# Approximation Algorithms for Degree-Constrained Minimum-Cost Network-Design Problems [4]

R. RAVI [1] MADHAV V. MARATHE [2] S. S. RAVI [3]
DANIEL J. ROSENKRANTZ [3] HARRY B. HUNT III [3]

**Abstract**

We study network-design problems with two different design objectives: the total cost of the edges and nodes in the network and the maximum degree of any node in the network. A prototypical example is the degree-constrained node-weighted Steiner tree problem: We are given an undirected graph $G(V, E)$, with a non-negative integral function $d$ that specifies an upper bound $d(v)$ on the degree of each vertex $v \in V$ in the Steiner tree to be constructed, nonnegative costs on the nodes, and a subset of $k$ nodes called *terminals*. The goal is to construct a Steiner tree $T$ containing all the terminals such that the degree of any node $v$ in $T$ is at most the specified upper bound $d(v)$ and the total cost of the nodes in $T$ is minimum. Our main result is a bicriteria approximation algorithm whose output is approximate in terms of both the degree and cost criteria – the degree of any node $v \in V$ in the output Steiner tree is $O(d(v) \log k)$ and the cost of the tree is $O(\log k)$ times that of a minimum-cost Steiner tree that obeys the degree bound $d(v)$ for each node $v$. Our result extends to the more general problem of constructing one-connected networks such as generalized Steiner forests. We also consider the special case in which the edge costs obey the triangle inequality and present simple approximation algorithms with better performance guarantees.

*AMS 1980 Subject Classification:* 68R10, 68Q25

*Keywords:* Approximation algorithms, Network design, Bicriteria problems.

[1] Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213-3890. Email: ravi@cmu.edu. Research supported by a NSF CAREER grant 96-25297.

[2] P.O. Box 1663, MS B265, Los Alamos National Laboratory, Los Alamos NM 87545. Email: marathe@lanl.gov. The work is supported by the Department of Energy under Contract W-7405-ENG-36.

[3] Department of Computer Science, University at Albany-SUNY, Albany, NY 12222. E-mail: {ravi, djr, hunt}@cs.albany.edu. Supported by NSF Grants CCR 89-03319, CCR 90-06396, CCR 94-06611 and CCR 97-34936.

[4] A preliminary version of this paper appeared as [25].

# 1 Introduction and Motivation

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 pairs of nodes (a generalized Steiner forest problem). The goal in such network-design problems can usually be expressed as minimizing some measure of cost associated with the network. Several examples of such cost measures have been considered in the literature. For example, 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 cost of construction is minimized. This is the *minimum-cost network design* problem and has been well studied. A notion of cost that reflects the vulnerability of the network to single point failures and the amount of load at a given point in the network is the maximum degree of any node in the network. Minimizing this cost gives rise to the *minimum-degree network design* problem, which has also been well studied. Another common cost measure is the maximum cost of any edge in the network. This goal falls under the category of *bottleneck problems* that have also received considerable attention.

Finding a network of sufficient generality and of minimum cost with respect to any one of these measures is often NP-hard [13]. Hence much of the work mentioned above focuses on approximation algorithms for these problems. However, in applications that arise in real-world situations, it is often the case that the network design problem involves the minimization of more than one of these cost measures simultaneously [9, 16].

In this paper, we concentrate on two objectives: (i) the degree of the network and (ii) the total cost of the network. Typically, our goal will be to find networks of minimum cost subject to degree constraints. For example, consider the following problem: Given an undirected graph $G = (V, E)$ with nonnegative costs on its edges 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 a minimum. Such *degree-constrained minimum-cost network* problems arise in diverse areas such as VLSI design, vehicle routing and communication networks. For example, Deo and Hakimi [8] considered this problem in the context of back-plane wiring among pins, where no more than a fixed number of wires can be wrapped around any pin on the wiring panel. In communication literature, this problem is commonly known as the *teleprocessing design problem* or as the *multidrop terminal layout problem* [2]. Here, we investigate the complexity and approximability of a number of such degree-constrained minimum-cost network-design problems. The main focus of our work is to develop a general technique for constructing near-optimal solutions to such problems.

The remainder of the paper is organized as follows. Section 2 contains basic definitions and formal statements of the problems considered in this paper. It also discusses a framework for evaluating approximation algorithms. Section 3 summarizes the results in the paper. Section 4 discusses related work. In Section 5 we present our algorithm for degree-bounded node-weighted networks. In that section we also discuss an extension of the algorithm to networks represented using proper functions. In Section 6, we outline the algorithms with improved performance and running times for constructing networks when restricted to input graphs obeying the triangle inequality. Section 7 contains negative

results on the approximabilities of some problems. Finally, Section 8 discusses some implications and directions for future research.

## 2 Basic Definitions and Problem Formulations

Following the framework developed in [21], a generic bicriteria network design problem, denoted by $(\mathbf{A}, \mathbf{B}, \mathcal{S})$, is defined by identifying two minimization objectives, denoted by **A** and **B**, from a set of possible objectives, and specifying a membership requirement in a class of subgraphs, denoted by $\mathcal{S}$. The problem specifies a budget value on the first objective (**A**) under one cost function, and the goal is to find a network having minimum possible value for the second objective (**B**) under another cost function, such that this network is within the budget on the first objective. The solution network must belong to the subgraph-class $\mathcal{S}$.

The two objectives we consider in this paper are: (i) degree of the network and (ii) the cost of the network. We consider two extensions of these objectives. The first extension deals with the budgeted objective, namely degree, and the second deals with the minimization objective, namely the total cost. The two versions of degree constraints that we consider are: (i) non-uniform degree (denoted by N-DEGREE) and (ii) uniform degree (denoted by U-DEGREE). In the *non-uniform degree* version, a possibly different degree bound $d(v)(\geq 2)$ is specified for each vertex $v$. The *uniform degree* version is a special case where $\forall v \in V, d(v) = b$ for some integer $b$; i.e., all the vertices have the same degree constraint. Thus, for the problems considered in this paper $\mathbf{A} \in \{$U-DEGREE, N-DEGREE$\}$. For the minimization objective, we focus on the total cost of the network. We assume we are given nonnegative costs on the edges and/or nodes of the input undirected graph. The *total cost* is given by the sum of the costs of all the edges (denoted by E-TOTAL-COST) or all the nodes (denoted by N-TOTAL-COST) in the network. Thus, $\mathbf{B} \in \{$N-TOTAL-COST, E-TOTAL-COST$\}$. Finally, the class of subgraphs $\mathcal{S}$ studied here includes SPANNING TREES, STEINER TREES, GENERALIZED STEINER TREES and networks specified using proper 0-1 functions introduced in [14].

