The Directed Minimum-Degree Spanning Tree Problem*

Radha Krishnan and Balaji Raghavachari

Computer Science Department, University of Texas at Dallas,
Richardson, TX 75083-0688, USA.
{Radha.Krishnan,Balaji.Raghavachari}@utdallas.edu

Abstract. Consider a *directed* graph $G = (V, E)$ with n vertices and a root vertex $r \in V$. The DMDST problem for G is one of constructing a spanning tree rooted at r, whose maximal degree is the smallest among all such spanning trees. The problem is known to be NP-hard. A quasi-polynomial time approximation algorithm for this problem is presented. The algorithm finds a spanning tree whose maximal degree is at most $O(\Delta^* + \log n)$ where, Δ^* is the degree of some optimal tree for the problem. The running time of the algorithm is shown to be $O(n^{O(\log n)})$. Experimental results are presented showing that the actual running time of the algorithm is much smaller in practice.

1 Introduction

The minimum-degree spanning tree (MDST) problem for an undirected graph $G = (V, E)$ is that of constructing a spanning tree of G, whose maximal degree is the smallest among all spanning trees of G. It is a generalization of the Hamiltonian Path problem and thus is also NP-hard. The problem can be defined on directed graphs as follows. Given a root vertex $r \in V$, find an incoming (or outgoing) spanning tree rooted at r, known as a branching, in which the maximal indegree (outdegree) of a vertex is minimized. We will refer to the directed version of the MDST problem as the DMDST problem. In the Steiner case, a set of terminals $D \subseteq V$ is also specified. The output tree must span the set D. However, it may contain any of the other vertices of G.

The general problem of computing low-degree trees is both fundamental and finds ready applicability, such as in noncritical broadcast and VLSI layout problems. It is also inherently appealing due to its seeming simplicity. Previous polynomial-time algorithms [2] for the DMDST problem find trees whose degree is at most $O(\Delta^* \log n)$, i.e., a *factor* of $\log n$ from the optimal degree Δ^*. On the other hand, the algorithm in [4] for *undirected* graphs finds a tree whose degree is at most $\Delta^* + 1$, i.e., an additive constant 1 away from the optimal. Our work tries to bridge this gap in performance of approximation algorithms for the undirected and directed versions of the MDST problem.

* Research supported by the National Science Foundation under grant CCR-9820902.

Our algorithm for the DMDST problem finds a tree whose maximal degree is at most $c\Delta^* + \lceil \log_c n \rceil = O(\Delta^* + \log n)$, for any constant $c > 1$. The approximation quality has therefore been improved from a multiplicative factor of $\log n$ to an additive term of $\log n$. The running time of the algorithm is shown to be quasi-polynomial, $O(n^{\log_c n + O(1)})$. However, it is conjectured that a better bound may be provable, and also that the running time may be much better in practice for this reason. We present experimental evidence to show that in practice the algorithm ran much faster than the theoretical bound obtained here. Also, the degree of the tree output is often very close to the optimal degree.

Previous work in undirected graphs. The first result on approximating a minimum-degree spanning tree was that of Fürer and Raghavachari [2]. They gave a polynomial-time approximation algorithm that returns a tree whose degree is at most $O(\Delta^* \log n)$. The first polynomial-time approximation algorithm for the Steiner version of the problem was provided by Agrawal, Klein and Ravi [1]. Their approximation ratio is $O(\log |D|)$, where D is the set of terminals. Ravi, Raghavachari and Klein [12] studied generalizations of the MDST problem and gave quasi-polynomial time approximation algorithms. Fürer and Raghavachari [4] improved their previous results and provided a new polynomial-time algorithm to approximate the MDST problem to within one of optimal. Their algorithm also extends to the Steiner version of the problem, but only works on undirected graphs. A survey of these and other minimum-degree problems has appeared in a book on approximation algorithms [9].

