Check for updates

Chapter 38
Approximating the Minimum Degree Spanning Tree to within One from the Optimal Degree

Martin Fürer* Balaji Raghavachari†

Abstract

We consider the problem of constructing a spanning tree for a graph $G = (V, E)$ with n vertices whose maximal degree is the smallest among all spanning trees of G. This problem is easily shown to be NP-hard. We describe an iterative polynomial time approximation algorithm for this problem. This algorithm computes a spanning tree whose maximal degree is at most $O(\Delta^* + \log n)$, where Δ^* is the degree of some optimal tree. The result is generalized to the case where only some vertices need to be connected (Steiner case) and to the case of directed graphs. It is then shown that our algorithm can be refined to produce a spanning tree of degree at most $\Delta^* + 1$. Unless $P = NP$, this is the best bound achievable in polynomial time.

1 Introduction

The problem of computing a spanning tree of a graph that satisfies given constraints has been studied before [6], [7], [8], [19]. Depending on the constraints, it has been characterized by different complexity classes. Though the problem of computing a minimum cost spanning tree in a graph with edge weights is efficiently solvable both in the sequential and the parallel cases, even simple constraints on the structure of the resulting tree frequently make the problem NP-hard. A comprehensive list of NP-complete constrained spanning tree problems can be found in [12], [15].

Consider the minimum degree spanning tree (MDST) problem, which is that of constructing a spanning tree for a graph $G = (V, E)$ with n vertices whose maximal degree is the smallest among all spanning trees of G. This problem is easily shown to be NP-hard. Let T^* be an optimal spanning tree, whose maximal degree is Δ^*. We are interested in finding approximation algorithms for this problem which guarantee some nontrivial approximation quality. Fürer and Raghavachari [10] have given a parallel approximation algorithm which produces a spanning tree of degree $O(\Delta^* \log n)$. In this paper, the problem is reduced to that of computing a sequence of maximal matchings on some auxiliary graphs. In the Steiner version of this problem, along with the input graph, we are also given a distinguished set of vertices ($D \subseteq V$) and a solution is a tree of minimum degree which spans the set D. The minimum degree Steiner tree problem is more general and the MDST problem is the special case where $D = V$. In another earlier work, Agrawal, Klein and Ravi [5] have shown that even the more general minimum degree Steiner tree problem can be approximated to within a $\log n$ factor. They use approximations to the multicommodity flow problem in their solution.

An approximation algorithm finds applications in the computation of spanning trees in dynamic networks for non-critical broadcast. Solutions to the broadcast problem have mainly concentrated on how to complete the broadcasting as quickly as possible [11]. But there are instances like the distribution of mail and news on the Internet, in which the broadcast need not be executed on a priority basis. One of the parameters that different sites may want to reduce is the amount of work done by their site. Broadcasting informa-

*Computer Science Department, Pennsylvania State University, University Park, PA 16802, furer@cs.psu.edu

†Computer Science Department, Pennsylvania State University, University Park, PA 16802, rbk@cs.psu.edu

tion on a minimum degree spanning tree is one such solution. Another application for this problem is in the area of designing power grids. Here the cost of splitting the output of a station may grow rapidly with the degree of the split. We need reliable networks of small maximal degree. Additionally, this problem is interesting in its own right. Recently there has been a flurry of activity on approximating various *NP*-complete problems [3], [4], [2], [10], [16], [17], [18], [20], [21]. Our result shows that the minimum degree spanning tree problem is approximable within an additive term of one. The only other problems which we know to have similar approximation properties are the edge coloring problem [14], [22] and 3-colorability of planar graphs [1], [13].

In this paper, we start with an approximation algorithm which finds a spanning tree of degree at most $O(\Delta^* + \log n)$. We show that a similar bound is achievable in the case of directed graphs and in the Steiner version of the problem. We then present a refined algorithm which produces a spanning tree of degree at most $\Delta^* + 1$. We observe that unless $P = NP$, this is the best bound achievable in polynomial time.

The following are the main results of this paper.

THEOREM 1.1. *There is a polynomial time approximation algorithm for the minimum degree spanning tree problem which produces a spanning tree of degree* $O(\Delta^* + \log n)$.

