The Full Degree Spanning Tree Problem*

Randeep Bhatia† Samir Khuller‡ Robert Pless§

Yoram J. Sussmann¶

Computer Science Department and Institute for Advanced Computer Studies

Univ. of Maryland, College Park, MD 20742.

Abstract

The full degree spanning tree problem is defined as follows: given a connected graph $G = (V, E)$ find a spanning tree T to maximize the number of vertices whose degree in T is the same as in G (these are called vertices of "full" degree). This problem is NP-hard. We present almost *optimal* approximation algorithms for it assuming $coR \neq NP$. For the case of general graphs our approximation factor is $\Theta(\sqrt{n})$. Using Håstad's result on the hardness of approximating clique, we can show that if there is a polynomial time approximation algorithm for our problem with a factor of $O(n^{\frac{1}{2}-\epsilon})$ then $coR = NP$. Additionally, we present two algorithms for optimally solving small instances of the general problem, and experimental results comparing our algorithm to the optimal solution and the previous heuristic used for this problem.

KeyWords: Graph, Algorithm, Approximation, Spanning Tree, NP-Hardness, Full Degree.

---

*A preliminary version of this paper appeared in the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (1999).

†Currently at Bell Labs, Lucent Technologies, Murray Hill, NJ 07974 E-mail:randeep@research.bell-labs.com. This research was done while this author was at the University of Maryland.

‡Research supported by NSF CAREER Award CCR-9501355. E-mail:samir@cs.umd.edu.

§E-mail:pless@cs.umd.edu.

¶Research supported by NSF CAREER Award CCR-9501355. E-mail:yoram@cs.umd.edu.

1 Introduction

In this paper we study the problem of computing a spanning tree T in a connected graph $G = (V, E)$ to maximize the number of vertices of *full* degree. These are vertices whose degree in the tree T is the same as their degree in the graph G. This problem was first mentioned by Lewinter [8]. Tree optimization problems have been very well studied (see [3] for a survey).

The problem arises as a central problem in the work by Pothof and Schut [11] which has to do with water distribution networks. To measure flow in pipes, you can install flow-meters on edges. However, you do not need to install flow-meters on all edges in the network. If you install flow meters on edges of a cotree (a cotree H is a set of edges in the graph, such that the deletion of edges in H leaves a spanning tree in the graph), then we can *infer* the flow on the edges of the spanning tree due to flow conservation (and the flow into and out of terminals). This requires that we install exactly $m - n + 1$ flow meters where m is the number of edges and n is the number of vertices in the network, since every cotree has $m - (n - 1)$ edges. Unfortunately, "the cost of flow meters is several times the cost of pressure meters" [11]. Pressure meters are installed at vertices, and one can compute the flow on an edge by measuring the pressures on both the incident vertices. Thus we are looking for a cotree that is incident on as few vertices as possible (to minimize cost). The *savings* is exactly the number of full degree vertices in a spanning tree, since these vertices have zero degree in the corresponding cotree and we do not have to install pressure meters at these vertices. Hence we would like to maximize the number of vertices of full degree. One pressure meter is installed at each vertex that does not have full degree; by using these meters, we can compute the flow on each edge in the cotree, and then infer the flow on each edge in the tree due to flow conservation (and the flow into and out of terminals).

Previous Work: Pothof and Schut [11] suggest the following heuristic to maximize the number of full degree vertices. Define the weight of each edge to be the sum of the degrees of the incident

vertices, and compute a minimum weight spanning tree. As discussed in their paper, there are cases when their algorithm does very poorly. Consider a wheel graph (a cycle with an additional central vertex with edges to all vertices on the cycle); then their algorithm may find no vertices of full degree! In fact, for this particular graph our full degree spanning tree algorithm (for arbitrary graphs) finds a spanning tree with $\frac{n}{3}$ vertices of full degree.

