Approximation algorithms for finding low-degree subgraphs*

Philip N. Klein† Radha Krishnan‡ Balaji Raghavachari§ R. Ravi¶

Abstract

We give quasipolynomial-time approximation algorithms for designing networks with minimum degree. Using our methods, one can design networks whose connectivity is specified by "proper" functions, a class of 0/1 functions indicating the number of edges crossing each cut. We also provide quasipolynomial-time approximation algorithms for finding a two-edge-connected spanning subgraph of approximately minimum degree of a given two-edge-connected graph, and a spanning tree (branching) of approximately minimum degree of a *directed* graph. The degree of the output network in all cases is guaranteed to be at most $(1+\epsilon)$ times the optimal degree, plus an additive $O(\log_{1+\epsilon} n)$ for any $\epsilon > 0$.

Our analysis indicates that the degree of an optimal subgraph for each of the problems above is well-estimated by certain polynomially solvable linear programs. This suggests that the linear programs we describe could be useful in obtaining optimal solutions via branch-and-bound.

1 Introduction

Minimizing the maximum degree is desirable in communications networks. The advantage of a network with low maximum degree is that the failure of a single node does not result in adverse conditions for a large part of the network. Keeping the maximum degree small is also essential in designing switching networks with identical switches installed at all nodes. In this case, switches for the network need to be designed to handle as many connections as the maximum degree of any node. Minimum-degree networks are also useful in building networks for broadcast where we

*Parts of this paper have appeared in the *Proceedings of the 12th and the 21st Annual Conferences on the Foundations of Software Technology and Theoretical Computer Science* (1992/LNCS 652/pp. 279-290 and 2001/LNCS 2245, pp. 232-243).

†Department of Computer Science, Box 1910, Brown University, Providence, RI 02912-1910. E-mail: pnk@cs.brown.edu. Research supported by NSF grant CCR-9012357 and NSF PYI award CCR-9157620, together with PYI matching funds from Thinking Machines Corporation and Xerox Corporation. Additional support provided by DARPA contract N00014-91-J-4052 ARPA Order No. 8225.

‡Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080-0688. E-mail: Radha.Krishnan@utdallas.edu.

§Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080-0688. E-mail: rbk@utdallas.edu. Research supported by the National Science Foundation under grant CCR-9820902.

¶Graduate School of Industrial Administration Carnegie Mellon University Schenley Park, Pittsburgh, PA 15213-3890. E-mail: ravi@cmu.edu. Research done while this author was at Brown University and 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.

wish to minimize the amount of work done at each site and also in designing power grids where the cost of a node increases with the degree of splitting the power [7]. Thus, the very nature of the network itself gives rise to certain connectivity requirements in the network. In this paper, we study algorithms for designing such low-degree communications networks.

The following versions of the basic minimum-degree subgraph problems are considered.

- Network design problems where the connectivity requirements can be modeled by a $\{0, 1\}$-valued function f on all the cuts in the graph. This framework is general enough to capture the Steiner and generalized Steiner tree problems.

- Two-edge-connected spanning subgraph of a given graph.

- Spanning tree in directed graphs.

In each case, the goal is to find, for the given graph, a minimum-degree subgraph that satisfies the given connectivity constraints. We provide approximation algorithms that output networks in each case, whose degrees are guaranteed to be at most $(1 + \epsilon)$ times optimal, plus an additive term of $O(\log_{1+\epsilon} n)$ for any $\epsilon > 0$. The algorithms run in quasipolynomial time $O(n^{\log_{1+\epsilon} n})$. As a direct consequence, we obtain polynomial-time approximation algorithms for all problems above with performance ratio $O(n^{1/\delta})$ for any $\delta > 0$ (by setting $\epsilon = n^{\delta}$). In addition to same performance guarantees and running times, all the algorithms we describe here also have in common the technique of local optimization, wherein we perform local improvements to the current solution to iteratively decrease the degrees of high-degree nodes.

In Section 2, we review previously known results for special cases of the problems we consider in this paper. In Section 3, we describe our results on general minimum-degree one-connected networks. In Section 4, we address the minimum-degree two-edge-connected spanning network-design problem. The directed minimum-degree spanning tree problem is considered in Section 5.

2 Previous Work

The basic minimum-degree spanning tree (MDST) problem: Given an undirected graph G, the problem is to find a spanning tree of G whose maximal degree is minimal. This problem is a generalization of the Hamiltonian Path problem and hence is NP-hard [10]. Fürer and Raghavachari [6] gave a polynomial time approximation algorithm for this problem with $O(\log n)$ performance guarantee. In subsequent work [7], they improved their previous results and provided another polynomial time algorithm to approximate the minimum-degree spanning tree to within one of the optimal. Clearly, no better approximation algorithms are possible for this problem.

Lawler [18] showed that matroid methods sufficed to solve the following variant of the minimum-degree spanning tree problem in polynomial time: given a graph G and an independent set I of nodes of G, find a spanning tree that minimizes the maximum degree of any node in I. Gavish [11] formulated the minimum-degree spanning tree problem as a mixed integer program and provided an exact solution using the method of Lagrangian multipliers.

The minimum-degree Steiner tree problem: In this extension to the MDST problem, given an input graph and a distinguished subset of the vertices D, we seek to find a Steiner tree (spanning at least the set D) whose maximum degree is minimum. The first polynomial-time approximation algorithm was provided by Agrawal, Klein and Ravi [2]. The performance guarantee is a factor of $O(\log k)$, where k is the size of D. This was improved upon by Fürer and Raghavachari [7], who provide a polynomial-time approximation algorithm for the problem, Their performance guarantee is essentially the same as that shown by us. In fact, our work is a generalization of their algorithm, and reduces to their algorithm for this special case. Fürer and Raghavachari [8] later demonstrated a polynomial-time algorithm that finds a tree whose degree is within one from optimal.

The directed minimum-degree spanning tree problem: Given a directed graph G and a root vertex $r \in V$, the problem is to find a spanning tree of G rooted at r whose maximal degree is minimized. For this problem, Fürer and Raghavachari [6] gave an algorithm with an approximation ratio of $O(\log n)$ for this problem. Our paper improves the multiplicative factor of $O(\log n)$ to an additive term of $O(\log n)$. Unfortunately, the "plus one" algorithm [8] does not generalize to directed graphs.

3 The Minimum-degree Constrained Forest Problem

We begin this section by describing a framework for formulating general one-connected network problems. We formulate the minimum degree problem in this framework.

3.1 Formulation

A framework for specifying connectivity requirements for networks was proposed by Goemans and Williamson [12]. This framework captures a wide variety of requirement specifications, including Steiner and generalized Steiner connectivity, general point-to-point connectivity, and T-joins. Building on the work of Agrawal, Klein, and Ravi on the generalized Steiner tree problem [1], Goemans and Williamson showed a way to find a network satisfying given connectivity requirements that has nearly minimum total edge cost. In this paper, we show how to find a network satisfying the requirements that has nearly minimum degree. We use the framework from [12] for specifying connectivity requirements. Our algorithm and analysis are based on the work of Fürer and Raghavachari [7].

