4.3. 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

$$\phi_j = \phi_{j-1} - (r_j - 1) \tag{2}$$

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 4.7, we have

$$r_j \geq \frac{5C_j q}{12OPT_b} \geq \frac{5C_j \phi_{j-1}}{12OPT_b} \tag{3}$$

We now use an analysis technique due to Leighton and Rao [16] to complete the proof as in [15]. Substituting Equation (3) into (2) and simplifying gives

$$\phi_j \leq \phi_{j-1}\left(1 - \frac{5C_j}{24OPT_b}\right) \tag{4}$$

Unraveling (4), taking natural logarithms and simplifying finally yields

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

This completes the proof of Lemma 4.3.

Lemmas 4.1, 4.2, 4.3 and 4.4 together prove the performance guarantee in Theorem 1.4.

5 Algorithms under triangle inequality

In this section, we present the short-cutting techniques used in proving Theorems 1.6 1.7 and 1.8.

The algorithm for an approximate b-MST

1 Find an MST of the given graph [3]. 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 $(d - b + 2)$ 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.

It is straightforward to verify that the short-cutting above produces a degree-b spanning tree. Using the fact that the cost of each claw is increased by at most the cost of $(d - b + 1)$ cheapest edges in it, 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 and we used short-cuts of length at most two.

Next, we describe how to obtain a spanning subgraph that is two-edge-connected and has small total cost, degree and bottleneck cost. No edge is allowed to be duplicated in this subgraph. We then short-cut this subgraph to obtain a TSP tour as described in 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.

The algorithm for 2-edge-connected spanning subgraphs

The algorithm for finding a 2-edge-connected spanning subgraph works in two phases. In the first phase we use our b-MST algorithm to obtain a spanning tree of degree at most 3. We then augment the tree using edges in the square of this tree to get a 2-edge connected subgraph.

1 Find a 3-bounded spanning tree of the input graph using the previous algorithm. Denote this tree by T_1. Note that every internal vertex has either one or two children and that the root has two children.

2 Initialize the solution to be the tree T_1. We loop through the vertices of the tree starting from the root in a breadth-first fashion. For each vertex v, we do one of the following.

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.

We can prove the following by a simple induction.

Claim 5.1 *The subgraph obtained at the end of the algorithm is 2-edge connected and has maximum degree 4.*

Theorem 5.2 *The cost of the output subgraph is no more than four times the cost of a minimum spanning tree and the maximum cost of any edge in the subgraph is no more than four times that of a minimum bottleneck-cost spanning tree.*

Using a more involved short-cutting procedure that bypasses the first phase of constructing a degree-three tree, we can improve the performance ratios in the above theorem to two for the cost and three for the bottleneck cost.

TSP by short-cutting

The 2-connected subgraph (call it G_2) obtained by the above algorithm has a nice structure: it can be viewed as a tree of cycles. Each vertex v with two children in T_1, can be uniquely identified with such a cycle that contains only descendants of v in T_1. Furthermore, these are exactly the vertices of degree four in G_2; all other vertices have degree two in G_2.

Using this structure we can perform two-edge-short-cuts on this 2-connected subgraph to obtain a TSP tour. We perform as many short-cuts as the number of degree-four nodes. Each short-cut involves two of the four edges incident on such a node. Also we can ensure that no two short-cuts involve the same edge of G_2. We omit further elaboration of this short-cutting for lack of space. Since the tour is derived by simply short-cutting G_2, the cost of the tour is at most that of G_2 by the triangle inequality. However, the bottleneck cost of the tour is at most twice that of G_2. Thus we prove the bounds in Theorem 1.7.