Our Results: The full degree spanning tree problem is NP-hard [1, 2]. We consider approximation algorithms for this problem — these are algorithms that have a polynomial running time, and produce a suboptimal solution. The ratio of the optimal solution to the solution we produce (for a maximization problem) is referred to as the approximation ratio ρ.

In Subsection 2.1 we provide an algorithm with an approximation ratio of $O(\sqrt{n})$ for the Full Degree Spanning Tree (FDST) problem. In Subsection 2.2 we show that if there is a polynomial time algorithm for the FDST problem with a worst case approximation ratio of $O(n^{\frac{1}{2}-\epsilon})$, then $coR = NP$. This proves that our worst case approximation ratio cannot be improved significantly, if NP-complete problems do not have randomized polynomial algorithms. Section 3 discusses absolute bounds on the size of the FDST solution.

In Section 4 we give two algorithms to compute the optimal solution for small problem instances, and compare and contrast our heuristic with the one given by Pothof and Schut [11]. For large random graphs (we tried various kinds of general graphs and planar graphs generated by LEDA [9] and Stanford Graphbase [7]) we found that our heuristic consistently gave better solutions. We also performed tests to compare our algorithm's solution to the optimal solution. For random planar graphs, we found that in all cases except one, our heuristic gave the optimal solution. For arbitrary random graphs, on half the tests our algorithm found the optimal solution, and on the other half of the tests it missed by one vertex (in one case it missed by two vertices).

Another paper has independently considered the same question, giving algorithms to find exact

solutions to the FDST problem for interval graphs, cocomparability graphs, graphs of bounded asteroidal number, and bounded tree-width graphs, and giving a polynomial time approximation scheme for this problem when the input graph is planar [2].

2 Approximation Algorithms

2.1 Full Degree Spanning Tree Algorithm

Let the given graph be $G = (V, E)$ and let G have n vertices. Let G^1 be a graph defined on vertex set V: $G^1 = (V, E^1)$, where $E^1 \subseteq E$. Let $X \subseteq E$ be a set of edges. We define $G^1 \oplus X$ ($G^1 \ominus X$) to be the graph with edge set $E^1 \cup X$ ($E^1 \setminus X$) and vertex set V. Let $Y \subseteq V$ be a set of vertices. We define $E_Y \subseteq E$ to be the set of all edges at least one of whose endpoints is incident on some vertex in Y. We define $G^1 \oplus Y$ ($G^1 \ominus Y$) to be the graph $G^1 \oplus E_Y$ ($G^1 \ominus E_Y$). These operations are augmentation (reduction) of G^1 by a given set of vertices or edges. Let $N(v)$ be the set of neighbors of vertex v.

We now present the **Greedy Star-Insertion Algorithm** that finds a good solution to the FDST problem. The algorithm is extremely simple. We show how to implement it efficiently, and then prove an $O(\sqrt{n})$ bound for its worst case approximation factor.

High Level Description: The algorithm first sorts the vertices in nondecreasing order of degree, then considers them one-by-one to be added to the current solution. Let S be the full degree vertices (initially, S is empty) and F the spanning tree that we are constructing. At each step we insert a vertex into S, together with its incident edges (a "star"), as long as it does not create a cycle. In some sense the algorithm is similar to Kruskal's MST algorithm, except that we are inserting vertices (along with all their incident edges) rather than simply adding single edges at each step.

The main hurdle regarding an efficient implementation is to check if the vertex v_i we are considering can be legally inserted or not, without creating a cycle in F. The edges incident on v_i are in two categories: edges already in F and edges not in F as yet. If the set of vertices $\{v_i\} \cup \{u | (v_i, u) \notin F \text{ and } u \in N(v_i)\}$ all belong to distinct connected components of F, then we can insert v_i without creating a cycle in F. However, if two vertices belong to the same component, then adding the edges incident to v_i will create a cycle in F.