Ravi, Marathe, Ravi, Rosenkrantz and Hunt [11] proposed algorithms for computing low-weight bounded-degree subgraphs satisfying given connectivity properties. Given a graph G with nonnegative weights on the edges, and a degree bound Δ, their algorithm computes a spanning tree of G whose degree is at most $O(\Delta \log \frac{n}{\Delta})$ and whose weight is at most $O(\log n)$ times the weight of a minimum-weight tree with degree at most Δ. Their techniques extend to the Steiner tree and generalized Steiner tree problems with the same ratio. They also studied special cases when the edge weights satisfy the triangle inequality and presented efficient algorithms for computing subgraphs that have low weight and small bottleneck cost. More recently, Könemann and Ravi [7] have given an algorithm that finds a tree of degree $O(\Delta + \log n)$ whose cost is at most $O(1)$ times the cost of an optimal tree of degree Δ.

Given a graph G and an independent set of nodes I, the problem of finding a spanning tree that minimizes the maximum degree of any node in I is solvable in polynomial time [8]. Gavish [5] formulated the MDST problem as a mixed integer program and provided an exact solution using the method of Lagrangian multipliers. Ravi [10] presented an approximation algorithm for the problem of finding a spanning tree whose diameter plus maximal degree is a minimum.

Previous work in directed graphs. Fürer and Raghavachari [2] showed that their algorithm for computing low-degree trees further generalizes to find branchings in directed graphs. Their algorithm builds a tree in stages by taking the union of a sequence of low-degree forests (e.g., matchings), and the degree of the resulting tree is shown to be $O(\Delta^* \log n)$. This paper improves the performance from a multiple of $\log n$ to an additive term of $\log n$.

2 Definitions and Notation

The input 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. Let T^* be an optimal DMDST whose maximal degree is Δ^*.

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 and r has no outgoing edges. It is easily shown that the only subgraphs with $n - 1$ edges in which there is a directed path from every vertex to r is a branching rooted at r. Sometimes this is also known as an in-branching. One can also define 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. However, our algorithm can be easily modified to find out-branchings with small outdegree.

Let T be a branching. For each edge (v, w) in T, we call w as the *parent* of v, denoted by $p[v]$. Since every vertex except r has a unique outgoing edge, 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 *descendent* 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 it as its *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. Our goal is to find a branching of as small a degree as possible.

3 MDST Problem: Directed vs Undirected Graphs

Our algorithm is based on an algorithm proposed by Fürer and Raghavachari [3] for undirected graphs that finds a tree whose degree is $O(\Delta^* + \log n)$. Their algorithm starts with an arbitrary spanning tree of G, and iteratively decreases the degree of high-degree vertices by applying "improvement" steps. An improvement step involves replacing an edge incident to a high-degree node by another edge that keeps the tree connected. They applied improvement steps to high-degree nodes repeatedly, until no improvement was possible at these nodes. In order to extend this algorithm to directed graphs, we make several modifications.

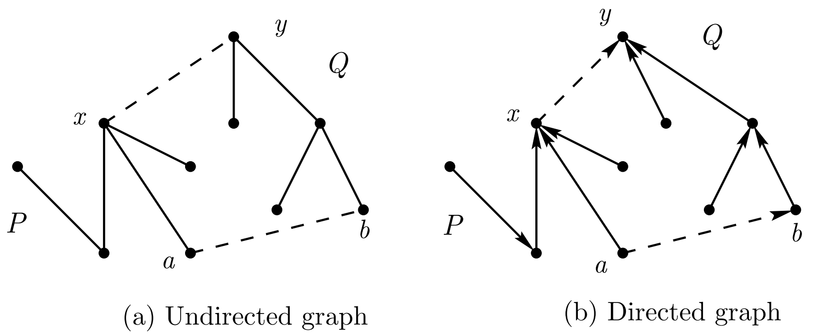


Fig. 1. Directed versus undirected graphs

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. 1). 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 Q. Otherwise the procedure would not yield a branching of G. To illustrate, as shown in Fig. 1 (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. 1 (b), only the edge (x, y) yields a branching. If (a, b) is added, then a has two outgoing edges, and the resulting graph is not a branching. Therefore the improvement step needs to be modified.

Second, the analysis of the algorithm for undirected graphs uses the notion of "witness sets". A witness set is a small set of nodes whose removal splits the graph into a large number of connected components. The ratio of the number of components to the number of nodes in the witness set is a lower bound on Δ^*. It was shown by Fürer and Raghavachari [3] that there are witness sets that can be used to find a tree whose degree is at most $\Delta^* + 1$. We will show that the notion of a witness set must also be modified for the case of directed graphs.