THEOREM 1.2. *There is a polynomial time approximation algorithm for the minimum degree Steiner tree problem which produces a Steiner tree of degree* $O(\Delta^* + \log n)$.

THEOREM 1.3. *There is a polynomial time approximation algorithm for the directed version of the minimum degree spanning tree problem which produces a directed spanning tree of degree* $O(\Delta^* + \log n)$.

THEOREM 1.4. *There is a polynomial time approximation algorithm for the minimum degree spanning tree problem which produces a spanning tree of degree at most* $\Delta^* + 1$.

2 A Simple Approximation Algorithm

The first algorithm we present provides the fundamental ideas that we use in all our algorithms. We start with an arbitrary spanning tree T of G. We denote by $\rho(u)$ the degree of vertex u in T. We try to reduce the degrees of vertices with "large" degrees iteratively using the following scheme. Consider an edge (u, v) of G which is not in T. Let C be the unique cycle generated when (u, v) is added to T. Suppose there is a vertex w in C with the property that $\rho(w) \geq \max(\rho(u), \rho(v))+2$. We now introduce an "improvement" in T by adding the edge (u, v) and deleting one of the edges in C incident to w. Note that we called this step an improvement because the maximum of $\{\rho(u), \rho(v), \rho(w)\}$ has decreased by at least one.

DEFINITION 2.1. *A locally optimal tree (LOT) is a tree in which none of the non-tree edges produce any improvements. Its maximal degree is always denoted by* k.

One of the attractive ideas to approximate the MDST problem is to construct a LOT, and then ask the question, how far from an optimum could such a tree be? This idea poses a little difficulty. We are not aware of how such a local improvement algorithm could be implemented to run in polynomial time. However we wish to observe some crucial properties of LOTs which we utilize in our first algorithm.

Let k be the maximal degree of a LOT T. Let X_i be the set of vertices of degree i in T. We define groups of vertices S_i as follows:

$$S_i = \bigcup_{j=i}^{k} X_j \tag{2.1}$$

In other words, S_i contains those vertices of T which have degree at least i.

THEOREM 2.1. *For any* $b > 1$, *the maximal degree* k *of a locally optimal tree* T *is less than* $b\Delta^* + \lceil \log_b n \rceil$. *Hence* $k = O(\Delta^* + \log n)$.

Proof. First we note that for $i = k, k-1, \ldots,$ the rate of expansion of the sets S_i (the ratio $|S_{i-1}|/|S_i|$) can be larger than b at most $O(\log_b n)$ times in a row. To be more precise, there is an i with $k - \lceil \log_b n \rceil + 1 \leq i \leq k$ such that

$|S_{i-1}| \leq b|S_i|$. Otherwise

$$|S_{k-\lceil \log_b n \rceil}| > b^{\lceil \log_b n \rceil}|S_k| \tag{2.2}$$

which is a contradiction because every $|S_i| \leq n$, while $b^{\lceil \log_b n \rceil} \geq n$ and $|S_k| \geq 1$. Suppose we remove the vertices of S_i from T. This splits T into a forest F with t trees. As each vertex in S_i has degree at least i, we count at least $i|S_i|$ edges incident to these vertices. Also because T is a tree, at most $|S_i| - 1$ of these edges are within S_i itself and thus counted twice. Therefore we have

$$t \geq i|S_i| - 2(|S_i| - 1) \tag{2.3}$$

By the local optimality condition, every edge between trees in F is incident to at least one vertex of degree $i - 1$. We have identified $t + |S_i|$ components such that each edge between components is incident to at least one vertex of S_{i-1}. The vertices of S_{i-1} are called *critical* vertices and the set S_{i-1} is a witness to a lower bound of $\Omega(k - \log n)$ for Δ^*. In any spanning tree of G, there are at least $t + |S_i| - 1$ edges connecting these components and each one of these edges is incident to at least one vertex in S_{i-1}. Hence the average degree of vertices in S_{i-1} in any spanning tree of G is at least

$$\frac{t + |S_i| - 1}{|S_{i-1}|} \tag{2.4}$$

The maximal degree Δ^* is at least the average degree of these vertices. Hence equation (2.3) implies

$$\begin{aligned} \Delta^* &\geq \frac{t + |S_i| - 1}{|S_{i-1}|} \\ &\geq \frac{(i-1)|S_i| + 1}{b|S_i|} \\ &> \frac{i-1}{b} \end{aligned} \tag{2.5}$$