The algorithm uses the well known Union-Find data structure (see Tarjan [12]) to maintain the connected components induced by the current spanning forest F. Maintaining also a bit vector encoding the edges in F allows the check ($e \in F$?) to be done in constant time. The algorithm scans the adjacency list of the current vertex v_i and, for each adjacent edge not already in the forest, checks to see which component the opposite vertex lies in. If all the components thus discovered along with the component in which v_i lies are distinct, then v_i can be added to the current solution. We do this check using an array $Validate[]$. While processing v_i, when we scan an edge $(v_i, u) \notin F$, and u belongs to component c, we set $Validate[c] = i$. If for some other edge $(v_i, u') \notin F$, u' also belongs to component c, then we can detect that we are trying to write i in location $Validate[c]$ that already contains i. This will detect the case when two neighbors of v_i belong to the same component.

A detailed description of the algorithm is given in Fig. 1. We assume that our input is a graph $G = (V, E)$, and the output is a spanning tree F of G and a set of vertices S that have full degree in F.

Theorem 1: The **Greedy Star-Insertion Algorithm** delivers a solution of size at least $\frac{OPT}{2\sqrt{2n}}$ where OPT is the size of the optimal solution. Moreover, the algorithm has a worst case time bound of $O(m\alpha(m, n))$ where n, m denote the number of vertices and edges in the graph, respectively, and $\alpha(m, n)$ is the inverse Ackermann function [12].

Greedy Star-Insertion Algorithm

$S \leftarrow \emptyset$.
$F \leftarrow (V, \emptyset)$.
For $i = 1$ to n
    $Validate[i] \leftarrow 0$.
Sort the vertices in nondecreasing order of degree.
Let the sorted list of vertices be $v_1, v_2, \ldots, v_n$.
For each vertex v_i **do** (* scan vertices in degree order *)
    $Validate[Find(v_i)] \leftarrow i$.
    $Flag \leftarrow TRUE$.
    For each edge $e = (v_i, u) \notin F$ **do** (* scan new edges *)
        $c \leftarrow Find(u)$.
        If $Validate[c] = i$ **then** (* cycle will be created on adding v_i*)
            $Flag \leftarrow FALSE$.
        Else $Validate[c] \leftarrow i$.
    If $Flag$ **then** (* insert v_i into S and F *)
        $c \leftarrow Find(v_i)$.
        For each edge $e = (v_i, u) \notin F$ **do**
            $Union(c, Find(u))$.
        $F \leftarrow F \oplus \{v_i\}$.
        $S \leftarrow S \cup \{v_i\}$.
End
Add edges to F to make it a spanning tree.
Output F, S.

Figure 1: Algorithm for finding a good FDST

Proof: Let T_{OPT} be an optimal solution to the FDST problem. Let OPT be the number of vertices of full degree. Let d be a degree threshold. Let A_d (B_d) be the set of full degree vertices of degree $\leq$ ($>$) d in an optimal solution. Hence $OPT = |A_d| + |B_d|$. We will bound A_d and B_d separately. Let I_d denote the set of vertices of degree $\leq d$ in S, where S is the set of full degree vertices obtained by the Greedy Star-Insertion Algorithm. Note the vertices in I_d have full degree in the final solution F. Lemma 2 shows that $|A_d| \leq 2d|I_d|$ and Lemma 4 shows that $|B_d| \leq \frac{n-2}{d-1}$. Therefore since $|S| \geq \max\{1, |I_d|\}$ we have:

$$\frac{OPT}{|S|} \leq \frac{|A_d| + |B_d|}{|S|} \leq \frac{2d|I_d| + \frac{n-2}{d-1}}{\max\{1, |I_d|\}} \leq 2d + \frac{n}{d-1}.$$

Let $d = \sqrt{n/2}$. Then we have:

$$|S| \geq \frac{OPT}{2\sqrt{2n}}.$$

Finally note that in the **Greedy Star-Insertion Algorithm** each edge is considered at most twice, once for each one of its incident vertices, and each vertex is considered once. Also note that for each edge considered by the algorithm a constant number of *Union* and *Find* operations are invoked. This therefore implies the bound on the running time. The first sorting step can be done in linear time, as we are only sorting degrees which are integral valued in the range $1, \ldots, n$. □

Lemma 2: (Bound on Low Degree Vertices) $|A_d| \leq 2d|I_d|$, where A_d (I_d) is the set of full degree vertices of degree $\leq d$ in the optimal solution (the tree F output by the algorithm).