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

A broad class of requirements can be specified using an approach of Goemans and Williamson [12]. They specify which cuts are active using a 0–1 function f on the node-subsets of a graph. For a node-subset S, let $\Gamma(S)$ denote the set of edges each having exactly one endpoint in S. To specify that the cut $\Gamma(S)$ is active, we set $f(S)$ to be 1. One can now formulate an integer program for a minimum-degree network crossing all active cuts:

$$
\begin{array}{l}
\text{Min } d \\
\quad \text{subject to constraints:} \\
\qquad \begin{array}{rcll}
x(\Gamma(S)) & \geq & f(S) & \forall\, S \subset V \\
x(\Gamma(\{v\})) & \leq & d & \forall\, v \in V \\
x_e & \in & \{0,1\} & \forall\, e \in E
\end{array} \qquad\qquad \text{(IP)}
\end{array}
$$

Here, for any subset F of edges, $x(F)$ is defined to be $\sum_{e \in F} x_e$. We call any feasible solution to the above IP, an f-join. It is easy to verify that minimal f-joins are forests.

Goemans and Williamson [12] focus on a special class of functions f that can be used to formulate many important connectivity requirements. They called these functions *proper*. A function $f : 2^V \rightarrow \{0,1\}$ is proper if the following properties hold.

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

We are interested in solutions to (IP) for the class of proper functions f. Examples of problems that fit within this framework other than the minimum-degree Steiner trees are minimum-degree generalized Steiner forests, minimum-degree T-joins, and minimum-degree point-to-point connection problems.

3.2 Preliminaries: Toughness, Weakness and lower bounds

Toughness, first defined by Chvátal in [4], is a measure of how hard it is to disconnect a graph into many components by removing a small number of nodes. The toughness of a graph is the minimum (over all non-trivial node-subsets) of the ratio of nodes removed (X) to the number of connected components in the remaining graph. That is, it is the minimum of all ratios of $|X|$ to the number of components in $G - X$, where X ranges over all non-trivial node-subsets of G. Computing the toughness of a graph was shown to be NP-complete by Bauer, Hakimi and Schmeichel [3]. Win [22] has shown the following interesting relationship between the toughness of a graph and the minimum-degree spanning tree problem. He showed that if the toughness of a graph is at least $\frac{1}{k-2}$, then it has a spanning tree whose degree is at most k (for $k > 2$).

The definition of toughness we have given differs slightly from Chvátal's original definition in [4]. According to our definition, the minimum toughness ratio is never more than 1 (for non-singleton graphs), since even a singleton X yields a ratio of at most 1. Chvátal defines toughness to be the same minimum ratio, but the minimum is taken only over those node-subsets X for which $G - X$ has at least two components. According to this definition, the minimum ratio can be arbitrarily large.

We generalize the above notion to allow active components defined by proper functions f. For any $X \subseteq V$, a connected component Y of $G - X$ is *active* under a proper function f whenever $f(Y) = 1$. The f-toughness of a graph for any given function f is the minimum ratio of the nodes removed to the number of active components formed. In other words, it is the minimum ratio of $|X|$ to the number of active components in $(G - X)$, where X ranges over all non-trivial node-subsets of G. For any subset X of nodes, this ratio is termed the f-toughness of the set X. Thus the f-toughness of the graph is the minimum f-toughness of any of its non-trivial node subsets. Note that ordinary toughness is a special case of f-toughness as we have defined it, since it corresponds to a function f where every non-trivial subset of the nodes of the graph is active. We are interested in computing the f-toughness of the graph for proper functions f. Call any single node that forms an active set a *terminal*. Note that as long as there is at least one terminal, the f-toughness ratio is at most 1, since this is the ratio achieved on removing a single terminal.

We shall refer to the reciprocal of the f-toughness of a graph as its *f-weakness*. The notion of weakness of a graph arises as a dual to the problem of constructing minimum-degree networks. The f-weakness of the graph is an estimate of the vulnerability of any f-join. A node subset with high value of f-weakness represents a weak spot in such a network: damage to this subset results in nearly the worst disconnection among the nodes in the network. To motivate the relationship between the minimum degree of any f-join and the f-weakness, we prove the following lemma.

Lemma 3.1 *For any function f, the optimum value of (IP) is at least the f-weakness of the graph.*

Proof: Consider the node set X achieving the optimal f-weakness (See Figure 1). Let the number of active components in $G - X$ under f be k. In any f-join, each of the k active components in $G - X$ must have an edge to X. Thus the average degree (and hence the maximum degree) of a node of X in an optimal solution to (IP) is at least $k/|X|$, the f-weakness of the graph. □

3.3 The algorithm and its analysis

We now describe the algorithm for providing an approximate solution to (IP) and analyze its performance. As input we are given an undirected graph $G = (V, E)$, an arbitrary number $\epsilon > 0$ to determine the performance accuracy, and a proper function f defined on the cuts of G. We can assume without loss of generality that the graph G is connected, for otherwise we can work on each connected component of G independently. Our algorithm uses iterative local-improvements and outputs a forest F that is feasible for the covering constraints of (IP). Note that we can always assume without loss of generality that any feasible solution to (IP) is the incidence vector of a forest [12]. Feasible solutions satisfy the following lemma.

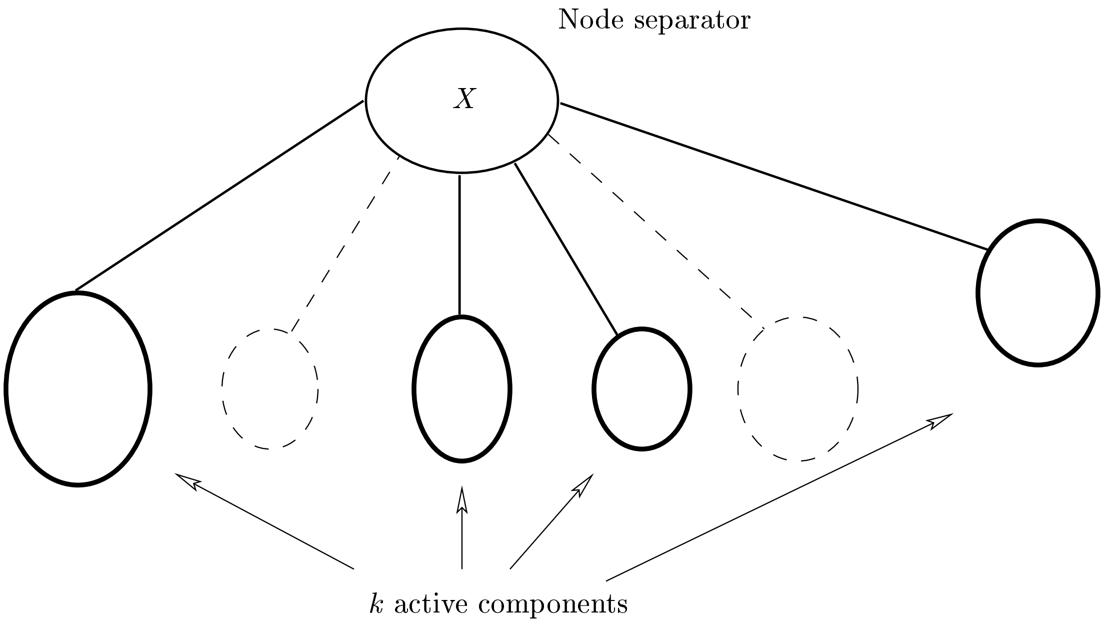


Figure 1: f-weakness is a lower bound on maximum degree of an f-join. In a feasible f-join, the degree of nodes in X is at least ratio of number of active components created on deleting X to cardinality of X.