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 a MST and the minimum bottleneck-cost of a spanning tree of the input graph respectively. By Theorem 1.7, we can obtain a TSP tour of cost at most $4c^*$ and bottleneck cost at most $8b^*$. Let the vertices in this tour be numbered $v_1, v_2, \ldots, v_n$. We add edges to the tour as follows to convert it into a k-connected graph as follows: For each vertex v_i, add edges joining v_j to each of its neighbors to the right reachable from it using a path of at most k edges in the tour. Thus for each vertex v_i, we add the edges $(v_i, v_{(i+2) \bmod n}), \ldots, (v_i, v_{(i+k) \bmod n})$. It is easy to see that the resulting graph is k-connected. The degree of every node in this graph is $k + 1$. Since each short-cut employed replaces a path of at most k edges, the bottleneck cost goes up by this factor. This proves that the bottleneck cost of this subgraph is at most $8kb^*$.

The cost of the graph obtained this way is $\frac{k(k+1)}{2}$ times that of the TSP tour that we started with. This in turn is at most $2k(k + 1)c^*$. However, we can apply an approximate min-max relation between the cost of a MST and the value of a packing of cuts in the graph derived in [10] to obtain a better performance guarantee. In particular, if OPT_k denotes the minimum cost of any k-connected subgraph, we show that $OPT_k \geq \frac{kc^*}{2}$. This proves that the cost of the k-connected subgraph output by our algorithm is at most $4(k+1) \cdot OPT_k$ as claimed.

Acknowledgements

We thank P. N. Klein, B. Raghavachari, V. S. Ramakrishnan and S. Sairam for helpful discussions.

References

[1] A. Agrawal, P. Klein, and R. Ravi, "When trees collide: an approximation algorithm for the generalized Steiner tree problem on networks," *Proc., 23rd Annual ACM STOC* (1991), pp. 134-144.

[2] A. Agrawal, P. Klein, and R. Ravi, "Near-optimal nested dissection," submitted to *SIAM J. on Computing*. A preliminary version appeared as P. Klein, A. Agrawal, R. Ravi, and S. Rao, "Approximation through multicommodity flow," in *Proc. 31th Annual IEEE FOCS* (1990), pp. 726-737.

[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, pp. 343-354.

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

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

[7] M. Fürer and B. Raghavachari, "An $\mathcal{NC}$ approximation algorithm for the minimum degree spanning tree problem," *Proc., 28th Annual Allerton Conference* (1990), pp. 274-281.

[8] M. Fürer and B. Raghavachari, "Approximating the minimum degree spanning tree to within one from the optimal degree," *Proc., 3rd Annual ACM-SIAM SODA* (1992), pp. 317-324.

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

[10] M. X. Goemans and D. P. Williamson, "A general approximation technique for constrained forest problems," *Proc. 3rd Annual ACM-SIAM SODA* (1992), pp. 307-316.

[11] 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).

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

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

[14] S. Khuller, B. Raghavachari, and N. Young, "Balancing Minimum Spanning and Shortest Path Trees," *Proc, 4th Annual ACM-SIAM SODA* (1993), pp. 243-250.

[15] P. Klein and R. Ravi, "A nearly best-possible approximation for node-weighted Steiner trees," to appear in *Proc., IPCO III* (1993).

[16] F. T. Leighton and S. Rao, "An approximate max-flow min-cut theorem for uniform multicommodity flow problems with application to approximation algorithms," *Proc., 29th Annual IEEE FOCS* (1988), pp. 422-431.

[17] J.-H. Lin and J. S. Vitter, "ϵ-approximations with minimum packing constraint violation," *Proc., 24th Annual ACM STOC* (1992), pp. 771-782.

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

[19] C. Lund and M. Yannakakis, "On the Hardness of Approximating Minimization Problems," in these proceedings.

[20] 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.

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

[22] R. Ravi, B. Raghavachari, and P. N. Klein, "Approximation through local optimality: Designing networks with small degree," *Proc., 12th Annual Conf. on FST&TCS* (1992), LNCS 652, pp. 279-290.

[23] 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.

[24] D. B. Shmoys and E. Tardos, "Scheduling unrelated parallel machines with costs," *Proc., 4th Annual ACM-SIAM SODA* (1993), pp. 448-454.

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