Proof: Let us first delete all the common vertices from A_d and I_d. Now we have $A_d \cap I_d = \emptyset$. We will prove the lemma for these new sets without any common vertices. Note that this will imply that the lemma also holds for the original sets as well.

For ease of presentation we drop the subscripts on A_d and I_d in the following proof.

Note that $|E_I| \leq d|I|$ and that both $G_A = (V, E_A)$ and $G_I = (V, E_I)$ are forests (acyclic graphs).

Let $G_{IA} = G_A \oplus I$, that is G_{IA} is obtained from G_A by adding all the edges incident on vertices in I ($G_{IA} = G_A \oplus E_I$). Since both G_I and G_A are forests, there must exist a set of edges $E'_A \subseteq E_A \setminus E_I$ and $|E'_A| \leq |E_I|$ such that $G_{IA} \ominus E'_A$ is a forest. Since $E'_A \cap E_I = \emptyset$, then by the definition of E_I none of the edges in E'_A is incident on any vertex in I. Therefore, if A' is the set of vertices each of which is incident on some edge in E'_A, then $A' \cap I = \emptyset$. Therefore by our construction, all vertices in I and $A \setminus A'$ have full degree in $G_{IA} \ominus E'_A$ and in addition $G_{IA} \ominus E'_A$ is a forest. But if $\exists v \in A \setminus A'$ then $v \notin I$ (since $I \cap A = \emptyset$) and v has degree $\leq d$. Therefore when the **Greedy Star-Insertion Algorithm** executes its outer **For** loop (for each vertex v_i) with $S = I$, it must add one more vertex of degree at most d to S, a contradiction since I is the set of all vertices of degree at most d in S. Therefore $A \setminus A' = \emptyset$ or $A \subseteq A'$. Note that by the definition of A' ($A' = \{u|(u,v) \in E'_A\}$), we have $|A'| \leq 2|E'_A|$. This is because an edge is incident on two vertices. But $|E'_A| \leq |E_I| \leq d|I|$. Hence $|A| \leq |A'| \leq 2d|I|$. □

The following corollary follows easily from Lemma 2.

Corollary 3: For graphs with degree bounded by d the **Greedy Star-Insertion Algorithm** delivers a solution of size at least $\frac{OPT}{2d}$ where OPT is the size of the optimal solution.

Lemma 4: (Bound on High Degree Vertices) $|B_d| \leq \frac{n-2}{d-1}$, where B_d is the set of full degree vertices of degree $> d$ in the optimal solution.

Proof: For ease of presentation we drop the subscript on B_d in the following proof.

Note that $G_B = (V, E_B)$ must be a forest. For each vertex $v \in B$, let $OUT(v)$ be the number of edges incident to v whose other end point is not in B. Let $IN(v)$ be the number of edges incident to v whose other end point is in B. Since for each such v, we have $IN(v) + OUT(v) \geq d$, by summing we obtain $\sum_{v \in B}(IN(v) + OUT(v)) \geq d|B|$. Let E'_B be the edges between vertices in B. On the left hand side of the summation, the edges in E'_B are counted twice, and the edges in

$E_B - E'_B$ are counted once. We thus obtain $|E_B| + |E'_B| \geq |B|d$. Since $|E'_B| \leq |B| - 1$, we obtain $|B| - 1 + (n-1) \geq |B|d$. Simplifying gives the bound of $|B| \leq \frac{n-2}{d-1}$. □

2.2 Lower Bounds on Approximation Factor for FDST

Let the input graph be $G = (V, E)$ and let G have n vertices. We show that it is not possible to design an $O(n^{(\frac{1}{2}-\epsilon)})(\epsilon > 0)$ approximation algorithm for the FDST problem unless $coR = NP$.