Combining equation (2.5) with the condition on the range of i, we get $k < b\Delta^* + \lceil \log_b n \rceil$. □

This can now be converted into a polynomial time algorithm with the desired performance as follows. Note that in the above theorem we used the local optimality condition only on "high" degree vertices. Therefore the same result holds for any tree satisfying the local optimality condition for the vertices in S_i with $i = k - \lceil \log_b n \rceil + 1$. We might call such a tree a *pseudo optimal tree* (POT). The following algorithm solves the problem in polynomial time. We start with an arbitrary spanning tree T of G. Let k be the maximal degree of T. In each phase, the algorithm tries to reduce the degree of some vertex whose degree is between k and $k - \lceil \log n \rceil$, using local improvement steps. Each phase of the algorithm can be implemented in polynomial time using standard techniques for searching graphs. The algorithm stops when no vertex in S_i has a local improvement. We use a potential function argument to show that the algorithm converges in a polynomial number of steps.

Proof of Theorem 1.1. We show that the iterative algorithm sketched above satisfies this theorem. The observation that followed Theorem 2.1 establishes that the degree of the resulting tree is bounded by $O(\Delta^* + \log n)$. We just need to show that the number of phases is bounded by a polynomial in n. Consider an exponential potential function ϕ on the vertex set of T. If the degree of a vertex u is d in the tree T, the potential $\phi(u)$ is defined to be c^d, for any constant $c > 2$. The total potential $\Phi(T)$ of the tree is defined to be the sum of the potentials of all the vertices. If k is the maximal degree of T,

$$\Phi(T) \leq nc^k \tag{2.6}$$

Any improvement step on T reduces the degree of some vertex in S_i for $i = k - \lceil \log n \rceil$. Therefore the reduction in potential due to any improvement is at least

$$\begin{aligned} \Delta\Phi &\geq (c^i + 2 \cdot c^{i-2}) - (3 \cdot c^{i-1}) \\ &= (c-1) \cdot (c-2) \cdot c^{i-2} \\ &> \epsilon \cdot \frac{c^k}{n} \quad \text{where} \quad \epsilon = \tfrac{(c-1)\cdot(c-2)}{c^3} \\ &\geq \epsilon \cdot \frac{\Phi(T)}{n^2} \end{aligned} \tag{2.7}$$

In other words, the potential reduces by at least a polynomial factor. Therefore in $O(n^2)$

steps, the potential reduces by a constant factor. Hence the number of phases is bounded by $O(n^3)$. Alternatively, we could argue that k cannot stay the same for more than n^2/ϵ phases. Also the value of k cannot decrease more than n times, again implying an $O(n^3)$ bound on the number of phases. As mentioned before, each phase can be implemented in polynomial time. Hence the above algorithm runs in polynomial time. □

3 The Steiner Problem

Consider a graph $G = (V, E)$ and a distinguished set of vertices $D \subseteq V$. The Minimum Degree Steiner Tree problem is that of finding a tree of minimum degree, which spans the set D. This is a generalization of the MDST problem. Agrawal, Klein and Ravi [5] have shown that this problem can be approximated in polynomial time to within a factor of $O(\log n)$ of the degree of an optimal tree. Their algorithm is based on reducing this problem to the multicommodity flow problem. They also provide approximations for computing the toughness of a graph, which was defined by Chvátal [9]. We provide an approximation algorithm for the Steiner problem which finds a Steiner tree whose degree is $O(\Delta^* + \log n)$. Our algorithms can also be used to give better bounds on the toughness of a graph.

We start with an arbitrary tree T which spans D and retain only those edges which separate the set D in T. Let W be the set of vertices spanned by T. For the spanning tree problem, the improvement step was just defined in terms of exchanging one edge for another edge. This idea is insufficient in this problem. We extend the exchanging idea as follows. Consider any path between two vertices in W which goes entirely through vertices of $V - W$ except for the end points. We call such a path as a *non-tree path.* Adding any non-tree path to T introduces a unique cycle. To make it into a tree again, we can remove from this cycle any one edge of T which is incident on a vertex of high degree. We also extend the notion of a locally optimal tree to the Steiner case.