Lemma 3.2 ([12]) *Let F denote a subset of the edge set of G. F is a feasible solution to (IP) if and only if $f(N) = 0$ for each connected component N of F.*

3.3.1 Overview

The algorithm starts with a feasible solution to (IP) and iteratively applies improvement steps aimed at reducing the degree of high-degree vertices. Intuitively, if we find a cycle in the graph that contains a node w of high degree, and if all edges that must be added to the current feasible solution to form this cycle are incident to low degree nodes, then we can improve the current solution by adding in the cycle and deleting an edge incident to w. This reduces the degree of w by one. For minimum-degree spanning trees [7, 8], improvements consisted of simply replacing one edge by another, and in the case of Steiner trees, they were defined to be replacing an edge by a path whose internal nodes are not in the tree. Therefore the degree of nodes in the tree increased by at most one. In our case, we are replacing an edge by a collection of paths that could go through nodes in other trees. Therefore, degree of a node may increase by two after an improvement. We examine this in greater detail below.

3.3.2 An improvement step

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

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

Lemma 3.3 *The forest F' formed at the end of an improvement step remains feasible.*

Proof: The lemma follows immediately if the improvement step is of the first type. Therefore suppose the improvement is of the second type. Let F be the feasible forest before the improvement step. Consider the cycle involved in the improvement. Since F is acyclic, the addition of the cycle and the deletion of an edge clearly leaves F' acyclic. But, we have merged many of the trees in F to form a single tree containing the edges of the cycle in F'. However, since F was feasible, Lemma 3.2 guarantees that each of the trees in the merged tree has f-value 0. Since the merged component is a disjoint union of these components, the disjointness property of f implies that the f value of this new component is also 0. All the remaining components in F' were also components of F having f-value 0. Therefore, Lemma 3.2 implies that F' is feasible. □

3.3.3 The algorithm

The algorithm starts with an initial feasible solution. Any spanning tree T of the given connected graph can be chosen as the initial feasible solution. Note that the null and symmetry conditions on f imply that $f(V) = 0$ and so, by Lemma 3.2, T is feasible. Let the maximum degree of any node in the current feasible solution be k. Ideally, we would like to run the algorithm until no further improvements are possible. Unfortunately, we are unable to bound the running time of such an algorithm even using a quasi-polynomial term. Therefore we restrict improvements only when they apply to high-degree vertices. The algorithm applies improvement steps with target d for $d \geq k - 2\lceil \log_{1+\epsilon} n \rceil$ until no such improvements are possible. The resulting forest is output as a *locally optimal* approximate solution.

Definition 3.4 *Define an edge e of a feasible forest F to be* critical *if $F - e$ is not feasible.*

As a result of performing the first type of improvement step, note that all the edges incident on nodes of degree at least $k - 2\lceil \log_{1+\epsilon} n \rceil$ are critical in the locally optimal solution.

3.3.4 Termination

Each improvement step is polynomial-time implementable. We adapt a potential-function argument from [7] to bound the number of improvement steps. We define the potential of a vertex v with degree d in the forest F to be $\phi(v) = n^d$ where n is the number of nodes in the graph. The potential of the forest F is defined as $\phi(F) = \sum_v \phi(v)$. Initially $\phi(F) \leq n^n$. Each improvement step with target d reduces the potential by at least $(n-1)n^{d-3}$, since the degree of a node of degree d is reduced by 1 and the degree of all the other nodes may increase to $d-1$. Since $d \geq k - 2\lceil \log_{1+\epsilon} n \rceil$ where k is the maximum degree of the current forest, this reduction in potential is at least a fraction $n^{-2\log_{1+\epsilon} n - 3}$ of the current potential of the forest $\phi(F)$. From this we can bound the maximum number of iterations before k decreases, and k can decrease at most $n-3$ times. It follows that the number of improvement steps is at most $n^{O(\log_{1+\epsilon} n)}$.

3.3.5 Performance guarantee

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

Proposition 3.5 *There is some $i \in [k - 2\lceil \log_{1+\epsilon} n \rceil, k]$ with $|S_{i-2}| \leq (1+\epsilon)|S_i|$.*

In the following discussion, let i be an index that satisfies the above proposition. We prove the following critical lemma that shows that when S_{i-2} (the set of nodes of degree at least $i-2$ in a locally optimal solution) is deleted from G, a large number of active components are created. This will then imply that the degree of nodes in S_{i-2} has to be large in any solution, since all these active components in $G - S_{i-2}$ can be connected together in a feasible solution only using edges that are incident to some vertex of S_{i-2}. The reason why our analysis is more complicated than the one used for minimum-degree Steiner trees is due to the fact that our feasible solution may have several trees, and therefore there may be edges in the graph that connect together some of these active components which are not useful in any improvements. But we show that the number of such components that may be merged with other components are small, and we can account for them in our analysis.

Lemma 3.6 *Let F be the locally optimal network and let i be the index in Proposition 3.5. The number of active components in $G - S_{i-2}$ is at least $i|S_i| - (2+\epsilon)|S_i|$.*

We prove the above lemma in the remainder of this section. Call a tree in F *relevant* if it contains a node in S_i. Define an equivalence relation $\sim$ on the edges of F by the rule: $e_1 \sim e_2$ if there is a path between an endpoint of e_1 and an endpoint of e_2 in F avoiding S_i. Intuitively, an equivalence class under this relation is a set of edges incident to nodes whose degree is smaller than i, forming a connected subtree in F. We define a bipartite graph that captures the way in which these subtrees are connected to nodes of S_i as follows. Define an auxiliary forest F_i whose nodes are S_i together with the equivalence classes of $\sim$. F_i has an edge between a node $v \in S_i$ and an equivalence class C under $\sim$ if there is an edge in C incident on v. Note that since F is acyclic, there is at most one edge in any equivalence class C incident to any node v in S_i. Thus if a node $v \in S_i$ has degree j in F, it also has degree j in F_i. Note also that F_i is acyclic and has exactly as many components as the number of relevant trees in F. We are especially interested in the leaves of F_i, i.e., subtrees of F that have a single edge of F that connects them to some node in S_i. We show a lower bound on the number of such subtrees.

Claim 3.7 *The number of nodes of degree one in F_i (corresponding to equivalence classes C) is at least $(i-2)|S_i| + 2r$ where r is the number of relevant trees in F.*

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

Each class C of degree one defines a set X_C of nodes as follows: a node v is in X_C if C contains all the edges of F that are incident on v. Note that each X_C is a component of $F - S_i$ and thus represents a subset of nodes of a relevant tree (See Figure 2). Consider an edge e in F incident on a node in S_i. Since F is feasible, by Lemma 3.2, the component of F containing e is inactive. Since e is critical on account of the first type of improvement step, the two distinct components formed in F on removing e and containing its endpoints must both be active. Consider an edge in a degree-one equivalence class C of $\sim$ that is incident on a node in S_i. One of the connected components formed on its removal from F is X_C, and so this set is active. This is summarized in the following proposition.

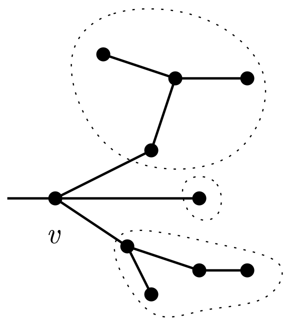


Figure 2: Three degree one equivalence classes under $\sim$, each with an edge to a node $v \in S_i$.