3.1 Witness Sets

The minimum ratio of the cardinality of a vertex set W to the number of components that are generated when W is removed from the graph is called the *toughness* of the graph. Win [13] has shown the following interesting relationship between the toughness of a graph and the MDST 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 $k > 2$). Vertex sets for which the above ratio is close to the toughness of the graph are called witness sets. Fürer and Raghavachari [3] used such witness sets to establish a lower bound on the degree of an optimal tree for the MDST problem. They showed that if there is a witness set of size w whose removal splits G into t components then $\Delta^* \geq \lceil \frac{w+t-1}{w} \rceil$.

This 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. 2). 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.

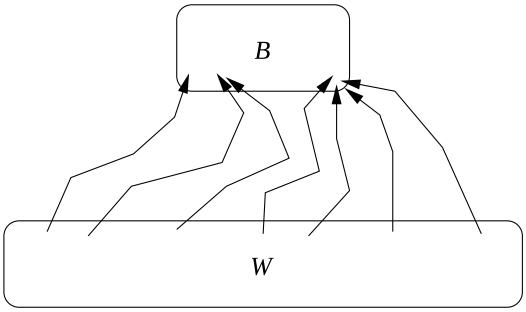


Fig. 2. Paths from W are internally disjoint

Lemma 1. *Let $G = (V, E)$ be a directed graph and $r \in V$. Suppose there are subsets of vertices $W \subset V$ and $B \subset V$ 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.*

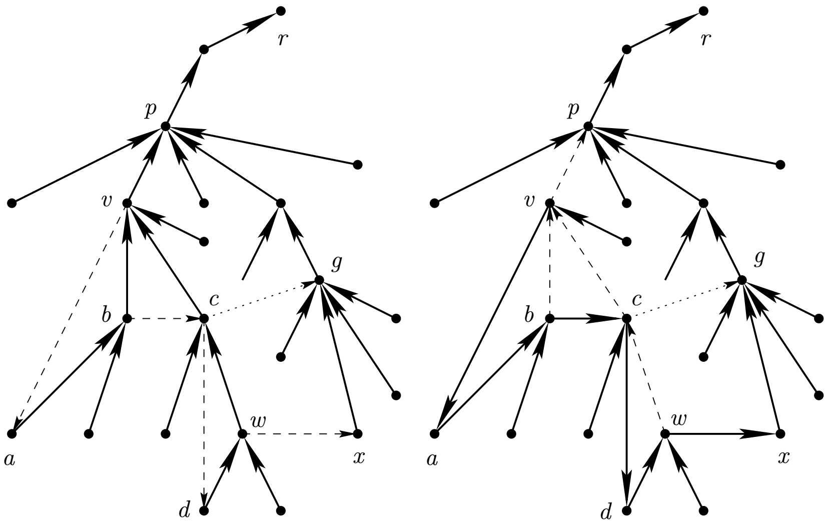
Then the degree of an DMDST rooted at r of G satisfies, $\Delta^* \geq \lceil |W|/|B| \rceil$.

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 $\{P_v : v \in V - \{r\}\}$ 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.

4 The DMDST 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 reach r. We perform an improvement step only if after the improvement, vertices whose degrees increased have a smaller degree than p.



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

Fig. 3. Example of an improvement applied to vertex p

Fig. 3 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. If v can find an alternate path to r so that the edge (v, p) may be deleted from T, the degree of p can be decreased to 4. The edge (c, g) is deleted because the indegree of g is already 4, and if we choose 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 — ($v \to a \to b \to c \to d \to w \to x \to \ldots \to r$) and the algorithm uses this path to modify the branching; the new branching is shown in Fig. 3 (b). The indegree of p has thus been successfully 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 $(y, p[y])$ 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 r.

```
Procedure IMPROVEMENT($T, v, p$)

1. Delete $(v, p)$ from $G$.
2. Let $\Delta$ 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
   $(a, p[a])$ in $T$ by $(a, b)$.
7. Restore all edges of $G$ and return True.
```