DEFINITION 3.1. *A pseudo optimal Steiner tree (POST) is a Steiner tree with the following properties:*

1. *Every edge in the tree separates at least two distinguished vertices. In other words, there are no useless edges because all leaves of the tree are distinguished vertices.*
2. *None of the non-tree paths produce any improvement for any vertex in S_i for $i = k - \lceil \log n \rceil$, where k denotes the maximal degree of the tree. If this property is true for all i, then the tree is a locally optimal Steiner tree (LOST).*

THEOREM 3.1. *For any $b > 1$, the maximal degree k of a locally optimal Steiner tree T is at most $b\Delta^* + \lceil \log_b n \rceil$. Hence $k = O(\Delta^* + \log n)$.*

Proof. The proof is similar in structure to the proof of Theorem 2.1. As before, we find some i between k and $k - \lceil \log n \rceil$ where the ratio between $|S_i|$ and $|S_{i-1}|$ is bounded by a constant. For such a value of i, S_{i-1} forms a critical set and we can show that a large number of components need to be connected through this small set of vertices in any Steiner tree. This gives a lower bound on the average degree of these vertices, which in turn is a lower bound on Δ^*. Note that we need to use the fact that every leaf of T is a distinguished vertex (condition 1 of Definition 3.1) in a crucial way. We count the number of components formed in the tree by the removal of certain vertices. This condition ensures that every component has a distinguished vertex and needs to be connected to the others. Every possible connection needs to use the critical vertices that we identified. With the above ideas, the proof is a simple extension of Theorem 2.1 and is left to the reader. □

This idea can be converted into a polynomial time algorithm which produces a Steiner tree of degree $O(\Delta^* + \log n)$, satisfying Theorem 1.2. An iterative algorithm similar to the one for spanning trees runs in polynomial time and produces a Steiner tree whose degree is $O(\Delta^* + \log n)$.

4 Directed Spanning Trees

We now show how to extend our algorithm to handle directed graphs. In this case the input is a directed graph G together with a special vertex r which is the root of the tree. The root is reachable

from all vertices of the graph. A rooted spanning tree T of G is a subgraph of G with the following properties:

1. T does not contain any cycles.
2. The outdegree of every vertex except r is exactly one.
3. There is a path in T from every vertex to the root r.

The degree of a rooted spanning tree is the maximal indegree of any vertex. Note that directed graphs pose a severe problem. There is no easy way to define improvements in the case of directed graphs. Consider a non-tree edge (u, v) in G. In the case of undirected graphs, adding such an edge to T always produces a cycle and every vertex in that cycle (except u and v) can potentially benefit from this edge. This does not work for directed graphs.

For directed graphs we define an improvement step in the following way. Consider a vertex v of indegree i. We try to see if the indegree of the vertex can be reduced by attaching one of the i subtrees of v to another vertex of smaller degree. The improvement step consists of two parts. We first move the root of the subtree T' that is being removed from v to a "convenient" vertex in that subtree. T' is then attached to another vertex outside the tree to which the new root has a connection. The set of convenient vertices are those in the strongly connected part of the root of T' in the graph induced by the vertices of T' with all non-tree edges removed from vertices of degree $i-1$ or greater.

We define locally and pseudo optimal directed spanning trees as before. The algorithm tries to decrease the degrees of vertices in S_i for $i = k - \lceil \log n \rceil$ using the improvement step above. The following lemma is useful in the proof of Theorem 1.3.