Proposition 3.8 *Each set X_C defined by a class C of degree one in F_i is an active set.*

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

Claim 3.9 *The total number of sets X_C's that merge with others on adding edges of $G - S_{i-2}$ is at most $2(r-1)$.*

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

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

The sets X_C's that remain might merge with inactive components of F on adding the nonforest edges of $G - S_{i-2}$. By the properties of a proper function, the resulting component in $G - S_{i-2}$ containing X_C is also active. By Claim 3.7, the number of degree-one equivalence classes is at least $(i-2)\,|S_i| + 2r$, and by Proposition 3.8 each of these defines an active set. We discount at most $\epsilon\,|S_i|$ of them that may break up on removing nodes in $S_{i-2} - S_i$. The number of such classes that may merge with one another is at most $2(r-1)$ by Claim 3.9. Thus the number of such active sets that remain is at least $i\,|S_i| - (2+\epsilon)\,|S_i|$. This completes the proof of Lemma 3.6.

3.4 An approximate min-max relation and applications

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

Theorem 3.10 *Let f be a proper function on the nodes of an n-node graph. The optimum value of (IP) is at most $(1+\epsilon)w^* + O(\log_{1+\epsilon} n)$ for any $\epsilon > 0$ where w^* is the f-weakness of the graph.*

Proof: Let i be as in Proposition 3.5. Our proof works by estimating the f-weakness of S_{i-2}. By Lemma 3.6, the number of active components in $G - S_{i-2}$ is at least $i|S_i| - (2+\epsilon)|S_i|$. By choice of i, note that $|S_{i-2}| \le (1+\epsilon)|S_i|$. Therefore the f-weakness of S_{i-2} and hence of G is at least $\frac{i-(2+\epsilon)}{1+\epsilon}$. Note that the degree of the solution F is at most $i + 2\log_{1+\epsilon} n$. Combining these proves Theorem 3.10. □

Note that the proof of Theorem 3.10 is constructive. We have provided an algorithm that constructs an f-join whose degree is close to the f-weakness ratio of a set we identify. Theorem 3.10 and Lemma 3.1 together prove the performance bounds given in Theorem 3.12. In addition, we also have the following result about approximating the f-weakness of the graph for any proper function f.

Theorem 3.11 *Let f be a proper function on the nodes of an n-node graph. Let w^* denote the f-weakness of a graph and let ϵ be an arbitrary small positive real number. There is an $n^{O(\log_{1+\epsilon} n)}$-time algorithm to determine a node subset X for which the f-weakness is at least $(1-\epsilon)w^* - O(\log_{1+\epsilon} n)$.*

An important application of the approximate min-max inequality presented in Theorem 3.10 arises from the fact that the relaxed version of (IP) is polynomially solvable. This follows from the observation that separation over (IP) is equivalent to solving an instance of finding the minimum-cut around any active set. The latter problem can be solved using the fact that proper functions are uncrossable and can be inferred from the results in [9, 21]. A 0-1 function f defined on subsets of vertices is called *uncrossable* if, whenever A and B are active sets with $f(A) = f(B) = 1$, then either $f(A \cup B) = f(A \cap B) = 1$ or $f(A - B) = f(B - A) = 1$ (see [12] for further details). Using the Ellipsoid method [13], which provides a polynomial-time reduction of the optimization problem to the separation problem, we have that the fractional relaxation of (IP) can be solved in polynomial-time. It follows from Theorem 3.10 that the value of this linear program is a good estimate of the minimum degree. While solving the linear program does not give us a solution network with this degree, just knowing the value can be useful, such as in a branch-and-bound search for an optimal solution [19].

Combining the above discussions yields the following theorem.

Theorem 3.12 *Let f be a proper function. Let d^* denote the optimum value of (IP). There is an $n^{O(\log_{1+\epsilon} n)}$-time algorithm for finding a feasible solution to (IP) whose objective value is at most $(1+\epsilon)d^* + O(\log_{1+\epsilon} n)$ for any $\epsilon > 0$.*

4 Minimum-Degree Two-Edge-Connected Subgraphs

In this section, we consider minimum-degree networks of a different type. Given a two-edge-connected undirected graph as input, we consider the problem of finding a two-edge-connected spanning subgraph of minimum degree. This problem can be easily seen to be NP-hard by a reduction from the Hamiltonian cycle problem. Using a local-improvement algorithm similar to that used in proving Theorem 3.12, we obtain an approximate solution for this problem as well.

Theorem 4.1 *Let Δ^* be the minimum degree of a two-edge-connected subgraph of a given graph G and let ϵ be a positive number. There is an $n^{O(\log_{1+\epsilon} n)}$-time algorithm for finding a two edge-connected subgraph of G having degree at most $(1+\epsilon)\Delta^* + O(\log_{1+\epsilon} n)$.*

As in Section 3, it is easy to cast this problem as an exponential-sized integer program. The first constraint is modified to $x(\Gamma(S)) \geq 2$ for all $S \subset V$. This captures the condition that our solution needs to have at least two edges across each cut, i.e., it is 2-edge-connected. The fractional relaxation of this program is solvable in polynomial time by using a minimum-cut procedure to solve the corresponding separation problem. Hence the above theorem can be applied to show that this linear program solution value is a very good estimate of the value of the exact solution to the minimum-degree problem. As mentioned earlier, such estimates can be used in pruning the search space for exact solutions to this problem.

4.1 Preliminaries

Let $G = (V, E)$ be an arbitrary k-edge-connected graph. Consider the problem of computing a k-edge-connected subgraph N of G which spans V such that the maximum degree of N is minimum. The case of $k = 1$ corresponds to the minimum-degree spanning tree problem and is NP-hard [10]. In this section, we consider the case when $k = 2$ (two-edge-connected graphs are also called bridge-connected). In this paper, "connectivity" always stands for edge connectivity (and not vertex connectivity). For a graph G, we use $\Delta^*(G)$ to denote the minimum degree of a two-connected spanning subgraph of G.

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

Definition 4.2 *A pair of vertices u and v is said to be* k-connected *in a graph N if there are k edge-disjoint paths between u and v in N. This relation is an equivalence relation. It partitions the vertices into equivalence classes of vertices. Within each class, each pair of vertices is k-connected. Such a class is called a* k-component.

Definition 4.3 *For a graph H, the* bridge-connected forest (bcf) *of H is obtained by contracting each 2-component of H to a supervertex and deleting self-loops. A* leaf 2-component *of H is a 2-component that has degree one in the bcf of H. An* isolated 2-component *of H is one of degree zero in the bcf of H.*

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

Lemma 4.4 *Let N be any 2-connected spanning subgraph of G, and let X be any subset of vertices. Let C be a 2-component of $G - X$. If C is a leaf 2-component, then N has at least one edge between C and X. If C is an isolated 2-component, then N has at least two edges between C and X.*

This lemma motivates the following definition.

Definition 4.5 *Let H be a graph with ℓ leaf 2-components and k isolated 2-components. Then the* deficiency *of H is defined as $\ell + 2k$ and is denoted by* $\mathrm{def}(H)$.

The stage is now set for an easy derivation of a lower bound on the minimum degree of a two-connected spanning subgraph. Consider removing a node subset X from an undirected graph G; the remaining graph has deficiency $\mathrm{def}(G-X)$. In any two-connected spanning subgraph of G, there must be at least $\mathrm{def}(G-X)$ edges going between $G-X$ and the nodes in X by Lemma 4.4. Thus, the maximum degree of a node in X is at least $\frac{\mathrm{def}(G-X)}{|X|}$. Applying this argument to the minimum-degree two-connected spanning subgraph, we arrive at the following corollary.