Using the above notation, the problem of finding a minimum-cost spanning tree in which each node has degree at most $b$ is denoted by (U-DEGREE, E-TOTAL COST, SPANNING TREE). Similarly, given a node weighted graph $G(V, E)$, an integer function $d$ specifying the upper bound on the degree of each node and a set of terminals $\mathcal{T}$, the (N-DEGREE, N-TOTAL-COST, STEINER TREE) problem is to find a minimum-cost tree $T$ spanning the nodes in $\mathcal{T}$ such that the nodes in $T$ obey the degree constraints. Problems in which the desired network is a generalized Steiner forest or a graph specified by a proper 0-1 function can be formulated along similar lines.

Some of the problems considered in this paper also involve the maximum cost of any edge in the network, i.e., the bottleneck cost, as a minimization objective. We use E-BOTTLENECK-COST to denote this objective. For the rest of the paper, we use the term "$d(v)$-bounded network" to mean a network in which the degree of node $v$ is at most $d(v)$ for all $v$.

Most of the degree-constrained network-design problems considered in this paper are NP-hard. In fact, for several problems (e.g. (U-DEGREE, E-TOTAL COST, SPANNING TREE)) we show (Theorem 7.1) that it is NP-hard to find a solution that is within any factor of the optimal objective value, if the solution is required to satisfy the budget constraint; alternatively, if the solution

must achieve exactly the minimum value of the total cost objective, then it is NP-hard to find one which satisfies the budget within any given factor. Motivated by these hardness results for unicriterion approximations, we focus on finding bicriteria approximations, that is, efficient algorithms that guarantee a solution which is approximate in terms of both the budget and the objective function.

An $(\alpha, \beta)$ approximation algorithm for a generic bicriteria problem $(\mathbf{A}, \mathbf{B}, \mathcal{S})$ is a polynomial-time algorithm that produces a solution in which the objective value for $\mathbf{A}$ is at most $\alpha$ times the budget and the cost of the solution with respect to $\mathbf{B}$ is at most $\beta$ times the value of an optimal solution with respect to $\mathbf{B}$ that respects the budget constraint with respect to $\mathbf{A}$. Our algorithms provide bicriteria approximations in the sense described above for a wide variety of one-connected network-design problems.

# 3 Summary of Results

## 3.1 Hardness Results

Our lower bound results on finding near-optimal solutions include the following. Additional hardness results are discussed in Section 7.

1. For general graphs, unless P = NP, for any $\rho > 1$, there is no polynomial time $(1, \rho)$ approximation algorithm for the (U-DEGREE, E-TOTAL COST, SPANNING TREE) problem.

2. For general graphs, unless P = NP, for any $\rho > 1$, there is no polynomial time $(\rho, 1)$ approximation algorithm for the problem (U-DEGREE, E-TOTAL-COST, STEINER TREE).

3. For general graphs, unless P $=$ NP, for any $\varepsilon > 0$ and $\rho > 1$, there is no polynomial time $(2 - \varepsilon, \rho)$-approximation algorithm for the (N-DEGREE, E-TOTAL-COST, STEINER TREE) problem.

4. For general graphs, unless P $=$ NP, for any $\varepsilon > 0$ and $\rho > 1$ , there is no polynomial time $(\rho, \tau - \varepsilon)$-approximation algorithm for the (N-DEGREE, E-TOTAL-COST, STEINER TREE) problem. Here $\tau$ is the lower bound on the performance guarantee of any algorithm for finding minimum Steiner trees (see Chapter 10 of [15] for the best bounds). This result is an immediate corollary of hardness results for the minimum Steiner tree problem.

These hardness results motivate the need for bicriteria rather than unicriterion approximation algorithms for these problems.

## 3.2 Approximation Algorithms

A problem with costs on nodes as well as edges can be transformed (for the purposes of designing approximation algorithms) into one with only node costs as follows: subdivide each edge by introducing a new node with cost equal to the cost of the edge[5]. Therefore, in stating our approximation results, we focus on the node-weighted case. To keep the description of our main result simple, we present

[5]This transformation is not applicable to minimum cost spanning trees, for which the node weighted case is trivial.

below the result for the case of degree-constrained node-weighted Steiner trees. The extension of this theorem to more general classes of one-connected networks representable as cut-covers of proper functions is deferred to Section 5.7.

**Theorem 3.1** *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 $d(v) \geq 2$ for every node $v$, constructs a Steiner tree spanning all the terminals, with degree $O(d(v) \log k)$ at a node $v$ and of cost $O(\log k)$ times that of the minimum-cost Steiner tree of $G$ that spans all the terminals and obeys all the degree bounds.*

A proof of this theorem is provided in Section 5. The positive result presented in this theorem should be contrasted with the hardness results mentioned earlier stating that there is no $(2 - \varepsilon, \rho)$ or $(\rho, \tau - \varepsilon)$ (for any $\rho > 1$ and some $\epsilon > 0$) approximation algorithm for the (N-DEGREE, E-TOTAL-COST, STEINER TREE) problem unless P = NP. Combining the above observations we get that finding an approximation algorithm with performance guarantee $(2 - \varepsilon, \tau - \varepsilon)$ is NP-hard. Note that the performance guarantee on the node-cost in the above theorem cannot be asymptotically improved (even if the other performance ratio is arbitrarily weakened) since one of the problems included in the framework of Theorem 3.1 is the node-weighted Steiner tree problem considered by Klein and Ravi in [18]. By a reduction from the set cover problem and the known non-approximability results for the latter problem, they note that the best possible performance ratio achievable for this problem (even without the degree restrictions imposed in Theorem 3.1) is logarithmic unless P = NP [20, 3, 27]. As an immediate corollary of Theorem 3.1, we obtain an $(O(\log n), O(\log n))$ approximation algorithm for the (U-DEGREE, E-TOTAL COST, SPANNING TREE) problem introduced earlier.

In Section 6, we address the special case in which the edge costs obey triangle inequality and present *simple* approximation algorithms with better performance guarantees. Further, for the problem of constructing spanning networks in this special case, we show that our algorithms also simultaneously approximate yet another objective, namely the maximum cost of any edge in the network.

# 4 Related Work

Much work has been done on approximating each of the two cost measures that we simultaneously minimize (see [4, 5] and the references therein). We also refer the reader to the comprehensive book edited by Hochbaum [15] for recent results and techniques for solving these problems.

There has also been extensive work on bicriteria network design problems. The (U-DEGREE, E-TOTAL COST, SPANNING TREE) problem, originally posed and studied in [8], has been recently considered in Boldon, Deo and Kumar [4]. They present heuristics and their parallel implementations but do not provide worst case performance guarantees. Papadimitriou and Vazirani [23] studied the Euclidean version of this problem for the case when $d = 3, 4$. Monma and Suri [22] showed that for any set of points in the plane, a minimum spanning tree with $d = 5$ can be constructed efficiently. Khuller, Raghavachari and Young [17] gave approximation algorithms with performance guarantees of 3/2 and 5/4 for $d = 3$ and $d = 4$ respectively for points in the plane. They also presented an approximation algorithm with a performance guarantee of 5/3 for point sets in higher dimensions when

$d = 3$. Iwainsky et al. [16] formulated a version of the minimum-cost Steiner problem with an additional cost based on node-degrees. Duin and Volgenant [9] formulated the degree-bounded Steiner tree problem motivated by practical considerations. In other related work, Fischer [11] considered the problem of finding a MST of minimum possible maximum degree in a weighted undirected graph. He showed that the techniques of Fürer and Raghavachari [12] can be applied to find a MST of approximately minimum degree.