We now consider the DMDST algorithm. The 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.

```
Algorithm DMDST($G, r$)

1. Find a branching $T$ of $G$ rooted at $r$. Let its degree be $k$. Fix some constant
   $c > 1$.
2. For each edge $(v, p) \in T$, run IMPROVEMENT($T, v, p$) if the degree of $p$ in $T$
   is more than $k - \lceil \log_c n \rceil$. 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 Analysis of the Algorithm

The analysis of the running time of the algorithm uses potential functions that were introduced by Fürer and Raghavachari [3], and adapted by Ravi, Raghavachari and Klein [12] and Könemann and Ravi [7]. In fact our analysis of the running time is almost the same as in [12]. The potential of a vertex of degree Δ is defined to be n^{Δ}, and therefore the total potential of all the vertices is at most n^{k+1}, where k is the current degree of T. An improvement is applied to a vertex of degree $\Delta > k - \lceil \log_c n \rceil$. Each improvement step that targets a vertex of degree Δ reduces the total potential by at least $n^{\Delta-2}$, since the degree of a node of degree Δ is reduced by 1 and the degree of all the other nodes may increase to $\Delta - 1$. Since $\Delta > k - \lceil \log_c n \rceil$, this reduction in potential is at least a fraction $n^{-\log_c n - 3}$ of the current potential of the branching. It follows that the number of improvement steps is at most $n^{\log_c n + 3}$. Each improvement step can be implemented in $O(n^3)$ time, thus giving a total running time of $O(n^{\log_c n + 6})$ for the algorithm.

The following lemma relates the running time of the algorithm and the number of improvement steps, I. This expression for the running time is a more meaningful measure, since our experiments show that I grows only linearly with n, making the observed running time $O(n^4)$.

Lemma 2. *The running time of Algorithm DMDST is $O(n^3 I)$, where I is the number of improvement steps.*

The analysis of the degree bound of the tree output by the algorithm is more interesting. For this analysis, the notion of witness sets that was used by the algorithm for undirected graphs has to be strengthened. In directed graphs, there may be edges in the "wrong" direction that don't help in constructing a branching, but they may stop the graph from falling apart when a few critical vertices are removed.

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 $k - \lceil \log_c n \rceil$.

Lemma 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 $|S_\Delta| > 1$, remove a node $v \in S_\Delta$ and all its descendents from T such that v has no descendents in S_Δ (except itself). 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, the set increases by at least Δ, and the number of nodes that we get is $(\Delta - 1)(|S_\Delta| - 1) + \Delta = (\Delta - 1)|S_\Delta| + 1$.

Lemma 4. *Let T be the branching output by our algorithm. Let its degree be k. Then for any $k - \lceil \log_c n \rceil < \Delta \leq k$,*

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

Proof. Let W be the set of vertices as in Lemma 3 that are children of nodes in S_Δ, but have no descendents 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 $p = p[v]$. 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 is entirely contained in C_v, the descendents 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 1. Therefore,

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

Theorem 1. *The degree of the branching returned by our algorithm is at most $c\Delta^* + \log_c n$, where $c > 1$ is the constant in Step 1 of the DMDST algorithm.*

Proof. Lemma 4 establishes a set of lower bounds on Δ^* for $\lceil \log_c n \rceil$ different values of Δ. At least for one of these values of Δ, $|S_{\Delta-1}| \leq c|S_\Delta|$. Using this value of Δ, we get $k \leq c\Delta^* + log_c n$.

6 Experimental Results