Corollary 4.6 *Let X be a subset of the nodes of a graph G. Then*

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

4.2 The algorithm and its time-complexity

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

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

Definition 4.7 *The* degree-d candidate graph *for an edge e in N is obtained from G as follows: contract each 2-component of $N-e$, and then delete e and the edges of $G-N$ incident to nodes with degree at least d in N (See Figure 3).*

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

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

Lemma 4.8 *Let Q be a minimal collection of edges whose addition two-connects a bcf N_e. Then for every node v of N_e, $\deg_Q(v) \leq 2$.*

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

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

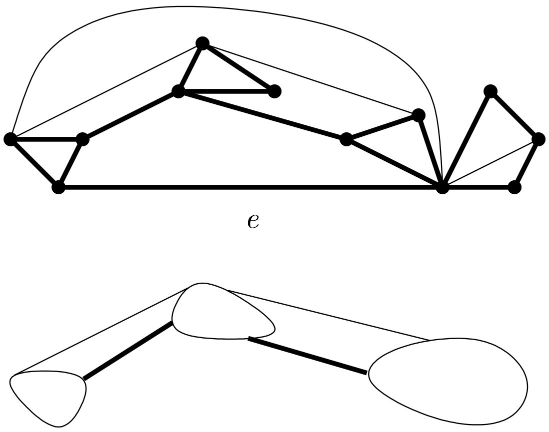


Figure 3: An example of a degree-5 candidate graph for an edge e. The figure on top illustrates the network N with thick edges. The figure in the bottom is the degree-5 candidate graph for the edge e, formed from G by contracting each of the three 2-components of $N - e$ to single nodes and deleting edges in $G - N$ incident on nodes of degree five or more.

Definition 4.9 *Let N be a two-connected spanning subgraph of G, and let Δ be the degree of N. Then N is called* locally optimal *if no vertex of degree at least $\Delta - 2\lceil \log_{1+\epsilon} n \rceil$ admits an improvement.*

Using a potential function argument as in Section 3, it is easy to show that $n^{O(\log_{1+\epsilon} n)}$ improvement steps suffice to yield a locally optimal subgraph. Since each improvement step can be executed in polynomial time, the running time is as claimed in Theorem 4.1. In the remainder of this section, we show that the degree of a locally optimal subgraph is as claimed in Theorem 4.1.

4.3 Performance guarantee

Theorem 4.10 *The degree of any locally optimal subgraph of a graph G with n nodes is at most $(1+\epsilon)\Delta^*(G) + 2\lceil \log_{1+\epsilon} n \rceil + 4(1+\epsilon)$.*

Let S_i denote the set of nodes whose degree in N is at least i. The proof of Theorem 4.10 consists in showing that there is an i in the range $[\Delta - 2\lceil \log_{1+\epsilon} n \rceil, \Delta]$ for which $G - S_i$ has many leaf and isolated 2-components. We show this in Lemma 4.13. Theorem 4.10 then follows from Corollary 4.6.

Lemma 4.11 *Let G be a 2-connected graph and let S be a subset of vertices of G. Then there is a 2-connected subgraph A of G that contains all vertices of S, such that $\sum_{v \in S} \deg_A(v) \leq 4|S|$.*

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

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

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

$$\sum_{v \in S} deg_{C_j}(v) \leq 2x_j. \tag{1}$$

It follows that

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

Next we bound the right-hand side of (2). Since at each iteration j, we contract x_j active vertices of H_j into a single active vertex in H_{j+1}, we have the following claim.

Claim 4.12 *(Number of active vertices in H_{j+1}) = (Number of active vertices in H_j) $-(x_j - 1)$*

Since H_k contains 1 active vertex and H_0 contains $|S|$ active vertices, it follows by Claim 4.12 that

$$\sum_{j=1}^{k} (x_j - 1) = |S| - 1 \tag{3}$$

In each iteration $x_j - 1 \geq 1$, so the number of iterations k is at most $|S| - 1$. Hence $\sum (x_j - 1) \geq \sum x_j - (|S| - 1)$. Combining this inequality with (3) yields $\sum x_j \leq 2(|S| - 1)$. Combining this with (2) yields $\sum_{v \in S} \deg_A(v) \leq 4(|S| - 1)$. This completes the proof of Lemma 4.11. $\square$

Lemma 4.13 *Let N be a locally optimal subgraph with degree Δ. Then there is an integer i in the range $[\Delta - 2\lceil \log_{1+\epsilon} n \rceil, \Delta]$ such that the deficiency of $G - S_{i-2}$ is at least $|S_i| \, (i - 4(1 + \epsilon))$ and $|S_{i-2}| \leq (1 + \epsilon)|S_i|$.*

Proof: As in Proposition 3.5, there is an i in the given range such that $|S_{i-2}| \leq (1 + \epsilon)|S_i|$. By Lemma 4.11, there exists a 2-connected subgraph A of N such that S_{i-2} is contained in A and $\sum_{v \in S_{i-2}} \deg_A(v) \leq 4|S_{i-2}|$. By choice of i, the value $4|S_{i-2}|$ is at most $4(1 + \epsilon)|S_i|$. To continue the proof, we show that a large portion of the degree of the nodes in S_{i-2} can be used to infer that $G - S_{i-2}$ has high deficiency.

Definition 4.14 *An edge e of a 2-connected graph H is* critical in H *if $H - e$ is not 2-connected.*

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

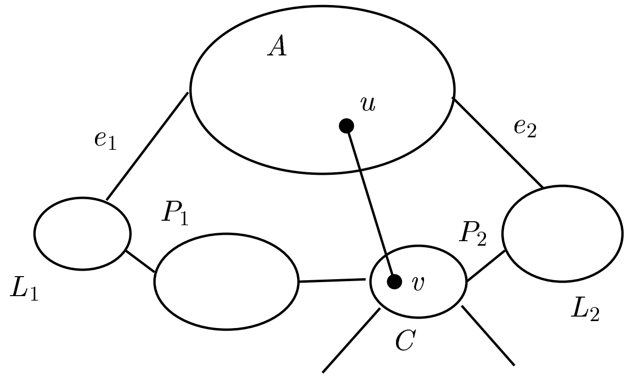


Figure 4: The edge (u, v) goes between $u \in S_{i-2}$ and a vertex v in a 2-component C in the bcf of $G - S_{i-2}$. Suppose C is internal, we can identify two edge-disjoint paths between u and C in the degree-$(i-3)$ candidate graph for e contradicting the local optimality of the network N.