In [25], we presented early versions of the results in this paper giving specific algorithms for the edge-cost versions, and using a simpler version of the techniques in this paper to give results for the uniform degree node-weighted versions. Building on our work there, in [21], we studied other bicriteria network design problems. There we also presented a polynomial-time algorithm for the (U-DEGREE, E-TOTAL COST, SPANNING TREE) problem when inputs are restricted to treewidth-bounded graphs. In [24], Ravi has applied some of the ideas here to solve a bicriteria problem that forms the basis for finding an approximately minimum broadcast-time scheme in an arbitrary graph.

# 5 Degree-Constrained Node-Weighted Steiner Trees

In this section, we present our algorithm in detail for the degree-constrained node-weighted Steiner tree problem. In Section 5.7, we briefly indicate how the algorithm can be extended to accommodate more general connectivity specifications.

Recall that, as input to the problem, we are given an undirected graph $G(V, E)$, with nonnegative costs on the nodes and a set of *terminals* to be connected together into a Steiner tree. In addition, for each vertex $v$, a budget $d(v)$ on its degree in the Steiner tree is specified. The goal is to find a Steiner tree of minimum node cost that obeys the degree constraint at every node. There are no edge costs in this version since the problem with node and edge costs can be transformed into one involving just node costs (see Section 3.2). We shall assume for the sake of simplicity that such a Steiner tree always exists on the input graph and address the problem of computing one that approximately obeys the degree budgets as well as minimizes the total node cost. In the description of the algorithm and its analysis, we use $c_G(v)$ to denote the cost of a node $v \in V$. We omit the subscript $G$ when there is no ambiguity.

## 5.1 High Level Description

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 whose union contains all the terminals. Define a connected component of $(S, F)$ to be *active* if it contains at least one terminal but not all of the terminals. The algorithm works in $O(\log k)$ iterations. In each iteration, we run a greedy algorithm to choose a subgraph (a collection of many smaller subgraphs called *spiders*) of small degree and small node-cost such that the addition of this subgraph to the current solution reduces the number of connected components of $(S, F)$ by a constant factor.

We first define a few additional terms used in describing our algorithm. We use $OPT$ to denote the minimum cost of any Steiner tree that obeys the degree restrictions in the input.

ALGORITHM-DEGREE-STEINER:

**Input**: An undirected graph $G(V, E)$ with nonnegative costs on its nodes, a set $\mathcal{T} \subseteq V$ of terminals (where $|\mathcal{T}| = k$), and a function $d$ assigning nonnegative values (each value is at least two) to the nodes of $G$. Let $b = \min_v\{d(v)\}$.

**Output**: A Steiner tree $T$ spanning the terminals $\mathcal{T}$ such that the degree of any node $v$ in $T$ is at most is $O(d(v) \log k)$ and the cost of $T$ is at most $O(\log k)$ times that of a minimum-cost degree-constrained Steiner tree spanning the terminals $\mathcal{T}$.

1 Initialization: $S = \mathcal{T}$ and $F = \phi$.

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 $q = |\mathcal{C}|$. Set $G'(V', E') := G(V, E)$.

4 **While** $|\mathcal{C}| \geq 11q/12$ **and** $q > 6$ **do**

5 Construct an auxiliary graph $H$ as follows: Starting with $G(V, E)$ delete the nodes in $V - V'$ to get a graph $G'$ on $V'$. For every component surviving (as active) in $\mathcal{C}$, contract all nodes within this component occurring in $G'$ to a single supernode.

6 **For** every node $v \in V'$, consider $v$ as the center of a spider.

7 If $v$ is in a supernode of $H$, then uncontract $v$ from this supernode and attach a zero-cost edge between them; if no nodes from $V'$ remain in the supernode after uncontracting $v$, then add a new dummy supernode to $H$ representing the active component containing $v$ and a zero-cost edge to it from $v$.

8 **For** $j = 2$ to $d(v) + 1$ **do** Find a minimum-cost spider centered at $v$ in $H$ with $j$ supernodes as its feet using PROCEDURE-FIND-SPIDER.

9 Among all the spiders produced in Step 6, choose one of minimum ratio-cost, defined as the ratio of the cost of all the real nodes in the spider to the number of feet in it.

10 Let $v$ be the center node and $C_1, \ldots, C_r$ be the components in $\mathcal{C}$ chosen as the feet of the spider in Step 9. Let $P_1, \ldots, P_r$ be the legs of the spider connecting $v$ to $C_1, \ldots, C_r$ respectively. Add $\cup_{a=1}^{r} P_a$ to the current solution $(S, F)$ so as to merge $C_1, C_2, \ldots, C_r$ into one active component. Update $\mathcal{C}$.

11 For every node $v \in V'$, if the degree of this node using edges added so far in this iteration (Steps 5 through 11) is between $2d(v)$ and $3d(v)$, then update $V' = V' - \{v\}$.

12 **If** $q \leq 6$ **then**

13 **Repeat** while there are active components

14 Run Steps 5 to 10.

15 Set $V' = \phi$.

16 **else** Goto Step 2. ($|\mathcal{C}|$ is now less than $\frac{11q}{12}$.)

17 Output $(S, F)$ as the solution.

**Definition 5.1** *[18]*
*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. The leaves of the spider are also called the* feet *of the spider. A* nontrivial spider *is one with at least two feet.*

## 5.2 The Algorithm and its Performance Guarantee

The rest of Section 5 is devoted to describing the algorithm and its performance for approximately solving the (N-DEGREE, E-TOTAL-COST, STEINER TREE). ALGORITHM-DEGREE-STEINER gives the details of the entire algorithm.

## 5.3 A Procedure to Find Minimum Ratio Spiders

The heart of ALGORITHM-DEGREE-STEINER is Step 8 — a procedure that chooses a nontrivial spider of minimum "ratio-cost". We describe this procedure informally. Consider a generic step of ALGORITHM-DEGREE-STEINER (Step 4). Observe that we maintain a current graph $G'$ and the current partial solution $(S, F)$. Let the connected components of $(S, F)$ be denoted by $\{C_1 \ldots, C_q\}$. The spider we use to merge these components must have a real node of $G'$ as the center and some of these components as its feet. During the course of an iteration, we may delete a node $v$ from $G'$ if the degree of $v$ due to the addition of edges in the generic step is between $2d(v)$ and $3d(v)$; i.e., a constant factor of the degree bound for $v$. We must then choose in the current graph $G'$ a spider of minimum ratio-cost, namely the ratio of the cost of all the nodes of $G' - S$ in the spider and the number of feet of the spider.

Although the concept of a spider is similar to the one used in [18], the degree constraint makes the problem of finding a "good spider" harder. As a result, the procedure in [18] for finding spiders cannot be used in place of PROCEDURE-FIND-SPIDER described below.