This shows that our approximation algorithm is almost optimal assuming that $coR \neq NP$. In addition our lower bound results hold for bipartite graphs.

We establish the lower bound by a linear reduction from the independent set problem to the FDST problem. It is known that no polynomial time $O(n^{1-\epsilon})(\epsilon > 0)$ approximation algorithm exists for the independent set problem, unless $coR = NP$ [5].

Theorem 5: No $O(n^{(\frac{1}{2}-\epsilon)})(\epsilon > 0)$ approximation algorithm exists for the FDST problem, unless $coR = NP$.

Proof: Given a graph H, an input instance of the independent set problem (we will assume w.l.o.g that H has at least two edges, and that there are no isolated vertices in H), we create an instance graph G of the FDST problem as follows. We also assume that the maximum independent set in H has size at least three. G can be viewed as a four layer graph whose edges only connect vertices in adjacent layers. Hence G is bipartite. Layer one consists of just one vertex a. Layer two has one vertex for every vertex of H, and every vertex in the second layer is connected to a. Layer three has one vertex for every edge of H, and if (u, v) is an edge in H then the corresponding vertex in the third layer is connected to the vertices in layer two, corresponding to the vertices u and v of H. Finally layer four has two vertices b and c, which are both connected to every vertex in the third layer.

Let T be a feasible solution to the FDST problem for graph G. First note that if any two

vertices have at least two common neighbors in G then they both cannot have full degree in T. Hence only one vertex from among $\{b, c\}$ has full degree in T. This is because H has at least two edges and hence b and c have at least two common neighbors. Similarly, at most one vertex in the third layer of G has full degree in T. This is because both b and c are in the neighborhood of every vertex in the third layer. If vertex a has full degree in T then none of the vertices in the third layer of G have full degree in T, since these vertices have two common neighbors in layer two. Finally note that all vertices in the second layer with full degree in T must form an independent set in H.

The above implies that if H has an independent set of size i then there is a feasible solution of size $i+1$, to the FDST problem, for the graph G. In this solution vertex a, and the vertices in the independent set have full degree. Similarly, if there is a feasible solution to the FDST problem for the graph G of size $i+1$, then if b or c has full degree (only one of them can be a full degree vertex) then no pair of vertices in the second layer can have full degree. To see this note that if v and v' are two vertices in the second layer that have full degree, then since they have edges to vertex a, and since each one of them has an edge to a vertex in layer three (corresponding to the edge incident on these vertices in H), which are both connected to either vertex b or c. By the assumption that one of the vertex b or c has full degree, this would thus imply the presence of a cycle in the solution to the FDST, a contradiction. We cannot pick more than one full degree vertex in layer three in any case, and also we can only pick vertex a or a vertex in layer three of full degree, but not both. Therefore we will be able to pick at most three vertices of full degree. Hence at least i vertices in layer two have full degree in this feasible solution and therefore H has an independent set of size i.

By our reduction an α-approximation algorithm for the FDST problem implies an α-approximation algorithm (up to an additive term) for the independent set problem. Let N be the number of vertices in H. Then the lower bound for the independent set problem [5] establishes that $\alpha = \Omega(N^{(1-\epsilon)})(\epsilon > 0)$ unless $coR = NP$. Note that G has $n = O(N^2)$ vertices where N is the

number of vertices in H. This therefore yields the claimed lower bound on α in terms of n. □

3 Absolute Size of the Solution to the FDST Problem

In this section we show that we can always find a solution to the FDST problem of size $\Omega(n/\Delta^2)$, where Δ is the maximum degree in the input graph G. In addition we give an example of a graph, with maximum degree Δ, for which the size of any solution to the FDST problem is $O(n/\Delta^2)$.