LEMMA 4.1. *Let T be a directed spanning tree of degree k. Let S_i consist of those vertices whose indegree is at least i. Suppose we remove the vertices of S_i from T, breaking T into a forest F. Then there are at least $|S_i| \cdot (i-1) + 1$ trees of F whose vertices do not have descendents of degree i or greater in T.*

Proof. The proof uses induction on the size of S_i and follows from this simple observation. Suppose we build the tree from scratch, starting from the root. This produces the base of the induction with at least one leaf. Each addition of a vertex to the set S_i can remove at most one tree from the set of candidate trees, but adds at least i more. □

Proof of Theorem 1.3. As in the proof of the undirected case, we look for an i in the range k to $k - \lceil \log n \rceil$ where the set S_i does not "expand" by more than a constant factor. This leads to a critical set S_{i-1}. In the undirected case, we look at all the trees in the forest generated by the removal of S_i from T. This does not work right in directed graphs due to the possibility of some of these trees having descendent vertices in S_i. Hence we concentrate only on those trees whose vertices have no descendents of degree i or larger. By Lemma 4.1, there are at least $|S_i| \cdot (i-1) + 1$ such trees. These trees along with the vertices of S_i form the large number of components which have to use vertices of S_{i-1} in order to form a directed spanning tree. The rest of the proof is similar to Theorem 1.1 and is left to the reader. □

5 The $\Delta^* + 1$ algorithm

We now show how the previous idea can be refined into an algorithm which produces a spanning tree of degree at most $\Delta^* + 1$. The local optimality property that we used, is that no edge should be able to reduce the maximal degree of vertices on the basic cycle induced by that edge. Suppose k were the maximal degree of a spanning tree T. Observe that the local optimality condition is stricter than necessary. For an algorithm to show progress, it is sufficient to reduce the number of vertices of maximal degree. In doing so, the degrees of other vertices can increase arbitrarily as long as they remain less than k.

DEFINITION 5.1. *Let $(u, v) \notin T$ be an edge in G. Suppose w is a vertex in the cycle generated by adding (u, v) to T. If $\rho(u) \geq k - 1$, we say that u blocks w from (u, v). If neither u nor v blocks w, then (u, v) can be used to reduce the degree of w through a local improvement step. In such a case,*

we say w benefits from (u, v).

Let S be the set of vertices of degree k. The algorithm works in phases. In each phase we try to reduce the size of S by one. If successful, we move on to the next phase. As the size of S reduces by one in each phase (except the last one), there are at most $O(n/k)$ phases when the maximal degree is k. Summing up the harmonic series corresponding to reducing values of k proves that there are at most $O(n \log n)$ phases. We will show later that if it is not possible to reduce the size of S, then there is a set B of degree $k-1$ vertices in T such that, the tree T satisfies the conditions of the following theorem and hence the degree of T is at most $\Delta^* + 1$.

THEOREM 5.1. *Let T be a spanning tree of degree k of a graph G. Let Δ^* be the degree of a minimum degree spanning tree. Let S be the set of vertices of degree k. Let B be an arbitrary subset of vertices of degree $k-1$ in T. Let $S \cup B$ be removed from the graph, breaking the tree T into a forest F. Suppose G satisfies the condition that, there are no edges between different trees in F. Then $k \leq \Delta^* + 1$.*

Proof. As there are no edges in G connecting the different subtrees of F, the only way we can make a spanning tree is by connecting these clusters through vertices in S and B. By a simple counting argument, it is easy to show that F contains at least $|S|k + |B|(k-1) - 2(|S| + |B| - 1)$ subtrees. Therefore in any spanning tree of G, the average degree of vertices in $S \cup B$ is at least $k - 1 - (|B|-1)/(|S|+|B|)$. A vertex with maximal degree has at least average degree and hence every spanning tree has at least one vertex of degree at least $k-1$ in $S \cup B$. Therefore $\Delta^* \geq k - 1$. □

We observe that Theorem 5.1 is powerful and easily generalizes to the Steiner case as well as the directed case. Though we do not have polynomial time algorithms for these two problems which produce a tree of degree $\Delta^* + 1$, we believe that Theorem 5.1 points the way. Note that a tree T of degree k, along with a blocking set B is a proof that Δ^* is at least $k - 1$.