We find a spider of minimum ratio-cost by using several calls to a minimum-cost flow algorithm on the auxiliary graph $H$. We describe how to find a minimum ratio spider centered at a specific node $v \in G'$, the current graph. By trying all nodes, we can choose the overall minimum ratio spider. To find a minimum ratio spider centered at $v$, it suffices to find a spider centered at $v$ containing exactly $j$ feet such that it has minimum total node cost. By trying all values of $j$ in the set $\{2, 3, \ldots, d(v) + 1\}$, we can find the value of $j$ minimizing the ratio cost of the resulting spider for $v$. PROCEDURE-FIND-SPIDER given below describes a method to find a minimum node-cost spider centered at $v$ with exactly $j$ feet.

PROCEDURE-FIND-SPIDER:
**Input**: An undirected graph $H$ containing real nodes and supernodes, a real node $v$ as the center and a number $j$ specifying the number of feet in the spider to be constructed.
**Output**: A minimum-cost spider centered at $v$ with $j$ feet that are supernodes.

1 Bi-direct all the undirected edges in $H$ giving each resulting arc the cost of the node at its tail. (Supernodes have zero cost.)

2 Reassign the cost of all the arcs leaving the center node $v$ to be $\frac{c(v)}{j}$.

3 Attach a new sink node $t_v$ with new arcs of zero-cost coming to it from all the supernodes.

4 In this digraph, impose a capacity bound of one unit on all nodes except $v$ and $t_v$ and find a minimum-cost flow of value $j$ from $v$ to $t_v$.

**Remarks:**

1. The solution to the above flow problem (when feasible) can be found in polynomial time and is integral (see [2] or Chapter 4 of [6]).

2. Such a flow gives a minimum-cost set of node-disjoint paths originating at $v$ and ending at a set of $j$ supernodes.

3. The cost of real nodes in $H$ other than $v$ that occur in flow paths are accounted for in the cost of the arcs leaving them. Node $v$ has exactly $j$ arcs leaving it in the flow solution, each of cost $\frac{c(v)}{j}$ for a total of $c(v)$. Thus, the total cost of the flow solution is equal to the cost of all the real nodes in the spider that are not in any component of $(S, F)$.

4. The set of edges in the solution to the flow problem contains no cycles. Consequently, the set of undirected edges from the original graph that correspond to these flow paths (i.e., ignoring the arcs into $t_v$) contain no cycles.

We now prove the claimed performance guarantee of the algorithm. For ease of exposition the proof is broken down into a sequence of lemmas and theorems.

**Proposition 5.2** *The number of iterations of Step 2 in the algorithm is $O(\log k)$ where $k$ is the number of terminals.*

The above proposition follows by observing that in each iteration of Step 4, we reduce the number of active components by a constant factor. We start with $k$ components and the last iteration runs to completion when this number drops to 6 or below.

**Proposition 5.3** *For each node $v$, the increase in the degree of $v$ in $(S, F)$ due to edges added in one iteration of Step 2 is at most $3d(v)$.*

**Proof:** Consider a node $v$ and fix an iteration $i$ (Step 4). If degree of $v$ exceeds $2d(v)$ using edges in this iteration, then it is deleted from further consideration in Step 11 and no more edges are added in this iteration that are adjacent to it. Furthermore, in Step 8 the increase in degree of $v$ is either (i) at most $d(v) + 1$ if it is the center of the chosen spider or (ii) at most 2 which is in turn at most $d(v)$ if it is a non-center node of the chosen spider (since for all $v$, $d(v) \geq 2$). Thus, if the degree of $v$ is no less than $2d(v)$ to begin with, it never exceeds $3d(v)$ after executing Step 8. In the last iteration, we merge at most 6 components using an acyclic set of edges. Thus, the degree of $v$ increases by at most $6 \leq 3d(v)$, since for all $v$, $d(v) \geq 2$. □

Combining Propositions 5.2 and 5.3 immediately leads to the performance guarantee on the degree of a node in the final solution. We now bound the total cost of the subgraph added in one iteration. Lemma 5.4 along with Proposition 5.2 yield the required performance guarantee on the total cost of the final solution, completing the entire proof.

**Lemma 5.4** *The cost of the set of nodes added to the solution in each iteration of Step 2 is at most* $O(OPT)$.

First we complete the proof with regard to the the cost added in the last iteration. Recall that at the beginning of the last iteration, the number of active components is at most 6. For this iteration, our algorithm reduces to that of Klein and Ravi [18] for node-weighted Steiner trees. Hence using their result with the number of "terminals" to be connected being at most 6, the cost of the nodes added is at most $O(OPT \log 6) = O(OPT)$.

The proof of the lemma for the remaining iterations is more involved and is described in Sections 5.4 through 5.6. The proof proceeds by deriving a decomposition of an optimal solution and using it as a witness to the performance of the algorithm in each iteration. In particular, we use the decomposition to prove an averaging lemma and use this in conjunction with a potential function argument due to Leighton and Rao [19] to prove Lemma 5.4. We begin by proving a bound on the total degree of all the nodes that are deleted from $G'$ in any iteration.

## 5.4 Bounding the Total Degree of Deleted Nodes

Fix an iteration $i$. Let the active components in the beginning of this iteration $i$ be $C_1, C_2, \ldots, C_q$. At 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.

**Lemma 5.5** *In each iteration of Step 2 of the algorithm, the sum of the degrees of all the nodes deleted from $G'$ due to edges added in this iteration is at most $q$.*

The proof relies on the following observations.

1. The subgraph added in a given iteration is acyclic.
2. The iteration terminates when at most $\frac{q}{12}$ of the active components are merged using edges added in a given iteration.

Using these observations we can show that a large fraction of the degree of the deleted nodes contributes to merging the $q$ active components. This implies an upper bound on the sum of the degrees.
**Proof:** Let $m$ be the number of components that were merged in this iteration. Note that $m \leq \frac{q}{12}$. We can assume without loss of generality that the $m$ components are merged into a single component. (It is easy to see that in other cases we obtain better bounds.)