Let G^2 be the graph obtained from G by adding additional edges between those vertices of G that have a common neighbor in G. Note that any two vertices that belong to an independent set of G^2, do not have a common neighbor in G. Hence the set of edges of G incident on the vertices in any independent set of G^2 do not contain any cycles. Hence, the set of vertices in any independent set of G^2 form a solution to the FDST problem on the graph G. Observe that we can pick a maximal independent set in G^2 of size $\Omega(n/\Delta^2)$. Therefore we can always find a solution to the FDST problem, of size $\Omega(n/\Delta^2)$. The following example shows that in general, the biggest possible solution to the FDST problem will be of size $O(n/\Delta^2)$.

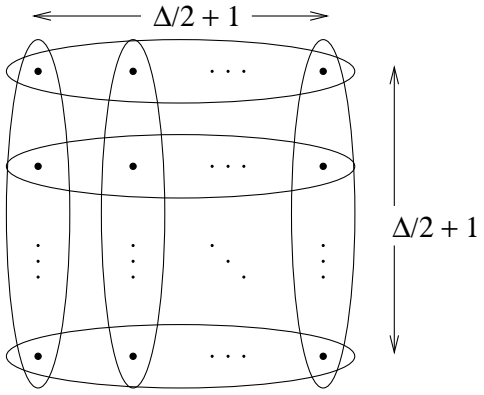

Figure 2: Tight example for the size of the FDST solution.

For our example we use the graph shown in Fig. 2 ($\Delta \geq 4$). The vertices of this graph can be thought of as the points on a grid of size $(\Delta/2+1) \times (\Delta/2+1)$: there are $\Delta/2+1$ vertices in any horizontal strip and there are $\Delta/2+1$ vertices in any vertical strip. Thus $n \approx \Delta^2/4$. For

every horizontal (vertical) strip there is a horizontal (vertical) clique that connects the vertices of the strip. Thus, every vertex is in two cliques, each of size $\Delta/2 + 1$: one vertical and the other horizontal. Every vertex thus has degree Δ. Clearly no two vertices that belong to a common clique can be of full degree in the FDST, since any three vertices in a clique are connected in a cycle. Also, every pair of vertices that are not in a common clique have two common neighbors, one in a horizontal clique and one in a vertical clique. Thus the maximum number of full-degree vertices in an FDST on this graph is 1. Clearly, this can be extended for larger values of n by duplicating this structure.

Note: For planar graphs, there are excellent (absolute) bounds on the size of independent sets obtained by the greedy algorithm (see Papadimitriou and Yannakakis [10]). Perhaps such bounds can be obtained for independent sets in the square (G^2) of bounded degree planar graphs? (If the degree is unbounded, then $K_{2,n-2}$ is an example of a planar graph whose square is a clique, and there is only one vertex of full degree.)

4 Optimal Solutions and Experimental Results

We implemented the Pothof and Schut heuristic [11], as well as our Greedy Star-Insertion heuristic and compared the performance of the two heuristics. We refer to the heuristics as PS and $BKPS$ respectively. In order to experimentally determine the empirical efficiency of the Greedy Star-Insertion Algorithm, it is necessary to find the optimal solution. We give an integer program that is feasible to run for small graphs, up to about 30 nodes with arbitrary connectivity, and then we give an algorithm that is especially efficient for larger, dense graphs.

4.1 Integer Program Formulation

$$\forall e, x_e \in \{0,1\}, \quad x_e = 1 \text{ iff } e \in \text{ Spanning Tree}$$

$$\forall v, f_v \in \{0,1\}, \quad f_v = 1 \text{ iff } v \text{ has full degree}$$

$$\begin{array}{rc} \text{maximize} & \sum_{v \in V} f_v \\ \text{with constraints:} & \sum_e x_e = |V| - 1 \\ \forall v, \forall\ e \text{ incident on } v, & f_v \leq x_e \end{array}$$

This integer program, as written, does not ensure that the set of edges with $x_e = 1$ forms a spanning tree. To enforce this condition, the Integer Program must be augmented with constraints that enforce connectivity. This can be implemented by arbitrarily defining a root node, creating a commodity for each vertex, and enforcing that there is a valid flow, along the spanning tree edges, from each node to the root. If the set of edges for which $x_e = 1$ has size $|V| - 1$ and connects every vertex to an arbitrary vertex, it must be a spanning tree.