The following is a top-down view of our algorithm. Suppose we remove the set S_k, the set of all vertices of maximal degree from T. Observe the forest F generated. If there are no edges between the different components of F, we are done. Theorem 5.1 can be applied here and in this case $k = \Delta^*$. Otherwise any edge between components of F can be used to reduce the degree of some vertex in S_k. If such an edge is not blocked by a vertex of degree $(k-1)$, we make this improvement, reducing the number of degree k vertices by one and continue. Otherwise let w be a vertex of degree k and let u be a vertex which blocks w. Let F_u be the component of F which contains u. Suppose the degree of u can be reduced by one by running our procedure in the graph induced by the vertices of F_u. Then u is made non-blocking and in this case, we say that a sequence of improvements *propogate* to w. The following observation is crucial and shows why vertices do not interfere with each other in trying to become non-blocking.

Observation: Let u be a vertex which blocks w as described above. When u tries to become non-blocking, it is sufficient if it tries to look for improvements within the subgraph induced by the vertices of F_u. Any improvement that is possible for u by using edges going outside of F_u can be used directly to benefit a vertex from S_k.

The algorithm is implemented in a bottom up fashion as follows. We remove all vertices in $S_k \cup S_{k-1}$ from T and mark all the connected components as being *good.* All vertices in $S_k \cup S_{k-1}$ are marked as *bad.* If there are no edges between good components, the algorithm stops. In this case, we will show that the set of bad vertices remaining are witnesses to the fact that $k \leq \Delta^* + 1$. Otherwise let (u, v) be an edge between two good components F_u and F_v. We add (u, v) to T and observe the cycle generated. If there is a vertex of degree k in this cycle, we have identified a set of improvements which propogate to this vertex. Making these changes reduces the size of S_k by one. Otherwise there is at least one bad vertex of degree $(k-1)$ on the cycle. We mark all bad vertices on this cycle as good and make a union of all components on this cycle along with all the degree $(k-1)$ vertices. In all cases, we either find a way to reduce the degree of some vertex in

S_k or find a blocking set with which we can apply Theorem 5.1.

LEMMA 5.1. *Any vertex u marked good can be made non-blocking within the subgraph generated by the vertices of the good component of u.*

Proof. The proof proceeds by induction on the number of unions made by the algorithm. When the algorithm begins, only vertices of degree less than or equal to $(k-2)$ are marked as good. By definition, these vertices are non-blocking. The only time that a vertex u of degree $(k-1)$ was marked as good was when it was on the cycle generated by an edge added between two good components. We can add that edge and reduce the degree of u by one and make it non-blocking. Hence our algorithm maintains only vertices which can be made non-blocking within good components. Note that any update needed to make a vertex non-blocking is within the vertices of its component. □

LEMMA 5.2. *When the algorithm stops, $k \leq \Delta^* + 1$.*

Proof. Let S be S_k and B be the bad vertices of degree $k-1$. Note that the algorithm stops only when there are no edges between the good components. Hence the tree T along with these sets S and B satisfy the conditions of Theorem 5.1 and we get the desired result. □

Proof of Theorem 1.4. As observed earlier, there are at most $O(n \log n)$ phases. In each phase we try to find improvements which propagate to vertices of S_k. Lemma 5.1 assures that whenever we find a vertex w of degree k which can be marked as good, we can indeed find a sequence of improvements which propogates to w. Lemma 5.2 shows that when the algorithm stops the degree of the resulting tree is within one from the optimal degree. Each phase of the algorithm can be implemented in nearly linear time using the disjoint set union-find algorithm for maintaining connected components. Therefore the entire algorithm runs in $O(mn \log n \alpha(n))$, where m is the number of edges and α is the inverse Ackerman function. □

6 Conclusions

We have demonstrated iterative algorithms based on combinatorial techniques for approximating the minimum degree spanning tree and related problems. We believe that it is possible to implement our algorithms efficiently. The exact complexities of some of our algorithms are not clear. Is it possible to find a LOT or a LOST in polynomial time? As we mentioned earlier, we do not know how to find one of these in polynomial time. We obtained polynomial time approximation algorithms by using the local optimality condition on high degree vertices only. We are not aware of any example which would require more than polynomial time to find a LOT. We showed that a LOT is within an additive term of $\log n$ from the optimal. Is there a tighter bound? We have an example where the optimal tree is of degree 3 and there is a LOT of degree $\Omega(\frac{\log n}{\log \log n})$. Is there a better example? Another natural open question is to ask whether the $\Delta^* + 1$ algorithm can be extended to handle the Steiner case or the directed case. How well can parallel algorithms do in this problem? Is there an *NC* algorithm which can obtain a tree of degree $\Delta^* + 1$? The list of *NP*-complete problems which can be approximated to within one from the optimal is small. Is there some relationship among these problems?