Let $\mathcal{R}$ denote the acyclic subgraph added to merge the $m$ components. By the working of the algorithm, the leaves of $\mathcal{R}$ are precisely the $m$ components that were merged. By our assumption, $d(v) \geq 2$ for all $v$. Thus, all vertices of degree 2 in $\mathcal{R}$ do not contribute to the degree sum of deleted nodes. Hence we modify $\mathcal{R}$ to obtain $\mathcal{R}'$ as follows: We contract all simple paths in which each internal node has degree 2 into a single edge. Now each internal node in $\mathcal{R}'$ has a degree of at least 3. Let $\mathcal{N} = \{w_1, w_2, \ldots, w_P\}$ denote the internal nodes of $\mathcal{R}'$. Note that some of these nodes might not have been deleted. Let $\mathcal{D}(w_i)$, $1 \leq i \leq P$ denote the degree of $w_i$ in $\mathcal{R}'$. We now prove a stronger statement and show that

$$D = \sum_{i=1}^{P} \mathcal{D}(w_i) \leq q \tag{1}$$

Note that $1 \leq i \leq P,\ \mathcal{D}(w_i) \geq 3$. Thus $D \geq 3P$. But since $\mathcal{R}'$ is a tree we know that the number of edges $|E(\mathcal{R}')|$ is given by $|E(\mathcal{R}')| = P + m - 1$. Thus

$$2|E(\mathcal{R}')| = 2(P + m - 1) = D + m \geq 3P + m \tag{2}$$

implying that $P \leq m - 2$. This gives an upper bound on the total number of internal nodes in $\mathcal{R}'$.

$$D + m = 2|E(\mathcal{R}')| = 2(P + m - 1) \leq 2(m - 2 + m - 1) \leq 3m - 5 \tag{3}$$

Combining this with the upper bound on $m$ we get

$$D \leq 2m - 5 \leq 2\frac{q}{12} - 5 \leq \frac{q}{6}$$

proving Equation (1). □

### 5.5 Spider Decompositions and an Averaging Lemma

We employ the notion of spider decompositions introduced by Klein and Ravi [18] in showing that the each node chosen in Step 9 has small ratio-cost with respect to the optimal solution.

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 and the centers of the spiders in the decomposition contains $M$.

**Theorem 5.6 ([18])** *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 $M$.* □

Let $v$ be a node chosen in Step 9 of the algorithm. 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 5.7**

$$r \geq \frac{5}{12}\frac{Cq}{OPT} \tag{4}$$

**Proof:** Let $T^*$ be a minimum-cost degree-bounded Steiner tree of cost $OPT$. Let $C_1, \ldots, C_p$ be the active components when a spider centered at node $v$ was chosen by the algorithm. 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. We then remove edges from $T^*(v)$ so as to make it acyclic; thus $T^*(v)$ is a tree.

Delete all edges incident on nodes in $V - V'$ (the deleted nodes) in $T^*(v)$. Consider a node $u$. By construction of $T^*(v)$, $u$'s degree in $T^*(v)$ (denoted by $d_T(u)$) is at most $d(u)$. Furthermore, $u$ is deleted in our algorithm only if its degree, denoted by $d^i(u)$, exceeds $2d(u)$ due to the edges added in a given iteration of Step 2. Thus we have

$$\forall u \in V - V', d^i(u) \geq 2d(u) \text{ and } d_T(u) \leq d(u)$$

Combining these observations with Lemma 5.5, we get

$$\sum_{u \in V - V'} d_T(u) \leq \frac{1}{2} \sum_{u \in V - V'} d^i(u) \leq q/2.$$

Thus, the total number of edges deleted from $T^*(v)$ is also at most $\frac{q}{2}$. Since there were $p$ active components (and hence supernodes) when $v$ was chosen, the tree $T^*(v)$ has $p$ supernodes in it. Since we deleted at most $\frac{q}{2}$ edges from this tree, 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. We summarize this in the following proposition.

**Proposition 5.8** *Let $M$ denote the subset of supernodes that are in subtrees with two or more supernodes. Then $|M| \geq \frac{5q}{12}$.* □

We apply Theorem 5.6 to each subtree of $T^*(v)$ with at least two supernodes to obtain a spider decomposition of $M$. We now compare the ratio cost of spider chosen by the algorithm with that of each spider in the decomposition. To do this however, we must ensure that the following two conditions hold.

(i) the center of each spider in the decomposition must be a real node (not a supernode) and

(ii) the number of legs of each spider must be at most $d(v) + 1$.

We achieve this as follows. We further partition a spider centered at a supernode into many nontrivial spiders each centered at a real node $v$ contained in this supernode such that the union of their feet contains the feet of the original spiders and the number of legs of the spider centered at $v$ is at most $d(v) + 1$. To do this, first consider all the real nodes in the central supernode with at least one leg of the spider incident on them. Each such real node can be made the center of a nontrivial spider (satisfying (i)) with all the legs incident on it as the legs of the spider, along with a zero cost leg to the

supernode that it belongs to. Since the degree of any real node in $T^*$ is at most $d(v)$, the number of legs of any such spider is at most $d(v)+1$ satisfying (ii).

Let the centers of the resulting spider decomposition satisfying (i) and (ii) be the set of real nodes $v_1, \ldots, v_t$. Let $\ell_1, \ldots, \ell_t$ denote the number of nodes of $M$ (feet) in each of these spiders respectively. Since every spider in the decomposition is nontrivial and is derived as above, each $\ell_j$ is at least two and at most $d(v)+1$. Moreover, a spider with center $v_j$ induces a subset of the current active components, namely the $\ell_j$ 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 $\ell_j$ components – if $v_j$ is already in a supernode, we may assume its cost to be zero since it has already been paid for in the formation of the supernode) be $Cost_j$. Then the ratio cost of the spider centered at $v_j$ in the auxiliary graph $H$ constructed in this loop is at most $\frac{Cost_j}{\ell_j}$.

Since the algorithm chooses a spider of minimum ratio-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 decomposition yields

$$\sum_{j=1}^{t} Cost_j \geq \frac{C}{r} \sum_{j=1}^{t} \ell_j. \tag{5}$$

Combining Proposition 5.8 with the observation that the union of the feet of the spiders contains $M$, we get

$$\sum_{j=1}^{t} \ell_j \geq |M| \geq \frac{5q}{12}. \tag{6}$$

Also note that

$$\sum_{j=1}^{t} Cost_j \leq COST(T^*(v)) \leq OPT \tag{7}$$

since (i) the cost of the nodes in the tree $T^*(v)$ is at most $OPT$ and (ii) each real node in $T^*(v)$ appears in at most one spider. Combining Equations (5), (6) and (7) yields Claim 5.7. □

## 5.6 A Potential Function Argument

Now we are ready to complete the proof of Lemma 5.4. 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 $\phi_j$ 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 $(S, F)$, $\phi_{f-1} > \frac{11q}{12}$ and $\phi_f \leq \frac{11q}{12}$. Let the number of trees merged using vertex $v_j$ be $r_j$. Then we have

$$\phi_j = \phi_{j-1} - (r_j - 1) \tag{8}$$

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

$$r_j \geq \frac{5}{12} \frac{C_j q}{OPT} \geq \frac{5}{12} \frac{C_j \phi_{j-1}}{OPT} \tag{9}$$

We now use an analysis technique due to Leighton and Rao [19] to complete the proof as in [18]. Substituting Equation (9) into (8) and simplifying using $r_j \geq 2$ gives

$$\phi_j \leq \phi_{j-1}(1 - \frac{5}{24}\frac{C_j}{OPT}) \tag{10}$$

Simplifying (10), we obtain

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

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

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

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

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

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

$$C_f \leq \frac{12}{5} OPT.$$

Using the above equation and (11), we have that the cost of the set of nodes added in this iteration is

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

This completes the proof of Lemma 5.4.