Claim 4.15 *Let e be an edge of N not in A such that one endpoint of e belongs to S_i. Then the other endpoint of e belongs to a leaf 2-component or isolated 2-component of the* bcf *of $G - S_{i-2}$.*

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

Next suppose that C has degree two or more in $\text{bcf}(G - S_{i-2})$. Then C is internal to some path in $\text{bcf}(G - S_{i-2})$ between leaf 2-components L_1 and L_2. For $j = 1, 2$, let P_j be the subpath from C to L_j. By Lemma 4.4, there is an edge e_j of N between each L_j and S_{i-2} for $j = 1, 2$. We have constructed edge-disjoint paths P_1e_1 and P_2e_2 from C to A. The edges of these paths are in $N \cup (G - S_{i-2})$ (See Figure 4). Recall that A is two-connected and contains only edges in N, and that u is in A. It follows that C is two-connected to u using edges in $(N \cup (G - S_{i-2}) - \{e\})$. This contradicts the fact that the degree-$(i-3)$ candidate graph for e is not two-connected. □

Now we complete the proof of Lemma 4.13. The sum of the degrees of all the nodes of S_i in N is at least $i \cdot |S_i|$. By choice of the set A, we have $\sum_{v \in S_{i-2}} \deg_A(v) \leq 4(1+\epsilon)|S_i|$. Hence there are at least $|S_i|(i - 4(1+\epsilon))$ edges incident on S_i not in A. By Claim 4.15, each of these edges goes to a leaf 2-component or isolated 2-component of $\text{bcf}(G - S_{i-2})$. By the criticality of these edges in N, it also follows that there is at most one such edge to a leaf-component and at most two such edges to a isolated-component in $\text{bcf}(G - S_{i-2})$. Hence each such edge contributes one to the deficiency of $G - S_{i-2}$. This concludes the proof of Lemma 4.13. □

5 Directed minimum-degree spanning tree (DMDST) problem

5.1 Definitions and Notation

The input in this case is an arbitrary directed graph $G = (V, E)$, and a root vertex $r \in V$. Let n be the number of vertices in G. It is assumed that r is reachable from all vertices of G.

A *branching* rooted at r is a subgraph of G whose underlying undirected graph is a spanning tree such that it has a directed path from any vertex to r. In a branching, each vertex other than r has exactly one outgoing edge; r has no outgoing edges. Sometimes this is also known as an in-branching. One can also define the notion of an out-branching, in which r can reach every vertex of G through directed paths. In this paper, a branching always refers to an in-branching. Note that our algorithm can be easily modified to find out-branchings of small outdegree.

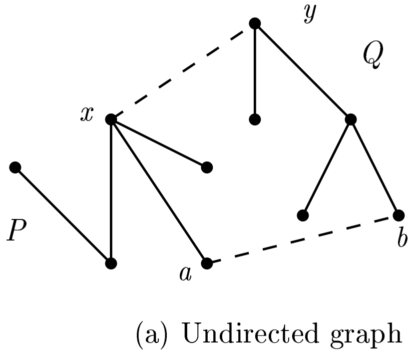
Let T be a branching. For each edge (v, w) in T, we call w the *parent* of v, denoted by $parent[v]$. Since every vertex except r has a unique outgoing edge in T, each vertex has a unique parent, and r has none. The reflexive and transitive closure of the parent function yields the ancestor relation. In other words, v is an *ancestor* of u if there is a directed path in the branching from u to v. We call u a *descendant* of v if v is its ancestor. We say that two vertices v and w are *related* if either v is an ancestor of w, or vice versa. Otherwise we say that the vertices are *unrelated*. For any two unrelated vertices v and w, the *least common ancestor* is the ancestor closest to v that is also an ancestor of w. We define C_v to be the set of all vertices in the subtree rooted at v, i.e., the set of all vertices including v, for which v is an ancestor.

The *degree* of a vertex in a given branching is the number of edges coming into that vertex. We may also refer to this as the *indegree*. For a branching, let S_Δ be the set of all vertices whose degree is Δ or more. The degree of a branching is the maximum degree of any of its vertices. Let T^* be an optimal DMDST whose maximal degree is Δ^*. Our goal is to find a branching whose (in)degree is as small as possible.

5.2 MDST problem: directed vs undirected graphs

In order to extend the earlier algorithms to directed graphs, we have to take note of the following differences.

First, in directed graphs, we face the following problem. Suppose an edge is removed that splits the current tree into two trees, namely P and Q (see Fig. 5). Suppose we try to combine the two trees by adding the edge (x, y), where $x \in P$ and $y \in Q$. In the case of undirected graphs, any such edge would do and we get back a spanning tree of G. But in the case of directed graphs, the vertex x must be the root of the tree P. Otherwise the procedure would not yield a branching of G. To illustrate, as shown in Fig. 5 (a), undirected trees P and Q can be merged into a single tree by adding either the edge (x, y) or the edge (a, b). But in the case of directed graphs, as shown in Fig. 5 (b), only the edge (x, y) yields a branching. If (a, b) is added, then a has two outgoing edges, and the resultant graph is not a branching. Therefore the improvement step needs to be modified.



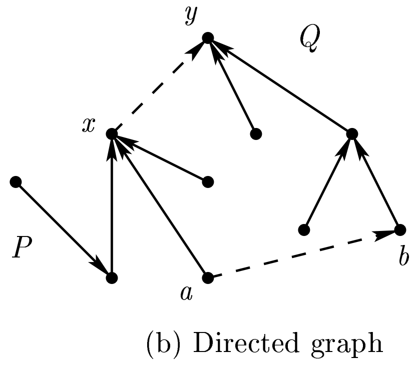


Figure 5: Merging in directed versus undirected graphs

Second, the notion of witness sets (see [8]) is slightly different. The earlier definition is not suitable for directed graphs. We prove a new lemma that can be used to establish a lower bound on Δ^* for directed graphs. Suppose we have a set of witness vertices W and a set of blocking vertices B satisfying the property that paths from different vertices of W do not intersect before being incident to a vertex in B (see Fig. 6). From this we show that there are $|W|$ paths that have distinct edges into B, thus establishing a lower bound on the degree of vertices in B in any branching. The following lemma formally establishes the lower bound.

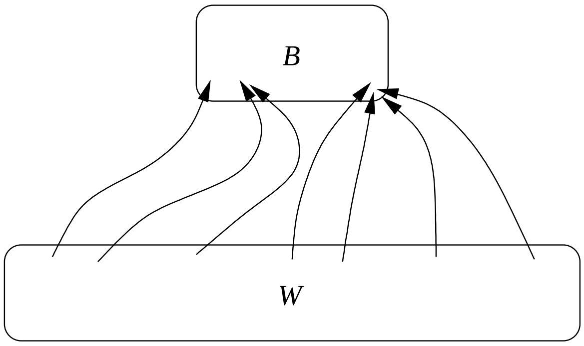


Figure 6: Paths from W are internally disjoint until reaching B. Therefore each of them forces an incoming edge to a node in B in any branching.

Lemma 5.1 *Let $G = (V, E)$ be a directed graph and $r \in V$. Suppose there are subsets of vertices $W \subset V$ and $B \subset V$ with $B \cap W = \emptyset$ that satisfy the following properties:*

1. *Any path from a vertex $v \in W$ to r must have an incoming edge into a vertex in B,*

2. *For any two vertices $v, w \in W$, any path from v to r can intersect a path from w to r only after it passes through a vertex in B. In other words, G has no branching wherein the path from v to the least common ancestor of v and w does not contain a vertex of B.*

Then the degree of an DMDST branching rooted at r of G satisfies, $\Delta^* \geq \lceil |W|/|B| \rceil$.