The algorithms were implemented in C, using Knuth's Stanford GraphBase toolkit [6] and tested on large numbers of randomly generated graphs. These input random graphs actually followed two different patterns, as described below individually. The running time clearly depends upon the initial tree, and this was indeed observed in the experimental study. We tried to generate the initial tree using both depth-first search (DFS) and breadth-first search (BFS). BFS tends to generate high degree nodes and therefore the number of improvement steps tends to be much higher than if the initial tree was generated by DFS. In dense graphs, DFS generally finds low-degree trees by itself. While we found that our algorithm further reduces the degree of the initial DFS tree significantly, we present experimental results with an initial BFS tree, because we wished to capture the worst-case performance of the algorithm.

6.1 Uniformly Distributed Random Graphs

For this class of randomly generated input graphs, the probability of existence of an edge between two vertices is set to be a constant. The number of vertices were varied from 100 to 9000. A small number of runs were on 20,000-node graphs. We also varied the density of the graph. This class of graphs tends to be Hamiltonian and a good algorithm should be able to find a low-degree branching. We observed that our algorithm also has no difficulty in achieving this.

The results for this class of graphs is presented in Fig. 4, which shows the number of improvement steps as a function of n. The x-axis shows n and the

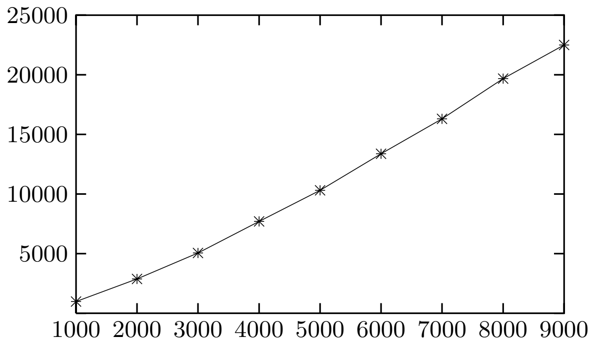


Fig. 4. Number of improvement steps versus n

y-axis shows the average number of improvement steps in random graphs with uniform edge probability. For each value of n, the algorithm was run on several random instances and the average number of improvement steps over these instances was used to generate this plot. Note that the number of improvement steps is almost linear in n.

In all of our test cases, we found that the algorithm always found a branching of degree two or less, irrespective of how large n was or how bad the initial degree of the tree was. For problems of small size (n less than 100) the algorithm would often return a branching of degree one, i.e., a Hamiltonian path.

Fig. 5 shows the degree of the intermediate trees as the algorithm progresses on a 20,000-node random graph with uniform edge probability. Observe that the algorithm makes rapid progress initially since there are very few vertices of high degree. Once the degree becomes small, a larger number of improvements are needed to decrease the degree of the tree. Note that if one terminated the algorithm earlier (say, to meet a fixed deadline) then the current tree can be used without a big sacrifice on the quality. The shape of the curve shows that we get about 50% of the progress in about 10% of the time.

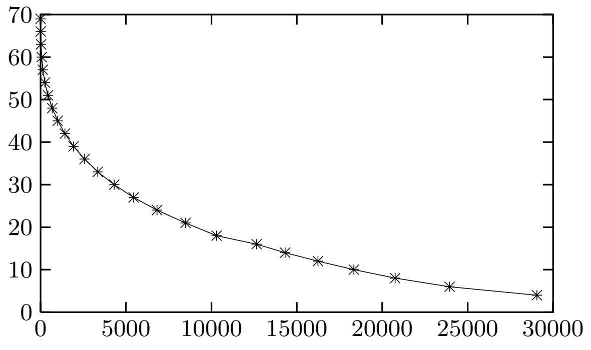


Fig. 5. Degree of tree versus number of improvements on a 20,000 node graph

6.2 Graphs with a Hidden Hamiltonian Path

The second class of graphs we tested were deliberately constructed to provide "bad" inputs to the algorithm to really test the algorithm's power and efficacy. The graphs in this class were generated by first obtaining a random bipartite graph with n_1 vertices on one side and n_2 vertices on the other side (say $n_1 < n_2$). The ratio of n_2/n_1 was varied (while holding $n = n_1 + n_2$ constant). We finally added to each of these bipartite graphs a random Hamiltonian path. Without the "hidden" path that we added at the end, the degree of any branching in the bipartite graph is at least n_2/n_1, since vertices in the tree must alternate between the two sides. We wanted to see if the algorithm finds this hidden path.

The algorithm performed very well even in these bad input instances, always returning no more than a degree 2 tree in the end. As shown in Fig. 6 the number of improvement steps varies significantly with the ratio n_2/n_1, a measure of the badness of the input graph. As expected, the algorithm takes longer as the ratio gets closer to 1. We observed that in all cases, the number of improvements is a small multiple of n.

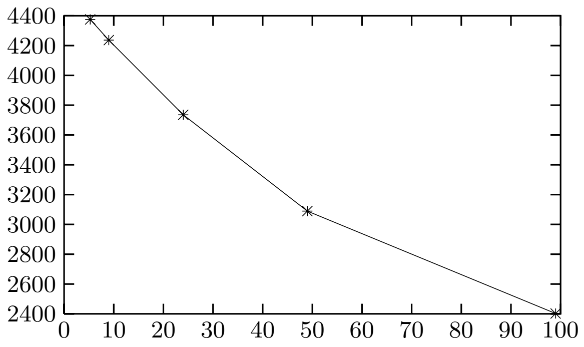