The performance of our approximation algorithm was summarized in Theorem 3.1.

### 5.7 Extension to Proper Function Cut Covers

The extension of Theorem 3.1 to construct cut-covers defined by proper 0-1 functions is fairly straightforward, and the algorithm for this case follows the same outline as the one above. The reader is referred to [14, 15] for the definition of proper 0-1 functions. The algorithm begins with the set $S$ being the set of terminals defined by the proper function. The definition of active components in the algorithm is now based on the $f$-values given to cuts by the input proper function. In other words, a component is deemed active if the cut around it is. Note that when all components are inactive, the set of edges added by the algorithm until then constitutes a feasible cut-cover.

The only additional issue is that in the proof of the upper bound on the cost of the subgraph added in each iteration, the optimal solution is a forest instead of a single tree. However, as in [18], we

can use the fact that each tree in the forest must contain at least two active components to infer that this forest contains at least as many edges as half the number of active components. This observation is sufficient to prove a modified version of Claim 5.7 with slightly worse constants. The details are straightforward and omitted to avoid repetition. Thus we have the following theorem.

**Theorem 5.9** *There is a polynomial-time algorithm that, given an undirected graph $G$ with nonnegative costs on its nodes, a proper function $f$ defined on the node subsets of $G$, and a function $d$ assigning a nonnegative value $d(v) \geq 2$ to each node $v$ of $G$, constructs a cut-cover for the family of cuts defined by $f$ in which the maximum degree of any node $v$ is at most $O(d(v) \log k)$ and the cost of the cover is at most $O(\log k)$ times that of the "minimum-cost degree-constrained cut cover" for $f$. Here $k$ represents the number of terminals defined by $f$. A degree-constrained cut cover is a subgraph which covers (i.e., contains at least one edge in) all the cuts defined by $f$ and has degree at most $d(v)$ at node $v$, for all $v$.* □

# 6 Algorithms Under Triangle Inequality

One way to circumvent the difficulty of approximating the problems studied is to consider more structured cost functions on the edges. In this direction, we turn to the case where 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 approximation algorithms that strictly conform to the degree restriction in the input problem and approximate the bottleneck cost of the output network as well. Most of the results in this section are straightforward and we discuss it here for the sake of completeness.

## 6.1 Results for Spanning Trees

**Proposition 6.1**

1. *There is a polynomial time approximation algorithm for (*N-DEGREE, E-TOTAL COST, SPANNING TREE*) problem restricted to edge-weighted graphs that satisfy triangle inequality. Its performance guarantee is $(1, (2 - \frac{(d_{\min}(v)-2)}{(n-1)}))$. Moreover, the bottleneck cost of the tree produced by* ALGORITHM-TI-SPANNING-TREE *is at most twice that of the minimum-bottleneck spanning tree. Here $d_{\min}(v)$ denotes the smallest degree constraint.*

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

**Proof:** First we sketch the proof of Part 1. The algorithm starts by constructing an MST. It then partitions the edges of the MST into claws and sorts the edges in every claw in the order of non-decreasing cost. Each claw is short-cut locally by replacing edges from the internal node to its children (except the very first child) with edges between consecutive children. Let $T$ denote the resulting tree.

To prove the first part of the proposition, for any set $E'$ of edges, let $c(E')$ denote the sum of the costs of all the edges in $E'$. We have the following relations.

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

For an internal node $v$, let $t(v)$ denote the number of children of $v$ in the rooted MST. For the solution $T$, we have

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

By triangle inequality on the costs $c$, we have

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

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

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

Since the cost of any $d(v)$-bounded spanning tree is at least as much as that of the MST, this gives the bound on the cost of the tree output by the algorithm.

We now complete proof by proving the bound of two on the bottleneck cost. It is well known that an MST is also an optimum bottleneck spanning tree. Since each short-cut used in forming the output tree $T$ is made up of at most two edges, the bottleneck cost of $T$ is at most twice that of the MST. Since the bottleneck cost of any $b$-bounded spanning tree is at least as much as that of the bottleneck spanning tree, the resulting tree has bottleneck cost at most twice the optimum.

Part 2 of the proposition follows from standard constructions based on a recursive short-cutting procedure using edges from the cube of the Minimum Spanning Tree. This is also hinted at in [7] (see problem 37.2-3 on page 975).

□

## 6.2 Extension to Higher Connectivities

Now we are ready to prove our result for networks with higher connectivities. The result is proved by using short-cuts that induce higher-connected graphs.

**Theorem 6.2** *There is a polynomial-time algorithm that, given an undirected graph with edge costs satisfying the triangle inequality, and an integer $k \geq 2$ (the vertex-connectivity requirement), outputs a $k$-connected spanning subgraph of $G$ in which the degree of every node is exactly $k$, the total cost of all the edges in the subgraph is at most $\frac{k+4}{2}$ times that of a minimum-cost $k$-connected subgraph, and the bottleneck cost of the subgraph is at most $3 \cdot \lceil \frac{k}{2} \rceil$ times that of a minimum bottleneck-cost spanning tree.*

**Proof:** Let $c^*$ and $\beta^*$ denote the cost of an MST and the optimum bottleneck cost of a spanning tree of the input graph. By Proposition 6.1, we can obtain a TSP tour $T$ of cost $c(T)$ and bottleneck cost $\beta(T)$ such that $c(T) \leq 2c^*$ and $\beta(T) \leq 3\beta^*$. 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: For every node, add edges joining it to vertices to its left in the cycle that are within $\lceil \frac{k}{2} \rceil$ edges from it and all vertices to its right in the cycle that are within $\lfloor \frac{k}{2} \rfloor$ edges from it. It is not hard to see that this graph is $k$-vertex-connected (by showing $\lceil \frac{k}{2} \rceil$ disjoint paths between any pair of nodes going clockwise in the cycle and another $\lfloor \frac{k}{2} \rfloor$ disjoint paths going counter-clockwise). The degree of every node in this graph is exactly $k$. Since each shortcut employed replaces a path of at most $\lceil \frac{k}{2} \rceil$ edges, the bottleneck cost goes up by this factor. This proves that the bottleneck cost of this subgraph is within $3 \cdot \lceil \frac{k}{2} \rceil$ of optimal.

The total cost of the graph obtained this way can be computed by bounding how many newly added edges contain a given edge in the TSP tour within their span of $\frac{k}{2}$ or less. We can compute this for an edge $uv$ by counting all the added edges that originate at $u$ or to the left of it and end at $v$ or to its right. The number of such edges originating at $u$ is $\lceil \frac{k}{2} \rceil$, and the number originating at the node before $u$ crossing over $uv$ is $\lceil \frac{k}{2} \rceil - 1$ and so on, giving a total of at most $\frac{k(k+2)}{8} + 1$. Thus the total cost of this graph is at most $\frac{k(k+2)}{8} + 1$ times that of the TSP tour $T$ that we started with. This in turn is at most $(\frac{k(k+2)}{4} + 1)c^*$. However, we can apply an approximate min-max relation between a MST and a packing of cuts in the graph that is derived in [1, 14] in proving a better performance guarantee of $\frac{k+2}{2} + 1$ 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 $(\frac{k+2}{2} + 1)OPT_k$ as claimed in Theorem 6.2.