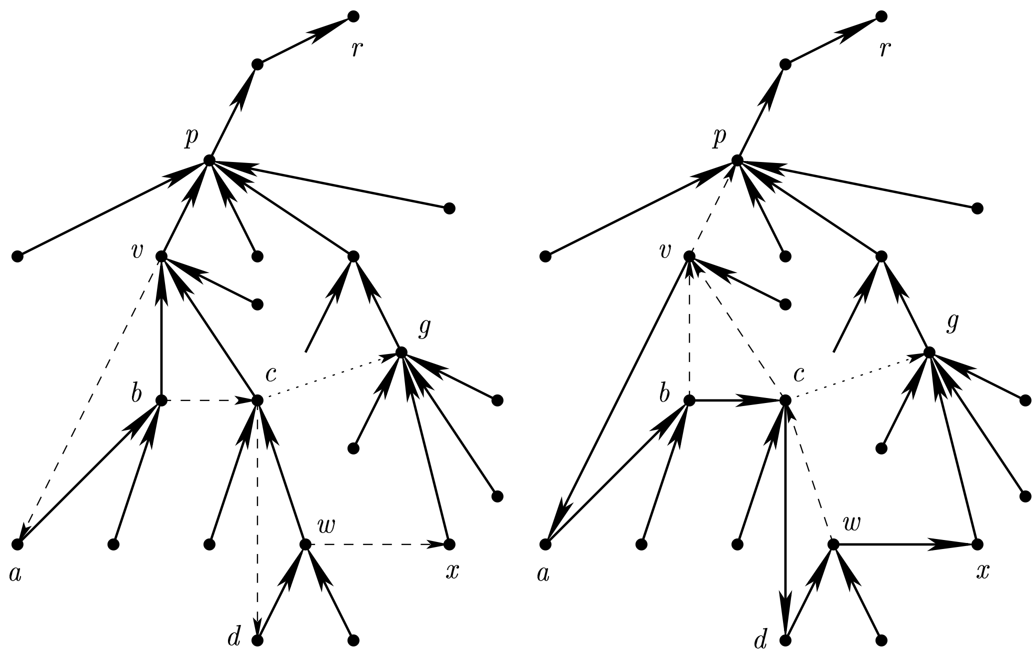
Proof: Let T^* be an optimal branching rooted at r for the DMDST problem. Since it is a branching, it contains a path from any vertex to the root. By Condition 1 of the lemma, a path from a vertex $v \in W$ to r contains at least one edge into a vertex in B. Let $f_v \neq v$ be the closest ancestor of v such that $f_v \in B$. Let P_v be this path from v to f_v. By Condition 2 of the lemma, the paths $\{P_v : v \in W\}$ are all internally disjoint. Therefore we have identified $|W|$ paths in T^*, and each of these paths has an incoming edge to some vertex in B. Therefore the average degree of a vertex in B is at least $|W|/|B|$, implying that there is at least one vertex in T^* whose degree is $\lceil |W|/|B| \rceil$ or more. □

5.3 The algorithm

Our algorithm starts with an arbitrary branching T of G and reduces the degree of high-degree nodes iteratively by applying improvement steps defined below. Consider a node v whose parent in the tree is p. We can decrease the indegree of p by 1 (which is an improvement step applied to p) if we can delete the edge (v, p) and find an alternate path for v to reach the root r. This new path from v to r initially goes through some nodes of C_v, vertices in the subtree rooted at v, reaching a node $w \in C_v$ (w may be v itself). A new edge (w, x) is added (replacing the edge from w to its parent) where x is unrelated to v. Since x is unrelated to v, it is unrelated to any vertex in C_v. Therefore the path from x to r in T is unaffected. Since v can reach x after the improvement, v can also reach r. We will perform such an improvement step only if, after the improvement, the degrees of none of those vertices whose degree increased, is more than the new degree of p.

Fig. 7 illustrates an example of an improvement step. In this example, the tree edges are shown in thick lines and other edges of G are shown in dashed lines. The indegree of p is 5, and we see if v can find an alternate path to r so that the edge (v, p) may be deleted from T, thus decreasing the degree of p to 4. The edge (c, g) is deleted because the indegree of g is already 4, and if we chose to add this edge, its indegree becomes 5. Decreasing the degree of p to 4 by increasing the degree of g to 5 (old degree of p) does not make progress. The edge (v, p) is also deleted and the algorithm tries to find a path from v to r. Such a path exists — $(v, a, b, c, d, w, x, \ldots)$ and the algorithm uses this path to modify the branching; the new branching is shown in Fig. 7 (b). The indegree of p has been reduced to 4.

We will now describe how to test if such an improvement exists. Let the degree of p be Δ. We first ensure that the degree of vertices whose degree is $\Delta - 1$ or greater does not increase. Delete all nontree edges of G that are incident into nodes of degree $\Delta - 1$ or greater, i.e., $S_{\Delta-1}$. In the remaining graph, delete the edge (v, p) and test if there is a path from v to r. If such a path exists, we can select a shortest such path and use it to make an improvement to p as follows. Let x be the vertex closest to v in the path such that $x \notin C_v$. For each edge (y, z) in the path from v to x, we replace the edge $(y, parent[y])$ by the edge (y, z). It can be verified that the above operation results in another branching since the number of edges is still $n - 1$ and all vertices can still reach



(a) Branching before improvement (b) Branching after improvement

Figure 7: Example: an improvement applied to vertex p

r. The algorithm can be summarized as follows.

Procedure IMPROVEMENT(T, v, p)

1. Delete (v, p) from G.
2. Let Δ be the degree of p. For each vertex $u \in V$ whose indegree in T is greater than $\Delta - 1$, delete from G edges going into u that are not in T.
3. Run Breadth-first search from v, and test if the root r is reachable from v.
4. If there is no path from v to r, return False after restoring all edges of G.
5. Otherwise, BFS finds a path P from v to r. Let w be the first vertex on the path with the property that $(w, x) \in P$ and $w \in C_v$ and $x \notin C_v$.
6. For each edge (a, b) in the subpath of P from v to x, replace the edge from $(a, parent[a])$ in T by (a, b).
7. Restore all edges of G and return True.

The DMDST algorithm tries to reduce the degree of high-degree vertices by finding suitable improvements. The target vertices are those whose degrees are within $O(\log n)$ from the maximal degree of the current branching. When no improvements are possible to these nodes, the algorithm terminates. The DMDST algorithm can be outlined as shown below.

Algorithm DMDST(G, r)

1. Find a branching T of G rooted at r. Let its (maximum) degree be k. Fix some constant $\epsilon > 0$.

2. For each edge $(v, p) \in T$, run IMPROVEMENT(T, v, p) if the degree of p in T is more than $k - \lceil \log_{1+\epsilon} n \rceil$. If the degree of T has changed, reset k to be its new degree.

3. Repeat the above step until IMPROVEMENT(T, v, p) returns false for every edge $(v, p) \in T$ for which it is called.

4. Return T.

5.4 Analysis of the algorithm

The analysis of the running time of the algorithm is similar to those described earlier. The following lemma shows the running time of the algorithm in terms of the number of improvement steps, I.

Lemma 5.2 *The running time of Algorithm DMDST is $O(n^3 I)$, where I is the number of improvement steps.*

We now show how to find a witness set W and its blocking set B for the branching T output by our algorithm. In fact we will identify one pair of sets W and B for each Δ in the range k to $k - \lceil \log_{1+\epsilon} n \rceil$.