Fig. 6. The x-axis shows the ratio of n_2/n_1 and the y-axis shows the average number of improvement steps for bipartite graphs with a total of 1000 vertices. The input graphs for this plot were randomly generated bipartite graphs that were augmented with a hidden Hamiltonian path.

7 Conclusions

We have presented an approximation algorithm for the directed minimum-degree spanning tree problem. We introduced a new notion of witness sets that works in directed graphs. Though we couldn't prove a polynomial running time for our algorithm, it is likely to be fast in practice as shown by the experimental evidence. There are several open questions that follow from this work. Is it possible to implement our algorithm to run in polynomial time? Currently, the Steiner version of the DMDST problem does not have an approximation algorithm even with an $O(\log n)$ approximation ratio.

We describe an example that shows why our current approach fails in the Steiner version of the DMDST problem. In this example, there is a high-degree node p of degree k. Its children are $c_1, \ldots, c_k$. Each of these k children have an edge into vertex s, which is not in the current Steiner tree, and s has an edge into p. It can be verified that there is no improvement possible for vertex p. But, the degree of p can be reduced to $\lfloor \frac{k}{2} \rfloor + 1$ by connecting $\lfloor \frac{k}{2} \rfloor$ of p's children through s. In fact if the graph has a number of other extra nodes similar to s, the degree of p can be reduced even to 2. This example shows that our algorithm does not guarantee any performance bound on the degree of the tree for the Steiner case. The reason that we were unable to apply Lemma 1 is that, the paths from c_1 through c_k (the nodes in W) to r intersect each other at s, before reaching p (which forms the set B), thus violating Condition 2 of the lemma.

References

1. A. Agrawal, P. Klein, and R. Ravi, How tough is the minimum-degree Steiner tree? A new approximate min-max equality, TR CS-91-49, Brown University, 1991.
2. M. Fürer and B. Raghavachari, An NC approximation algorithm for the minimum degree spanning tree problem, In *Proc. of the 28th Annual Allerton Conf. on Communication, Control and Computing*, pages 274–281, 1990.
3. M. Fürer and B. Raghavachari, Approximating the minimum degree spanning tree to within one from the optimal degree, In *Proc. of 3rd ACM-SIAM Symp. on Disc. Algorithms (SODA)*, pages 317–324, 1992.
4. M. Fürer and B. Raghavachari, Approximating the minimum-degree Steiner tree to within one of optimal, *J. Algorithms*, 17:409–423, 1994.
5. B. Gavish, Topological design of centralized computer networks – formulations and algorithms, *Networks*, 12:355–377, 1982.
6. D. E. Knuth, The Stanford GraphBase: a platform for combinatorial computing, Addison Wesley, 1993.
7. J. Könemann and R. Ravi, A matter of degree: improved approximation algorithms for degree-bounded minimum spanning trees, In *Proc. of 32nd annual ACM Symposium on Theory of Computing (STOC)*, pages 537-546, 2000.
8. E. L. Lawler, Combinatorial optimization: networks and matroids, Holt, Rinehart and Winston, New York, 1976.
9. B. Raghavachari, Algorithms for finding low degree structures, In "Approximation algorithms," D. Hochbaum (ed.), PWS Publishers Inc., pages 266-295, 1996.
10. R. Ravi, Rapid rumor ramification: approximating the minimum broadcast time, In *Proc. of 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 202–213, 1994.
11. R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III, Many birds with one stone: multi-objective approximation algorithms, In *Proc. of 25th Annual ACM Symp. on the Theory of Computing (STOC)*, pages 438–447, 1993.
12. R. Ravi, B. Raghavachari, and P. Klein, Approximation through local optimality: designing networks with small degree, In *Proc. of 12th Conf. on Foundations of Software Tech. and Theoret. Comp. Sci. (FSTTCS)*, pages 279–290. Lect. Notes in Comp. Sci. 652, 1992.
13. S. Win, On a connection between the existence of k-trees and the toughness of a graph, *Graphs and Combinatorics*, 5:201-205, 1989.