It remains to prove that $OPT_k \geq \frac{kc^*}{2}$. We do this in the remainder of this section. Before that we need some definitions. Given a graph $G$, recall that an edge 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 $W$. 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, 14].

**Theorem 6.3** *Given an undirected graph with edge-weights, a minimum-weight spanning tree has weight at most twice the value of a maximum packing of cuts.* □

The algorithms in [1, 14] find a greedy packing of cuts and simultaneously build 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 $OPT_k \geq \frac{kc^*}{2}$. □

# 7 Hardness Results

In this section, we prove hardness results that motivate the need for bicriteria approximations rather than approximating only one objective while strictly obeying the budget on the other. We first prove the results for spanning trees and then strengthen the results for Steiner trees.

## 7.1 Hardness Results for Spanning Tree Problems

**Theorem 7.1** *1. Unless* P = NP*, for any* $\rho > 1$*, there is no polynomial time* $(1, \rho)$ *approximation algorithm for the problem (*U-DEGREE, E-TOTAL COST, SPANNING TREE*).*

*2. Unless* P = NP*, for any* $\rho > 1$*, there is no polynomial time* $(1, \rho)$ *approximation algorithm for the problem (*U-DEGREE, E-BOTTLENECK-COST, SPANNING TREE*).*

*3. Unless* P = NP*, for any* $1 \leq \rho < 2$*, there is no polynomial time* $(1, \rho)$ *approximation algorithm for the problem (*U-DEGREE, E-BOTTLENECK-COST, SPANNING TREE*), even when edge weights satisfy triangle inequality.*

**Proof:** The NP-hardness of (U-DEGREE, E-TOTAL COST, SPANNING TREE) and (U-DEGREE, E-BOTTLENECK-COST, SPANNING TREE), follows via a straightforward reduction from the HAMILTONIAN PATH problem in which we add a the right number of distinct leaves to each node of the original graph.

To prove the third part, we use the cost assignment as in the first part of the proof 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 7.1. □

## 7.2 Hardness Results for Steiner Tree Problems

Since a spanning tree is a special case of a Steiner tree, it follows from Part 1 of Theorem 7.1 that unless P = NP, there is no polynomial time $(1, \rho)$ or $(\rho, 1)$ approximation algorithm for the (U-DEGREE, E-TOTAL-COST, STEINER TREE) problem for any $\rho > 1$. Furthermore, since the problem of computing a Steiner tree of minimum total edge weight (even without any degree constraints on nodes) is NP-hard, it follows that unless P = NP, there is no polynomial time $(\rho, 1)$ approximation algorithm for the (U-DEGREE, E-TOTAL-COST, STEINER TREE) problem for any $\rho > 1$.

These hardness results require either the budget to be satisfied exactly or the cost of the network to be optimal. We now present a result which points out the difficulty of solving the Steiner version of the non-uniform degree bounded problem within constant factors. This result is obtained by a reduction from the SET COVER problem. Recently, Arora and Sudan [3], and independently Raz and Safra [27] have shown the following non-approximability result for MIN SET COVER.

**Theorem 7.2** *Unless* P = NP, *the* MIN SET COVER *problem, with a universe of size* $k$*, cannot be approximated to better than a* $\ln k$ *factor.* □

**Theorem 7.3** *Unless* P = NP, *for any* $\varepsilon > 0$*, there is no polynomial time* $(2 - \varepsilon)$*-approximation algorithm for the non-uniform degree-bounded Steiner tree problem.*

**Proof:** Suppose there is a polynomial time $(2 - \varepsilon)$-approximation algorithm A for the problem. We will show that A can be used to obtain a polynomial time 2-approximation for the MIN SET COVER. In view of Theorem 7.2, the required result would follow.

Given an instance of MIN SET COVER, we construct the natural bipartite graph with one partition for set nodes (denoted by $Q_1$, $Q_2$, ..., $Q_m$) and the other for element nodes (denoted by $q_1$, $q_2$, ..., $q_n$), and edges representing element inclusion in the sets. To this bipartite graph, we add an "enforcer" node (denoted by $x$) which is adjacent to each of the set nodes. Let $G$ denote the resulting bipartite graph. The set $R$ of terminals for the Steiner tree instance is given by $R = \{x, q_1, q_2, \ldots, q_n\}$.

In this way, we create a sequence of $m$ instances of the problem (N-DEGREE, E-TOTAL-COST, STEINER TREE). In all these instances, the degree bound for each element node is chosen as 1 and the degree bound for each set node is chosen as $n + 1$. For the $j^{\text{th}}$ instance of the (N-DEGREE, E-TOTAL-COST, STEINER TREE) problem, the degree bound on the enforcer node is chosen as $j$ $(1 \leq j \leq m)$.

Suppose there is an optimal solution $Q' = \{Q_{i_1}, Q_{i_2}, \ldots, Q_{i_k}\}$ consisting of $k$ sets to the MIN SET COVER instance. Then the Steiner tree $T$ in $G$ consisting of $x$, the edges $(x, Q_{i_j})$, $1 \leq j \leq k$, and one edge from each element node to some set node in $Q'$ satisfies all the degree constraints. The cost of $T$ is equal to $k$.

Suppose we run the approximation algorithm A successively on instances 1, 2, ..., $m$ of the (N-DEGREE, E-TOTAL-COST, STEINER TREE) problem. Note that A may fail to produce a Steiner tree on some of these instances since there may be no Steiner tree satisfying the degree constraints, even after allowing for degree violations by a factor of $2 - \varepsilon$. We stop as soon as A produces a solution. We now argue that from this solution, we can obtain a 2-approximate solution to the MIN SET COVER instance. To see this, note that when we run A on instance $k$, A must produce a Steiner tree $T'$, since as argued above, there is a feasible solution to instance $k$. Since the degree requirement for each element node is 1 and the violation factor is less than 2, the degree of each element node in $T'$ is 1. Similarly, the degree of the enforcer node $x$ in $T'$ is less than $2k$. The set nodes adjacent to $x$ must cover all the element nodes since the degree of each element node is 1. We thus have a solution of size at most $2k$ for MIN SET COVER and this completes the proof. □

**Corollary 7.4** *Unless* P = NP, *for any* $\varepsilon > 0$ *and* $\rho > 1$ *, there is no polynomial time* $(2 - \varepsilon, \rho)$*-approximation algorithm for the (*N-DEGREE, E-TOTAL-COST, STEINER TREE*) problem.* □

# 8 Concluding Remarks

We have introduced bicriteria approximation algorithms for degree-constrained minimum-cost one-connected network problems, that allow general degree specifications and node costs. Our results for

bicriteria problems can be used to improve previous results on approximating certain minimum degree network problems. In particular, Theorem 5.9 implies a polynomial-time approximation algorithm for a class of minimum-degree forest problems considered by Ravi, Raghavachari and Klein [26]. They address the problem of finding one-connected networks that are cut-covers of proper functions such that the maximum degree of any node in the network is minimum. This is a single criterion problem without the node weight objective. They provide a quasi-polynomial ($n^{O(\log_{1+\epsilon} n)}$-time) approximation algorithm for these problems on an $n$-node graph that provides a solution of degree at most $(1+\epsilon)$ times the minimum with an additive error of $O(\log_{1+\epsilon} n)$, for any $\epsilon > 0$. A prototypical example of the one-connected network problem considered in [26] 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 [26] can be adapted to provide polynomial-time approximation algorithms with performance ratio $\Omega(n^{\delta})$ for any constant $\delta > 0$ (by setting $\epsilon = n^{\frac{1}{\delta}}$). By a direct application of Theorem 5.9, an improved (logarithmic) approximation ratio can be achieved in polynomial time for this problem.