4.2 Dense Graphs

A different algorithm can find the optimal solution efficiently for dense graphs. No pair of nodes can simultaneously have full degree if they have at least two paths of length at most two between them. Thus, in dense graphs, choosing one node as having full degree immediately eliminates the possibility for many other nodes to have full degree. This leads to a recursive algorithm which we outline as follows (see Fig. 3). Initially color all nodes gray. Start each recursive iteration by choosing any gray vertex v, which may or may not be a full degree node in the OPT solution. We

follow both possibilities. In case one, we assume v does not have full degree, we color v black, and recursively choose another gray node to continue. In the other case v does have full degree, we color v white, and then color black all the nodes in the graph which can now no longer have full degree in the spanning tree. The point is, while the running time of this algorithm is exponential, it grows slowly for dense graphs, because we get to color many nodes black each time we postulate a node as having full degree.

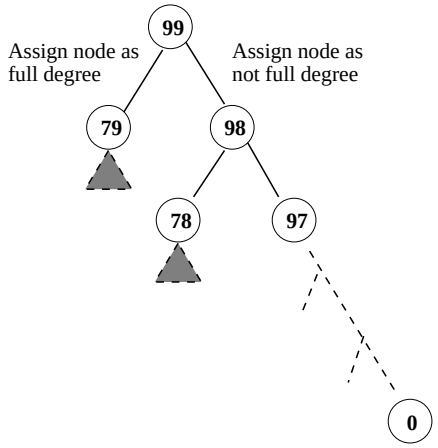

Figure 3: The recursive call structure for dense graphs – the number inside the circle is the number of unassigned nodes, a measure of how much work the algorithm has left to do. For the instance shown here, making one node full degree prohibits 20 others, leading to a slow exponential growth.

4.3 Experiments

Preliminary testing has been done on various kinds of graphs generated in LEDA [9] and Stanford Graphbase [7] (random planar and non-planar graphs) of sizes ranging from 10 vertices to 200 vertices. Figure 4 displays the results of tests on graphs with 100 vertices for various edge densities.

The first chart shows the performance of the two algorithms on random planar graphs of 100 vertices. The second chart shows the performance on random graphs of 100 vertices. The x axis records the number of edges in the graph, and the y axis records the number of full degree nodes that we find.

For small graphs we were able to compare the algorithm solution with the optimal solution, computed as described above. For random planar graphs only in *one case* (out of about 35 tests) was there a difference between the optimal solution and $BKPS$, and that too by only one vertex. For arbitrary random graphs (out of 25 tests), for half the tests $BKPS$ obtained an optimal solution, and for the other half of the tests the optimal solution had one more vertex (in one case, $BKPS$ missed by two vertices). The $BKPS$ algorithm can be further improved in two ways. The degree of a node may be viewed as a measure of "how many vertices of full degree are possibly prevented from being picked". The "effective" degree of a node is computed as if the edges of the star were deleted from the graph; the performance is improved by dynamically updating the degree ordering of the nodes. An additional strategy that uses local search to improve the final solution of $BKPS$ is discussed in [6].