Lemma 5.3 *Let T be a branching whose degree is Δ or more. Let S_Δ be the set of vertices whose degree is Δ or more. There are at least $(\Delta - 1)|S_\Delta| + 1$ unrelated vertices such that the parent of each of these vertices is in S_Δ.*

Proof: The proof is by induction on the cardinality of S_Δ. If $|S_\Delta| = 1$, then the single vertex in that set has at least Δ children, and the children of this vertex satisfy the lemma. If $|S_\Delta| > 1$, select a node $v \in S_\Delta$ such that it has no descendants in S_Δ (except itself). Remove v and all its descendants from T. Now the resulting branching has $|S_\Delta| - 1$ nodes of degree Δ or more, and by the induction hypothesis, has at least $(\Delta - 1)(|S_\Delta| - 1) + 1$ unrelated nodes that are children of S_Δ. Since all these nodes are unrelated to each other, at most one of these nodes is an ancestor of v. Therefore there are $(\Delta - 1)(|S_\Delta| - 1)$ nodes left that are not ancestors of v. Now we add the children of v to this set, its cardinality increases by at least Δ, and therefore the number of nodes is at least $(\Delta - 1)(|S_\Delta| - 1) + \Delta = (\Delta - 1)|S_\Delta| + 1$. $\square$

Lemma 5.4 *Let T be the branching output by the DMDST algorithm. Let its degree be k. Then for any Δ with $k - \lceil \log_{1+\epsilon} n \rceil < \Delta \leq k$,*

$$\Delta^* \geq \frac{(\Delta - 1)|S_\Delta| + 1}{|S_{\Delta-1}|}.$$

Proof: Let W be the set of vertices as in Lemma 5.3 that are children of nodes in S_Δ, but have no descendants in S_Δ. We know that $|W| \geq (\Delta - 1)|S_\Delta| + 1$. Let B be $S_{\Delta-1}$, the set of all vertices whose degree is at least $\Delta - 1$. For each vertex $v \in W$, the algorithm tries to find an improvement that decreases the degree of $p = parent[v]$. Since it failed (the condition under which the algorithm stops), any path from v to r that doesn't use (v, p) must go through a vertex x in $S_{\Delta-1}$. By construction, the internal vertices of the path from v to x are entirely contained in C_v, the set of descendants of v in T. Since all vertices of W are unrelated to each other, these subtrees are disjoint. Therefore, the sets W and B that we have defined satisfy the conditions given in the statement of Lemma 5.1. Therefore,

$$\Delta^* \geq \lceil |W|/|B| \rceil \geq \frac{(\Delta - 1)|S_\Delta| + 1}{|S_{\Delta-1}|}.$$

□

Theorem 5.5 *The degree of the branching returned by our algorithm is at most $(1 + \epsilon)\Delta^* + \lceil \log_{1+\epsilon} n \rceil$, where $\epsilon > 0$ is the constant chosen by the DMDST algorithm in Step 1.*

Proof: Lemma 5.4 establishes a set of lower bounds on Δ^* for $\lceil \log_{1+\epsilon} n \rceil$ different values of Δ. At least for one of these values of Δ, $|S_{\Delta-1}| \leq (1 + \epsilon)|S_\Delta|$. Using this value of Δ, we get $k \leq (1 + \epsilon)\Delta^* + \lceil \log_{1+\epsilon} n \rceil$. □

6 Conclusions

Although we could only show a quasipolynomial bound on the running times of our algorithms so far, we believe that our techniques can be used to obtain polynomial-time approximation algorithms with the same performance guarantees. This is the most important open problem resulting from this work. Recent techniques in multi-objective network-design approximation algorithms [20] give polynomial-time approximation algorithms with a logarithmic performance guarantee for the general one-connected network design problem formulated in (IP). This is better than the performance guarantee achievable in polynomial time using our results. However, the techniques used therein are quite different, and do not extend to the two edge-connected and directed graph problems that we addressed in this paper.

Könemann and Ravi [14, 15] have extended the techniques of Fürer and Raghavachari [8] to find approximately minimum-cost spanning trees of approximately minimum max-degree. They have also used techniques similar to the ones in this paper to improve the results on finding approximately minimum-cost Steiner trees of approximately minimum max-degree in [16].

It may be possible to extend the techniques used in this paper to more general proper functions with values in $\{0, 1, 2, \ldots, k\}$, with a loss of a factor k in the approximation ratio for the degree, by employing an iterative method as in [9]. In another direction, extending the result from 2-edge-connected to 2-vertex-connected graphs may also be possible by proving the corresponding vertex-connectivity analogues of the results in Section 4. An easier approach losing an additional factor of two would be to first find a two-edge-connected subgraph of minimum degree and augment this further in a second stage to be biconnected by using local search techniques. All of these remain promising directions for future work.

References

[1] A. Agrawal, P. Klein and R. Ravi, When trees collide: an approximation algorithm for the generalized Steiner tree problem on networks, SIAM J Computing 24 (1995), 445-456.

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

[3] D. Bauer, S.L. Hakimi, and E. Schmeichel, Recognizing tough graphs is NP-hard, Discret. Appl. Math. 28 (1990), 191-195.

[4] V. Chvátal, Tough graphs and Hamiltonian circuits, Discret. Math. 5 (1973), 215-228.

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

[6] M. Fürer and B. Raghavachari, An $\mathcal{NC}$ approximation algorithm for the minimum-degree spanning tree problem, Proc 28th Ann Allerton Conference on Communication, Control and Computing, 1990, pp. 274-281.

[7] M. Fürer and B. Raghavachari, Approximating the minimum-degree spanning tree to within one from the optimal degree, Proc 3rd Ann ACM-SIAM Symp Discrete Algorithms, 1992, pp. 317-324.

[8] M. Fürer and B. Raghavachari, Approximating the minimum-degree Steiner tree to within one of optimal, J Algorithms 17 (1994), 409-423.

[9] H. N. Gabow, M. X. Goemans, and D. P. Williamson, An efficient approximation algorithm for the survivable network design problem, Math. Programming 82 (1998), 13-40.

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

[11] B. Gavish, Topological design of centralized computer networks - formulations and algorithms, Networks 12 (1982), 355-377.

[12] M. X. Goemans, and D. P. Williamson, A general approximation technique for constrained forest problems, SIAM J Computing 24 (1995), 296-317.

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

[14] J. Könemann and R. Ravi, A Matter of Degree: Improved Approximation Algorithms for Degree-Bounded Minimum Spanning Trees, SIAM J Computing 31 (2002), 1783-1793.

[15] J. Könemann and R. Ravi, Primal-dual meets local search: approximating MST's with nonuniform degree bounds, Proceedings of the ACM Symposium on Theory of Computing (STOC), 389-395 (2003).

[16] J. Könemann and R. Ravi, Quasi-Polynomial Time Approximation Algorithm for Low-Degree Minimum-Cost Steiner Trees, Proc 23st Conf Foundations of Software Technology and Theoretical Computer Science (FST&TCS), Lecture Notes in Computer Science 2914, Springer 2003, pp. 289-301.

[17] R. Krishnan and B. Raghavachari, The directed minimum-degree spanning tree problem, Proc 21st Conf Foundations of Software Technology and Theoretical Computer Science (FST&TCS), Lecture Notes in Computer Science 2245, Springer 2001, pp. 232-243.

[18] E. L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.

[19] G. L. Nemhauser, and L. A. Wolsey, Integer and Combinatorial Optimization, Wiley Interscience series in Disc. Math. and Optimization, 1988.

[20] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H.B. Hunt III, Approximation algorithms for degree-constrained minimum-cost network-design problems, Algorithmica 31 (2001), 58-78.

[21] D. P. Williamson, M. X. Goemans, M. Mihail, and V. Vazirani, A primal-dual approximation algorithm for generalized Steiner network problems, Combinatorica 15 (1995), 435-454.

[22] S. Win, On a connection between the existence of k-trees and the toughness of a graph, Graphs and Combinatorics, 5 (1989), 201-205.