### Subsequent Work

In subsequent work, we have used a similar framework to devise approximation algorithms for other bicriteria problems (see [21, 24]). An obvious open problem resulting from this work is to improve the performance ratios in all our results; although different techniques than those given seem to be required. In this context, it would be interesting to investigate whether the primal-dual method [1, 14] can be applied to provide such better guarantees and also provide a general framework for bicriteria network-design problems. Another interesting question is to investigate the extension of our work to higher-connected degree-constrained networks without the triangle inequality.

In other follow-up to our work, the special case of the (U-DEGREE, E-TOTAL COST, SPANNING TREE) problem in the Euclidean plane was addressed in [17], and improvements to the short-cutting scheme of Proposition 6.1 using network flow techniques are presented in [10].

**Acknowledgments:** We thank the referee for several valuable suggestions. We gratefully acknowledge helpful conversations with M. X. Goemans, P. N. Klein, G. Konjevod, S. Krumke, B. Raghavachari, V. S. Ramakrishnan, S. Subramanian and R. Sundaram.

## References

[1] A. Agrawal, P. Klein and R. Ravi, "When Trees Collide: An Approximation Algorithm for the Generalized Steiner Problem on Networks," *SIAM J. Computing*, Vol. 24, pp. 440–456, 1995.

[2] R. Ahuja, T. Magnanti and J. Orlin, *Network Flows: Theory and Algorithms*, Prentice Hall, Englewood Cliffs, N.J. 1993.

[3] S. Arora and M. Sudan, "Improved Low-Degree Testing and its Applications," *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC'97)*, pp. 485–496, 1997.

[4] B. Boldon, N. Deo and N. Kumar, "Minimum Weight Degree Constrained Spanning Tree Problem: Heuristics and Implementation on a SIMD Parallel Machine," *Parallel Computing*, Vol. 22, No. 3, pp. 369–382, March 1996.

[5] P. M. Camerini, G. Galbiati and F. Maffioli, "The Complexity of Weighted Multi-Constrained Spanning Tree Problems," *LOVSZEM: Colloquium on the Theory of Algorithms*, North-Holland, 1985.

[6] W. Cook, W. Cunningham, W. Pulleybank and A. Schrijver, *Combinatorial Optimization*, Wiley-Interscience Series on Discrete Mathematics and Optimization, New York, NY, 1998.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill Book Co., Cambridge, MA, 1990.

[8] N. Deo and S. L. Hakimi, "The Shortest Generalized Hamiltonian Tree," *Proc. 6th Annual Allerton Conference*, pp. 879–888, 1968.

[9] C. W. Duin and A. Volgenant, "Some Generalizations of the Steiner problem in Graphs," *Networks*, Vol. 17, pp. 353–364, 1987.

[10] S. Fekete, S. Khuller, M. Klemmstein, B. Raghavachari and N. Young, "A Network Flow Technique for Finding Low-Weight Bounded-Degree Spanning Trees," *J. Algorithms*, Vol. 24, No. 2, pp. 310–324, August 1997.

[11] T. Fischer, "Optimizing the Degree of Minimum Weight Spanning Trees," Technical Report TR 93-1338, Department of Computer Science, Cornell University, Ithaca, New York, April 1993.

[12] M. Fürer and B. Raghavachari, "Approximating the minimum-degree Steiner tree to within one of optimal," *J. Algorithms*, Vol. 17, No. 3, pp. 409–423, November 1994.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.

[14] M. Goemans and D. Williamson, "A General Approximation Technique for Constrained Forest Problems," *SIAM J. Computing*, Vol. 24, pp. 296–317, 1995.

[15] D. Hochbaum (Editor), *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Company, Boston, MA, 1997.

[16] A. Iwainsky, E. Canuto, O. Taraszow and A. Villa, "Network Decomposition for the Optimization of Connection Structures," *Networks*, Vol. 16, pp. 205–235, 1986.

[17] S. Khuller, B. Raghavachari and N. Young, "Low-Degree Spanning Trees of Small Weight," *SIAM J. Computing*, Vol. 25 No. 2, pp. 355–368, April 1996.

[18] P. Klein and R. Ravi, "A Nearly Best-Possible Approximation for Node-Weighted Steiner Trees," *J. Algorithms*, Vol. 19, No. 1, pp. 104–115, July 1995.

[19] 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 Symp. Foundations of Computer Science (FOCS)*, pp. 422–431, 1988. (Complete version to appear in *J. ACM*.)

[20] C. Lund and M. Yannakakis, "On the Hardness of Approximating Minimization Problems," *J. ACM*, Vol. 41, No. 5, pp. 960–981, September 1994.

[21] M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz and H. B. Hunt III, "Bicriteria Network Design Problems," *J. Algorithms*, Vol. 28, No. 1, pp. 142–171, July 1998.

[22] C. Monma and S. Suri, "Transitions in Geometric Minimum Spanning Trees," *Discrete & Computational Geometry*, Vol. 8, No. 3, pp. 265–293, 1992.

[23] C. Papadimitriou and U. Vazirani, "On Two Geometric Problems Related to the Traveling Salesman Problem," *J. Algorithms*, Vol. 4, pp. 231–246, 1984.

[24] R. Ravi, "Rapid Rumor Ramification," *Proc. 35th Annual IEEE Symp. on the Foundations of Computer Science (FOCS'94)*, pp. 202–213, November 1994.

[25] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III, "Many Birds with One Stone: Multi-Objective Approximation Algorithms," *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC'93)*, pp. 438–447, 1993.

[26] R. Ravi, B. Raghavachari and P. N. Klein, "Approximation Through Local Optimality: Designing Networks with Small Degree," *Proc. 12th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FST & TCS)*, Springer Verlag, LNCS 652, pp. 279–290, December 1992.

[27] R. Raz and S. Safra, "A Sub-Constant Error-Probability Low-Degree Test and a Sub-Constant Error-Probability PCP characterization of NP," *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC'97)*, pp. 475–484, 1997.

[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*, Vol. 6, No. 3, pp. 563–581, 1977.