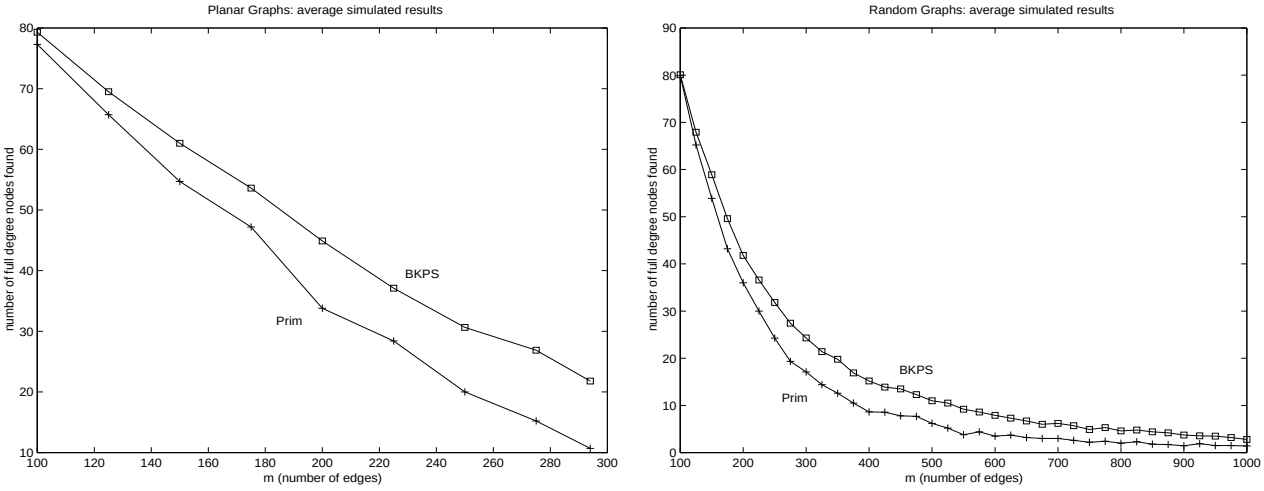

Figure 4: Experimental comparison of new approximation algorithm, $BKPS$, and previous heuristic PS, for random planar and random arbitrary graphs.

Acknowledgments: We thank Jan Schut for sending us a copy of [11] and Sudipto Guha for useful comments.

References

[1] R. Bhatia, S. Khuller, R. Pless and Y. Sussmann, "The full degree spanning tree problem", Tech. Report CS-TR-3931, Univ. of Maryland, (1998).

[2] H.J. Broersma, A. Huck, T. Kloks, O. Koppius, D. Kratsch, H. Müller, H. Tuinstra, Degree-preserving forests, *Networks*, 35 (1) 2000, 26–39.

[3] G. Galbiati, A. Morzenti and F. Maffioli, On the approximability of some maximum spanning tree problems, *Theoretical Computer Science*, Vol 181(1), 1997, pp. 107–118.

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

[5] J. Håstad, Clique is hard to approximate within $n^{1-\epsilon}$, *37th Annual Symposium on Foundations of Computer Science*, 1996, 627–636.

[6] S. Khuller, R. Bhatia, R. Pless, On local search and placement of meters in networks, *11th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2000, 319-328.

[7] Donald E. Knuth. The Stanford Graphbase. ACM/Addison Wesley, 1993.

[8] M. Lewinter, Interpolation theorem for the number of degree-preserving vertices of spanning trees, *IEEE Trans. Circ. Syst.*, CAS-34, (1987), 205.

[9] K. Mehlhorn and S. Näher, LEDA: A platform for combinatorial and geometric computing, *Communications of the ACM*, Vol 38 (1), 1995, 96–102. `http://mpi-sb.mpg.de/LEDA/leda.html`.

[10] C. H. Papadimitriou and M. Yannakakis, Worst-case ratios in planar graphs and the method of induction on faces, *22nd Annual Symposium on Foundations of Computer Science*, 1981, 358–363.

[11] I. W. M. Pothof and J. Schut, Graph-theoretic approach to identifiability in a water distribution network, *Memorandum No 1283*, Universiteit Twente 1995.

[12] R. E. Tarjan, Data Structures and Network Algorithms. Society for Industrial and Applied Mathematics, Philadelphia, 1983.