References

[1] K. Appel and W. Haken, Every planar map is four-colorable, *Illinois J. Math.* **21** (1977), 429-567.

[2] A. Agrawal, P. Klein, S. Rao and R. Ravi, Approximation through Multicommodity Flow, Proc. of 31st FOCS (1990), 726-737.

[3] A. Agrawal, P. Klein and R. Ravi, When Trees Collide: An Approximation Algorithm for the Generalized Steiner Problem on Networks, Proc. of 23rd STOC (1991), 134-144.

[4] A. Agrawal, P. Klein and R. Ravi, Ordering Problems Approximated: Single-Processor Scheduling and Interval Graph Completion, Proc. of 18th ICALP (1991), LNCS 510, 751-762.

[5] A. Agrawal, P. Klein and R. Ravi, How Tough is the Minimum-degree Steiner Tree? A New Approximate Min-max Equality, Personal Communication.

[6] P.M. Camerini and G. Galbiati, The Bounded

Path Tree Problem, *SIAM J. Algeb. Disc. Meth.* **3**, (1982) 474-484.

[7] P.M. Camerini, G. Galbiati and F. Maffioli, Complexity of Spanning Tree Problems: Part I, *European J. Oper. Res.* **5** (1980) 346-352.

[8] P.M. Camerini, G. Galbiati and F. Maffioli, On the Complexity of Finding Multi-constrained Spanning Trees, *Disc. Appl. Math.* **5** (1983) 39-50.

[9] V. Chvátal, Tough Graphs and Hamiltonian Circuits, *Disc. Math.* **5** (1973), 215-228.

[10] M. Fürer and B. Raghavachari, An *NC* Approximation Algorithm for the Minimum Degree Spanning Tree Problem, Proceedings of the 28th Annual Allerton Conference on Communication, Control and Computing (1990), 274-281.

[11] H. Garcia-Molina and B. Kogan, An Implementation of Reliable Broadcast Using an Unreliable Multicast Facility, *Proc. Seventh Symp. on Reliable Dist. Syst.* (1988) 101-111.

[12] M.R. Garey and D.S. Johnson, "Computers and Intractability: A Guide to the Theory of *NP*-completeness," W.H. Freeman, 1979.

[13] M.R. Garey, D.S. Johnson and L. Stockmeyer, Some Simplified *NP*-complete Graph Problems, *Theor. Comput. Sci.* **1** (1976), 237-267.

[14] I. Holyer, *The NP-Completeness of Edge-Coloring*, *SIAM J. Comput.* **10** (1981) 718-720.

[15] D.S. Johnson, The *NP*-completeness Column: An Ongoing Guide, *J. Algorithms* **6** (1985) 145-159.

[16] F.T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos and S. Tragoudas, Fast Approximation Algorithms for Multicommodity Flow Problems, Proc. of 23rd STOC (1991), 101-111.

[17] F.T. Leighton and S. Rao, An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Applications to Approximation Algorithms, Proc. of 29th FOCS (1988), 422-431.

[18] J.K. Lenstra, D.B. Shmoys and E. Tardos, Approximation Algorithms for Scheduling Unrelated Parallel Machines, Proc. of 28th FOCS (1987), 217-224.

[19] C.H. Papadimitriou and M. Yannakakis, The complexity of restricted spanning tree problems, JACM 29 (1982), 285-309.

[20] S.A. Plotkin, D.B. Shmoys and E. Tardos, Fast Approximation Algorithms for Fractional Packing and Covering Problems, Proc. of 32nd FOCS (1991), 495-504.

[21] H. Saran and V.V. Vazirani, Finding a k-Cut within Twice the Optimal, Proc. of 32nd FOCS (1991), 743-751.

[22] V.G. Vizing, *On an estimate of the chromatic class of a p-graph (Russian)*, *Diskret. Anal.* **3** (1964) 25-30.